

ESCOLA POLITÉCNICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO  
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

PAULO HENRIQUE VANCIN

**SOFTWARE FRAMEWORK OF CONTROL SYSTEMS ON  
AN MPSOC PLATFORM**

Porto Alegre  
2023

PÓS-GRADUAÇÃO - *STRICTO SENSU*



Pontifícia Universidade Católica  
do Rio Grande do Sul

**PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL  
SCHOOL OF TECHNOLOGY  
COMPUTER SCIENCE GRADUATE PROGRAM**

**SOFTWARE FRAMEWORK OF  
CONTROL SYSTEMS ON AN  
MPSOC PLATFORM**

**PAULO HENRIQUE VANCIN**

Doctoral Thesis submitted to the  
Pontifical Catholic University of Rio  
Grande do Sul in partial fulfillment of  
the requirements for the degree of Ph. D.  
in Computer Science.

Advisor: Prof. Fernando Gehm Moraes

**Porto Alegre  
2023**

## Ficha Catalográfica

V222s Vancin, Paulo Henrique

Software Framework Of Control Systems on an MPSoC Platform /  
Paulo Henrique Vancin. – 2023.

234.

Tese (Doutorado) – Programa de Pós-Graduação em Ciência da  
Computação, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Control Systems. 2. Embedded Systems. 3. Heterogenous Computing. 4.  
Robotics. 5. MPSoC. I. Moraes, Fernando Gehm. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS  
com os dados fornecidos pelo(a) autor(a).

Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

**PAULO HENRIQUE VANCIN**

# **SOFTWARE FRAMEWORK OF CONTROL SYSTEMS ON AN MPSOC PLATFORM**

This Doctoral Thesis has been submitted in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science, of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul

Sanctioned on January 19<sup>th</sup>, 2023.

## **COMMITTEE MEMBERS:**

Prof. Aurélio Tergolina Salton (PPGEE/UFRGS)

Prof. Rafael Fraga Garibotti (EP/PUCRS)

Prof. César Augusto Missio Marcon (PPGCC/PUCRS)

Prof. Fernando Gehm Moraes (PPGCC/PUCRS - Advisor)

To my parents,

“Scientists have become the bearers of the torch of discovery in our quest for knowledge.”  
(Stephen Hawking)

## **AGRADECIMENTOS**

I would like to thank my first Thesis advisor, Prof. Alexandre de Morais Amory, for giving me this opportunity and guiding and supporting me all along this project. To my second Thesis advisor Prof. Ney Laert Vilar Calazans, that took me under his wing in the final stages of this project, providing me with constant support. To my friends at the GAPH laboratory: thanks for the laughs, advice, and support. And to my family, I could not have reached this goal without them.

Thank you all.

# ARCABOUÇO DE PROGRAMAÇÃO DE SISTEMAS DE CONTROLE SOBRE UMA PLATAFORMA MPSOC

## RESUMO

Com a crescente complexidade de sistemas robóticos, diversos aspectos de seu controle tornam-se desafiantes. O sensoriamento gera agregados de dados volumosos a coletar e processar; atuadores demandam a rápida manipulação de sinais, e controladores evoluem para usar algoritmos sofisticados. Tal progressão de demanda exige um aumento do poder de processamento. Novas tecnologias de processadores podem introduzir limites energéticos. Esses limites, implicam em reduzir que a totalidade de recursos intra-chip seja simultaneamente empregada no seu máximo de desempenho. Dada esta limitação física, alternativas são necessárias para aumentar o desempenho do hardware. Soluções apontam para o paralelismo e a computação heterogênea. Um sistema de computação heterogênea distribui dados, processamento e execução de programas em diferentes processadores. Esta Tese propõe aplicar sistemas de controle digital em sistemas de computação heterogênea e paralela para aumentar a eficiência do controle, permitindo adicionar múltiplas técnicas a este. Exemplos são a descentralização da arquitetura de controle, a auto adaptação do controlador, técnicas de tolerância à falhas e a gestão de energia. A Tese propõe um *arcabouço* de programação para implementar sistemas de controle sobre sistemas multiprocessados *on-chip* (MPSoCs). O arcabouço foi desenvolvido de forma genérica, visando servir a múltiplos artefatos robóticos. O estudo de caso de aplicação usado é um veículo aéreo não-tripulado quadrirrotor. Tal tipo de veículo possui uma dinâmica *rápida* e alta sensibilidade à falhas e demanda por gestão de energia e controladores poderosos. O arcabouço de software foi embarcado em um ambiente de simulação habilitado a simular tanto o MPSoC quanto o quadrirrotor. Um conjunto de experimentos valida a hipótese geral da tese. Eles testam itens como requisitos temporais, descentralização do controle, capacidade de realizar controles com baixo impacto no desempenho, tolerância à falhas e gestão de energia e adaptatividade dos controladores.

**Palavras-Chave:** Sistemas de controle, Sistemas embarcados, Computação heterogênea, Robótica, MPSoC.



# SOFTWARE FRAMEWORK OF CONTROL SYSTEMS ON AN MPSOC PLATFORM

## ABSTRACT

With the increasing complexity of robotic systems, many aspects of their control system architecture also become more complex. Sensing produces huge data aggregates to collect and process; actuators demand rapid signal manipulation, and controllers evolve to include highly complex algorithms. This progression in processing demand requires computing power to keep up. However, new processor technologies introduce power limits. These limits, implies that the dissipation of energy inside the *chip* prevents all its resources from being used simultaneously at their maximum performance rate. With this physical limitation, a distinct method is needed to continue to increase hardware performance. One way to deal with such high processing demands is through the use of heterogeneous computing. A heterogeneous computing system distributes data, processing, and program execution across different processors. The basic idea of this thesis is that the application of a digital control system in a heterogeneous computing system increases the efficiency of the controller, while allowing multiple techniques to be added to the control. Examples are the decentralization of the control architecture, the self-adaptation of the controller, fault tolerance techniques and the energy management. This thesis actually proposes a software *framework* for the implementation of control systems in a multiprocessor embedded system (MPSoC). This framework was developed in a generic way to serve multiple robotic artifacts. The application case study employed herein is a quadrotor unmanned aerial vehicle. Such an example of robotic equipment is considered due to its fast dynamics, its sensitivity to faults and its high demand for energy management and powerful controllers. The software framework was embedded in a simulation environment capable of simulating both the processor and the quadrotor. Six sets of experiments validate the general hypothesis of the thesis. These experiments tested the time requirements, the decentralization of the control architecture, the ability to process complex control algorithms without impacting the performance, the intra-chip fault tolerance, not forgetting the power management applications and the self-adaptation of controllers.

**Keywords:** Control systems, Embedded systems, Heterogeneous computing, Robotics, MPSoC.

## LIST OF FIGURES

1.1	Representations of: (a) control systems running in embedded systems and (b) the control of embedded systems. . . . .	26
1.2	Quaternion representation of a rigid body attitude. . . . .	28
1.3	Energy consumption profile of a sensor fusion task running in a processor core. . . . .	28
1.4	Accumulated energy consumption of a sensor fusion task running in a processor core. . . . .	29
1.5	Diagram representing the research workflow . . . . .	32
2.1	Block diagram representation of an open-loop control system. . . . .	35
2.2	Block diagram representation of a closed-loop control system. . . . .	35
2.3	Representation of a traditional implementation of a digital control system. . .	36
2.4	Diagram of a multicore architecture. Adapted from [Zah19]. . . . .	39
2.5	MPPA multicore architecture. Adapted from [DDAB <sup>+</sup> 13]. . . . .	40
2.6	Diagram of the big.LITTLE architecture. Adapted from [CECK12]. . . . .	41
2.7	Discrete CPU + GPU architecture. Adapted from [Aro12]. . . . .	42
2.8	NVIDIA Tegra K1 mobile processor (32-bit version). Adapted from [LSN14].	43
2.9	CPU + FPGA architecture. Adapted from [CCF <sup>+</sup> 19]. . . . .	44
2.10	A Block Diagram of the ZYNQ Architecture. Adapted from [MWH13]. . . . .	45
2.11	DSP-GPU heterogeneous computing system structure. Adapted from [LHL <sup>+</sup> 12]. . . . .	46
2.12	Snapdragon 800 block diagram. Adapted from [CAV <sup>+</sup> 14]. . . . .	47
2.13	Topics selected to start the state-of-the-art research . . . . .	48
2.14	Processor-in-the-loop test. Adapted from [KKSC18]. . . . .	49
2.15	HESSLE-FREE architecture. Adapted from [MMY <sup>+</sup> 19]. . . . .	50
2.16	A high level view of the software framework architecture. Adapted from [GKFF20]. . . . .	51
2.17	Experimental results showing the obstacles and paths taken by the robot in each case [GKFF20]. . . . .	52
2.18	The execution modes, roles and phases of the platform. Adapted from [USK19]. . . . .	53
2.19	Architecture of $\pi$ -RT and implementation of $\pi$ -RT in the Snapdragon 820. Adapted from [LTL <sup>+</sup> 21]. . . . .	54
2.20	Structure and workflow of ParNMPC. Adapted from [DO18]. . . . .	55

2.21	Heterogeneous implementation on an ARM v8 SoC and results. Adapted from [TLG17]. . . . .	57
2.22	$\pi$ – Soc heterogeneous architecture and results. Adapted from [TLG17]. . . .	58
2.23	High-level overview of the RUC framework. Adapted from [MCDH17]. . . . .	59
2.24	Closed-loop view of the proposed all-digital coordinated energy-budget and energy-allocation system. Adapted from [ZCF20]. . . . .	60
2.25	MIMO design methodology for HMPs. Adapted from [MDM+18]. . . . .	61
2.26	The logical view of the proposed framework. Adapted from [ATR+17]. . . . .	63
2.27	Tegra X2 architecture diagram. Adapted from [LLZ+20]. . . . .	64
2.28	Multi-layer model predictive control framework that decouples slow energy optimization from dynamic power management based on heterogeneous real-time computing platform. Adapted from [JB20]. . . . .	66
2.29	Overview of the PVF system. Adapted from [GQC+20]. . . . .	67
2.30	LoPECS architecture. Adapted from [TLL+20]. . . . .	69
2.31	The LoPECS runtime design. Adapted from [TLL+20]. . . . .	70
3.1	Diagram of forces and movements of a Quadrotor. . . . .	79
3.2	Main simulators of quadrotors. . . . .	80
3.3	Quadrotor mesh-based model - Gazebo screenshot. . . . .	82
3.4	ROS file system level. Adapted from [JC18]. . . . .	83
3.5	Basic ROS architecture. Adapted from [PG19]. . . . .	84
3.6	Gazebo simulator running Hector quadrotor. . . . .	85
3.7	Diagram showing with the most commonly used controllers by quadrotors in the literature. . . . .	96
3.8	Block diagram of a PID controller in a feedback loop. . . . .	100
3.9	PID with Back-Calculation Anti-Windup Block Diagram. . . . .	101
3.10	Fuzzy system diagram. Adapted from [lan12]. . . . .	102
3.11	Fuzzy system inference engine. Adapted from [dRdS21]. . . . .	102
3.12	Representation of a defuzzification using COG method. . . . .	103
3.13	Center of gravity method - fuzzy set aggregation. . . . .	104
3.14	Example of two output fuzzy sets. . . . .	104
3.15	Aggregation of the fuzzy sets process. . . . .	105
3.16	Schematics of three membership functions. . . . .	105
3.17	Example of a fuzzy system membership functions. . . . .	106
3.18	PID controller structure with Fuzzy System for gains tuning. Adapted from [AMY13]. . . . .	107

3.19	Diagram of the fuzzy algorithm for PID gains determination. . . . .	109
3.20	Block diagram of an LQR control system. . . . .	110
3.21	Block diagram of an LQR control system with integral action. . . . .	111
3.22	The "moving horizon" approach of Model Predictive Control. Adapted from [GPM89]. . . . .	112
3.23	Chapter 3 research topics summary diagram. . . . .	119
3.24	Closed-loop system diagram with Chapter 3 research topics. . . . .	120
4.1	MPSoC Diagram of a 6×6 MPSoC instance and the internal structure of a processing element, typically composed of a processor, a local memory, a DMA/network interface module (DMNI) and a network router. Adapted from [PUC19]. . . . .	122
4.2	HellfireOS Structure. Adapted from [AH12]. . . . .	125
4.3	The organization of ORCA platform, depicting software, hardware, and tools. Adapted from [D <sup>+</sup> 20]. . . . .	125
4.4	The MPSoC comprising four different nodes. The top-left most node is a networking node, while the others are processing nodes. Nodes are interconnected through a network-on-chip, namely Hermes. Adapted from [DJSFA19].	126
5.1	Overall representation of the proposed system. . . . .	132
5.2	Experimental Setup. . . . .	133
5.3	Data-flow of the MPSoC integration into a ROS system. . . . .	134
5.4	Algorithm implementation within the ORCA platform. . . . .	135
5.5	Application implementation of the EKF algorithm. . . . .	135
5.6	Euler angles representation. Adapted from [JCH <sup>+</sup> 17]. . . . .	145
5.7	A block diagram shows the cascade control strategy. . . . .	151
5.8	Block diagram representation of a generic estimator . . . . .	154
5.9	Block diagram of the LQR controller with a kalman filter estimator. . . . .	155
5.10	Basic concept for model predictive control. . . . .	156
6.1	PID - EKF Implementation Diagram. . . . .	159
6.2	Results of Experiment 1.A - Graphical comparison between the EKF estimation of the attitude quaternion and the real value presented by the simulation.	160
6.3	Results of Experiment 1.B - Height measured in meters representing the output of PID controller. . . . .	161
6.4	Energy estimation results by algorithm with calculations made with fixed-point format. With and without hardware multiplier unit. . . . .	162
6.5	Energy estimation results by algorithm with calculations made with software-emulated floating-point format. With and without hardware multiplier unit. . .	163

6.6	Results of Experiment 1.C - Graphical comparison between the XY position estimation by the KF algorithm, the GPS measurements and the simulated real position. . . . .	164
6.7	EKF + KF Centralized Processing Implementation Diagram. . . . .	165
6.8	Results of Experiment 2.B.A - Graphical comparison between the EKF estimation of the attitude quaternion and the real value presented by the simulation, utilizing a centralized framework. . . . .	166
6.9	Results of Experiment 2.B.C - Graphical comparison between the XY position estimation by the KF algorithm, and the simulated real position, utilizing a centralized framework. . . . .	167
6.10	Results of Experiment 2.B.B - Height measured in meters representing the output of PID controller, utilizing a centralized framework. . . . .	168
6.11	Energy estimation results by algorithm within cores. . . . .	168
6.12	Overall time diagram for centralized processing configuration. . . . .	169
6.13	EKF + KF Decentralized Processing Implementation Diagram. . . . .	170
6.14	Results of Experiment 2.C.B - Graphical comparison between the EKF estimation of the attitude quaternion and the real value presented by the simulation, utilizing a decentralized framework. . . . .	171
6.15	Results of Experiment 2.B.C - Graphical comparison between the XY position estimation by the KF algorithm, and the simulated real position, utilizing a centralized framework. . . . .	172
6.16	Results of Experiment 2.C.A - Height measured in meters representing the output of PID controller, utilizing a decentralized framework. . . . .	172
6.17	Energy estimation results by algorithm within cores. . . . .	173
6.18	Overall time diagram for decentralized processing configuration. . . . .	174
6.19	LQR/MPC Control Implementation Diagram. . . . .	175
6.21	Results of Experiment 3.A.A - Height measured in meters representing the output of LQR controller. . . . .	175
6.20	Results of Experiment 3.A.A - Graphical representation of the measured quaternion of the LQR controller output. . . . .	176
6.22	Results of Experiment 3.A.A - Graphical representation of the measured quaternion of the MPC controller output. . . . .	177
6.23	Results of Experiment 3.B.A - Height measured in meters representing the output of MPC controller. . . . .	177
6.24	Energy estimation results by algorithm within cores. . . . .	178
6.25	Controller Migration Implementation Diagram. . . . .	179

6.26	Diagram of Experiment 4.A, where two cores run PID tasks in parallel. . . . .	180
6.28	Results of Experiment 4.A - Height measured in meters representing the output of PID controller running in parallel cores. . . . .	180
6.27	Results of Experiment 4.A - Graphical representation of the measured quaternion of the PID controller the running in parallel cores. . . . .	181
6.29	Energy estimation results by algorithm within cores. . . . .	181
6.30	Diagram of Experiment 4.B, where two cores run PID tasks with no context.	182
6.31	Results of Experiment 4.A - Graphical representation of the measured quaternion of the PID controller the running in separate cores with no context migration. . . . .	183
6.32	Results of Experiment 4.B - Height measured in meters representing the output of PID controller running in separate cores with no context migration.	183
6.33	Energy estimation results by algorithm within cores with no context migration.	184
6.34	Diagram of Experiment 4.C, where two cores run PID tasks with context. . . . .	184
6.35	Results of Experiment 4.A - Graphical representation of the measured quaternion of the PID controller the running in separate cores with context migration. . . . .	185
6.36	Results of Experiment 4.A - Height measured in meters representing the output of PID controller running in separate cores with context migration. . . . .	185
6.37	Energy estimation results by algorithm within cores with no context migration.	186
6.38	Energy Management Implementation Diagram. . . . .	187
6.39	Results of Experiment 5.A - Graphical representation of the measured quaternion of the PID controller the running in energy saving mode while hovering. . . . .	188
6.40	Results of Experiment 5.A - Height measured in meters representing the output of PID controller running in energy saving mode while hovering. . . . .	188
6.41	Results of Experiment 5.B - Graphical representation of the measured quaternion of the PID controller the running in maximum energy mode while hovering. . . . .	189
6.42	Results of Experiment 5.B - Height measured in meters representing the output of PID controller running in maximum energy mode while hovering. . . . .	190
6.43	Results of Experiment 5.C - XY position control by the PID running in energy saving mode while in motion. . . . .	190
6.44	Results of Experiment 5.D - XY position control by the PID running in maximum energy mode while in motion. . . . .	191
6.45	Accumulated energy estimated in the core running the PID task in energy saving mode. . . . .	192

6.46 Accumulated energy estimated in the core running the PID task in maximum energy mode. . . . . 192

6.47 Controller Online Adaptation Implementation Diagram. . . . . 193

6.48 XY position control by the PID running with a parallel Fuzzy controller. . . . . 194

6.49 Calculated KP gains for PID controller by the Fuzzy controller running in parallel to the PID task. . . . . 194

## LIST OF TABLES

2.1	State-of-the-art Analysis Table . . . . .	74
2.2	State-of-the-art Analysis Table (Continuation) . . . . .	75
3.1	Physical parameters of the quadrotor system. . . . .	93
3.2	Example of a fuzzy system rule table. . . . .	103
3.3	Fuzzy rules for $K_p$ and $K_i$ . . . . .	107
3.4	Fuzzy rules for $K_d$ . . . . .	108
4.1	RISC-V base instruction sets and extensions. Adapted from: [Kan16]. . . . .	124
4.2	RTL simulation results obtained from instruction set classes. Adapted from [M <sup>+</sup> 18]. . . . .	128
4.3	Power characterization results and energy estimation for each instruction class of the processor (65nm), at 1.1V, 25° C (T=4ns). Adapted from [M <sup>+</sup> 18].	128
4.4	Router Average Power, at 1.1V, 25° C (T=4ns). Adapted from [M <sup>+</sup> 18]. . . . .	129
4.5	CACTI-P Report for a Scratchpad Memory, at 1.1V, 25° C (T=4ns). Adapted from [M <sup>+</sup> 18]. . . . .	130
4.6	Counters available for energy estimation in ORCA. Adapted from [D <sup>+</sup> 20]. . . . .	130
5.1	Computational setup specifications. . . . .	133
5.2	Fixed-point functions used in this Thesis. . . . .	143
6.1	Response time and energy evaluation for PID and EKF tasks. . . . .	163
6.2	Table of mean errors of the XY position estimation made by the KF algorithm and the GPS measurements. . . . .	165
6.3	Table of mean errors of the quaternion estimation made by the EKF algorithm and the simulated real attitude quaternion. . . . .	166
6.4	Table of mean errors of the XY position estimation made by the KF algorithm and the simulated real position, utilizing a centralized framework. . . . .	167
6.5	Table of the results of energy estimation results by algorithm within cores. . . . .	169
6.6	Table of mean errors of the quaternion estimation made by the EKF algorithm and the simulated real attitude quaternion. . . . .	170
6.7	Table comparing the mean errors of the quaternion estimation made by the EKF algorithm in both configuration. . . . .	170
6.8	Table of mean errors of the XY position estimation made by the KF algorithm and the simulated real position, utilizing a decentralized framework. . . . .	171
6.9	Table of the results of energy estimation results by algorithm within cores. . . . .	173



## LIST OF ALGORITHMS

3.1	KF Algorithm. . . . .	87
3.2	EKF Algorithm. . . . .	89
5.1	Attitude Estimation EKF. . . . .	147
5.2	XY Position Estimation KF. . . . .	149
5.3	Generic PID used for the height and attitude parameters. . . . .	150
5.4	LQR gain with integral action used for the control the height and attitude parameters. . . . .	153
5.5	Kalman Filter Gain Algorithm for full-state estimator. . . . .	154
5.6	Algorithm For Augmented System Build. . . . .	156
5.7	Algorithm For Auxiliary MPC Build. . . . .	157
5.8	Algorithm MPC Controller. . . . .	157
5.9	Algorithm for PID gain output. . . . .	157
A.1	SET VALUES . . . . .	217
A.2	TRANSPOSED MATRIX . . . . .	217
A.3	SUM OF MATRICES . . . . .	217
A.4	SUBTRACTION OF MATRICES . . . . .	218
A.5	MULTIPLICATION OF MATRICES . . . . .	218
A.6	MATRIX MULTIPLICATION BY A SCALAR . . . . .	218
A.7	MATRIX DIVISION BY A SCALAR . . . . .	219
A.8	CREATE AN IDENTITY MATRIX . . . . .	219
A.9	CREATE A MATRIX OF ZEROS . . . . .	219
A.10	CREATE A MATRIX OF ONES . . . . .	219
A.11	CREATE A $3 \times 3$ SKEW-SYMMETRIC MATRIX . . . . .	220
A.12	BLOCK DIAGONAL CONCATENATION (2 by 2) . . . . .	220
A.13	CUSTOM MATRIX OF MATRICES . . . . .	221
A.14	DELETE ROW AND COLUMN OF A MATRIX . . . . .	221
A.15	DETERMINANT OF A MATRIX . . . . .	222
A.16	INVERSE OF A MATRIX . . . . .	222
A.17	COPY OF A MATRIX . . . . .	224
A.18	RANK OF A MATRIX . . . . .	224
A.19	CHECK SYMMETRY OF A MATRIX . . . . .	225
A.20	LU DECOMPOSITION . . . . .	226
A.21	GET A MATRIX DIAGONAL . . . . .	227
A.22	GET SECTION OF A MATRIX . . . . .	228
A.23	QR DECOMPOSITION . . . . .	228
A.24	MATRIX TO THE POWER OF $N$ . . . . .	228
A.25	TRACE OF A MATRIX . . . . .	229

A.26 EIGENVALUES OF A MATRIX .....	229
A.27 EIGENVALUES OF A MATRIX ( $2 \times 2$ ) .....	229
A.28 ZEROS ALL THE ELEMENTS BELOW THE MATRIX DIAGONAL .....	230
A.29 ZEROS ALL THE ELEMENTS ABOVE THE MATRIX DIAGONAL .....	230
A.30 2-NORM OF A MATRIX .....	231
A.31 DISCRETE-TIME ALGEBRAIC RICCATI EQUATION SOLVER .....	231
B.1 Algorithm for open left fuzzyfication .....	232
B.2 Algorithm for open right fuzzyfication .....	232
B.3 Algorithm for triangular fuzzyfication .....	232
B.4 Algorithm for open left defuzzyfication .....	232
B.5 Algorithm for open right defuzzyfication .....	232
B.6 Algorithm for triangular defuzzyfication .....	233
B.7 Algorithm for mathematical representation of the fuzzy rules .....	233
B.8 Algorithm for defuzzyfication output .....	234

## LIST OF ACRONYMS

AD – Analog-to-Digital  
API – Application Programmer Interfaces  
ASIC – Application-Specific Integrated Circuits  
BC – Base Controller  
CAPSC – Compute-Aware Physical System Controller  
COG – Center Of Gravity  
COTS – Commercial Off-The-Shelf  
CPI – Cycles Per Instruction  
CPLD – Complex Programmable Logic Device  
CPPC – Computer Power and Performance controller  
CPU – Central Processing Unit  
DA – Digital-to-Analog  
DM – Decision Module  
DMA – Direct Memory Access  
DMNI – Direct Memory Network Interface  
DOF – Degree-Of-Freedom  
DSP – Digital Signal Processors  
DVFS – Dynamic Voltage and Frequency Scaling  
ECS – Embedded Control Systems  
EKF – Extended Kalman Filter  
FO – fail-operational  
FPGA – Field-Programmable Gate Array  
FS – fail-safe  
FSM – Finite State Machines  
GPP – General-Purpose Processor  
GPS – Global Positional System  
GPUS – Graphical Processing Units  
GUC – Global Utilization Controller  
HCS – Heterogeneous Computing System  
HMP – Heterogeneous Multicore Processor  
IC – Integrated Circuit  
IMU – Inertial Measurement Unit

KF – Kalman Filter  
LIDAR – Light Detection And Ranging  
LLC – Last-Level Cache  
LQG – Linear Quadratic Gaussian  
LQR – Linear Quadratic Regulator  
MC – Mission Controller  
MOS – Metal-Oxide-Semiconductor  
MPC – Model-Predictive Controller  
MPSOC – Multiprocessor System-on-Chip  
MU – Multiplier Unit  
ODE – Ordinary Differential Equation  
NMPC – Nonlinear Model Predictive Control  
NOC – Network-on-Chip  
ORCA – self-adaptive multiprocessOR system-on-Chip plAtform  
OS – Operational System  
OSRF – Open Source Robotics Foundation  
PC – Personal Computer  
PDF – Probability Density Function  
PE – Processing Elements  
PID – Proportional-Integral-Derivative  
POC – Proof-of-Concept  
PWM – Pulse-Width Modulation  
QOS – Quality-of-Service  
ROS – Robotic Operating System  
RTL – Register Transfer Level  
RTOS – Real-Time Operating System  
RUC – Reliability-ware Utilization Control  
SA – Situation-Aware  
SE – Single Execution  
SEFP – Software-Emulated Floating-Point  
SIMD – Single Instruction Multiple Data  
SITL – Software-In-The-Loop  
SLAM – Simultaneous Localization And Mapping  
SOC – System-on-Chip

SWC – Sampling Window Controller  
UAV – Unmanned Aerial Vehicle  
UDP – User Datagram Protocol  
URSA – Micro ( $\mu$ ) Rapid-Simulation API  
VCS – Version Control System  
VHDL – VHSIC Hardware Description Language  
VLSI – Very Large Scale Integration  
VTOL – Vertical Take-Off and Landing  
WCET – Worst-Case Execution Time

# CONTENTS

<b>1</b>	<b>INTRODUCTION AND MOTIVATION</b>	<b>25</b>
1.1	MOTIVATIONAL EXAMPLE	27
1.2	THESIS HYPOTHESIS	29
1.3	THESIS GOALS	30
1.4	THESIS CONTRIBUTIONS AND ORIGINALITY	30
1.5	RESEARCH WORKFLOW	31
1.6	DOCUMENT STRUCTURE	32
<b>2</b>	<b>BACKGROUND AND STATE-OF-THE-ART</b>	<b>34</b>
2.1	BACKGROUND	34
2.1.1	CONTROL THEORY AND EMBEDDED SYSTEMS EARLY WORK	34
2.1.2	HETEROGENEOUS COMPUTING	37
2.2	STATE-OF-THE-ART	46
2.2.1	BULAT KHUSAINOV ET AL.	47
2.2.2	KASRA MOAZZEMI ET AL.	49
2.2.3	MICHAEL GIARDINO ET AL.	50
2.2.4	MARKUS ULBRICHT ET AL.	52
2.2.5	LIU LIU ET AL.	53
2.2.6	HAOYANG DENG AND TOSHIYUKI OHTSUKA	55
2.2.7	JIE TANG ET AL.	56
2.2.8	YU MA ET AL.	58
2.2.9	DAVIDE ZONI ET AL.	59
2.2.10	TIAGO MÜCK ET AL.	60
2.2.11	FARDIN ABDI ET AL.	62
2.2.12	WEI LI ET AL.	64
2.2.13	ZHENHUA JIANG ET AL.	65
2.2.14	YIMING GAN ET AL.	66
2.2.15	JIE TANG ET AL.	68
2.3	STATE-OF-THE-ART ANALYSIS	70
<b>3</b>	<b>THE QUADROTOR CASE STUDY</b>	<b>76</b>
3.1	INTRODUCTION TO QUADROTORS	76

3.2	QUADROTOR DYNAMICS .....	78
3.2.1	FLIGHT MODES .....	78
3.3	COMPUTATIONAL SIMULATIONS .....	80
3.3.1	HECTOR QUADROTOR .....	82
3.4	THE ROBOT OPERATING SYSTEM (ROS) .....	83
3.4.1	GAZEBO SIMULATOR .....	85
3.5	SENSORS .....	86
3.6	SENSOR FUSION .....	86
3.6.1	KALMAN FILTER .....	87
3.6.2	EXTENDED KALMAN FILTER .....	88
3.6.3	QUATERNION REPRESENTATION .....	89
3.7	STATE SPACE REPRESENTATION .....	91
3.7.1	LINEARIZATION .....	94
3.8	MOST COMMONLY USED CONTROLLERS IN QUADROTORS .....	96
3.8.1	LINEAR CONTROLLERS USED BY QUADROTORS .....	97
3.8.2	NONLINEAR CONTROLLERS USED BY QUADROTORS .....	97
3.8.3	LEARNING-BASED CONTROLLERS USED BY QUADROTORS .....	98
3.9	CHOSEN CONTROL METHODS AND QUADROTOR IMPLEMENTATION ....	99
3.9.1	PROPORTIONAL INTEGRAL DERIVATIVE CONTROL - PID .....	99
3.9.2	LINEAR QUADRATIC REGULATOR - LQR .....	108
3.9.3	MODEL PREDICTIVE CONTROL - MPC .....	111
3.10	OVERALL VIEW OF THE QUADROTOR STUDY CASE .....	119
<b>4</b>	<b>HARDWARE PLATFORM AND SUPPORTING RESOURCES .....</b>	<b>121</b>
4.1	MULTIPROCESSED SYSTEM-ON-CHIP .....	121
4.2	RISC-V PROCESSOR ARCHITECTURE .....	122
4.3	HELLFIREOS .....	123
4.4	ORCA PLATFORM .....	125
4.5	URSA SIMULATOR .....	127
4.6	PROCESSING ELEMENT ENERGY ESTIMATION .....	127
4.6.1	PROCESSOR CHARACTERIZATION .....	127
4.6.2	ROUTER CHARACTERIZATION .....	129
4.6.3	MEMORY CHARACTERIZATION .....	129
4.6.4	ORCA MONITORING .....	130

<b>5</b>	<b>FRAMEWORK IMPLEMENTATION AND APPLICATIONS</b>	<b>132</b>
5.1	TARGET ARCHITECTURE	132
5.1.1	EXPERIMENTAL SETUP	133
5.2	ROS - URSA INTERFACE	134
5.3	IMPLEMENTATION OF THE ALGORITHMS	134
5.4	C AND C++ LANGUAGE LIBRARIES	136
5.5	DESCRIPTION OF THE ALGORITHMS	144
5.5.1	ATTITUDE MEASUREMENT OF THE QUADROTOR BASED ON ACCELEROMETER AND MAGNETOMETER	144
5.5.2	EXTENDED KALMAN FILTER FOR ATTITUDE ESTIMATION	145
5.5.3	KALMAN FILTER FOR LINEAR POSITION ESTIMATION	147
5.5.4	PROPORTIONAL, INTEGRAL AND DERIVATIVE CONTROL (PID) - ATTITUDE AND HEIGHT CONTROL	149
5.5.5	PROPORTIONAL, INTEGRAL AND DERIVATIVE CONTROL (PID) - XY POSITION CONTROL	151
5.5.6	LINEAR QUADRATIC REGULATOR (LQR) WITH INTEGRAL ACTION - ATTITUDE AND HEIGHT CONTROL	151
5.5.7	KALMAN FILTER FOR FULL STATE ESTIMATION - HEIGHT AND ATTITUDE CONTROL	153
5.5.8	MODEL PREDICTIVE CONTROL (MPC) - HEIGHT AND ATTITUDE CONTROL	155
5.5.9	PID ONLINE UPDATE - FUZZY CONTROL	157
<b>6</b>	<b>EXPERIMENTS AND RESULTS</b>	<b>159</b>
6.1	EXPERIMENT 1 - EKF WITH PID IMPLEMENTATION - PROOF OF CONCEPT	159
6.1.1	EXPERIMENT 1.A - EKF PERFORMANCE	160
6.1.2	EXPERIMENT 1.B - PID PERFORMANCE	161
6.1.3	EXPERIMENT 1.C - FLOATING POINT $\times$ FIXED POINT	162
6.2	EXPERIMENT 2 - EKF/KF WITH PID IMPLEMENTATION - CENTRALIZATION $\times$ DECENTRALIZATION	164
6.2.1	EXPERIMENT 2.A - KF PERFORMANCE	164
6.2.2	EXPERIMENT 2.B - EKF + KF (CENTRALIZED PROCESSING)	165
6.2.3	EXPERIMENT 2.C - EKF + KF (DECENTRALIZED PROCESSING)	169
6.3	EXPERIMENT 3 - CAPABILITY OF MORE COMPLEX CONTROL IMPLEMENTATION EVALUATION	174
6.3.1	EXPERIMENT 3.A - LQR	175
6.3.2	EXPERIMENT 3.B - MPC	176



6.3.3	EXPERIMENT 3.C - LQR AND MPC - ENERGY ESTIMATION .....	178
6.4	EXPERIMENT 4 - FAULT TOLERANCE ON PID CONTROL .....	178
6.4.1	EXPERIMENT 4.A - PIDS RUNNING IN PARALLEL .....	179
6.4.2	EXPERIMENT 4.B - PIDS - NO CONTEXT MIGRATION .....	182
6.4.3	EXPERIMENT 4.C - PIDS - CONTEXT MIGRATION .....	184
6.5	EXPERIMENT 5 - ENERGY MANAGEMENT .....	186
6.5.1	EXPERIMENT 5.A - SAVING MODE/HOVERING .....	187
6.5.2	EXPERIMENT 5.B - MAX POWER MODE/HOVERING .....	189
6.5.3	EXPERIMENT 5.C - SAVING MODE/MOTION .....	190
6.5.4	EXPERIMENT 5.D - MAX POWER MODE/MOTION .....	191
6.5.5	EXPERIMENT 5 - ENERGY ANALYSIS .....	191
6.6	EXPERIMENT 6 - CONTROLLER ONLINE ADAPTATION .....	193
6.6.1	EXPERIMENT 6 - SAVING MODE/MOTION - WITH FUZZY CONTROL .....	193
6.7	FINAL REMARKS .....	194
<b>7</b>	<b>CONCLUSION AND FUTURE WORK .....</b>	<b>197</b>
7.1	FUTURE WORK .....	199
	<b>REFERENCES .....</b>	<b>201</b>
	<b>APPENDIX A – Matrix Library Algorithms .....</b>	<b>217</b>
	<b>APPENDIX B – PID - Fuzzy Library Algorithms .....</b>	<b>232</b>

## 1. INTRODUCTION AND MOTIVATION

Embedded systems are application-specific systems that contain hardware and software tailored for a particular task. They are generally part of a larger system, and employ at least one microprocessor [GS98]. The embedding of microprocessors into equipment such as consumer appliances started before the appearance of the personal computer and takes up most of the currently manufactured microprocessors. Embedded microprocessors are deeply ingrained into everyday life, more than any other electronic circuit [Hea02]. They can be found in automobiles, in the medical field, in industrial control systems, in entertainment electronics, and even in prosaic elements such as household appliances' remote controls [TAK06].

Some embedded systems are often used in life-critical situations, where reliability and safety are more important criteria than performance [ELLSV97]. Safety-critical systems are systems whose failure could result in loss of lives, significant property damage, or damage to the environment [Kni02].

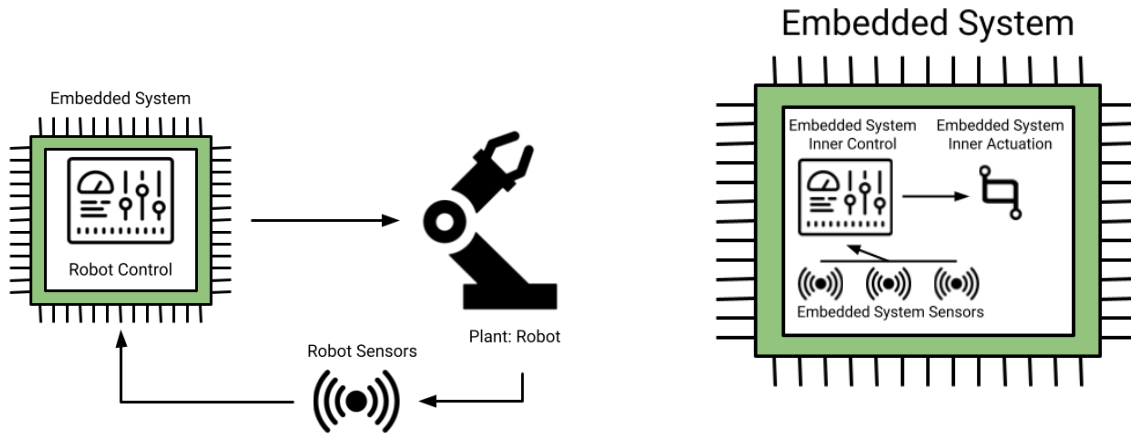
Modern technological systems heavily rely on sophisticated control systems to meet increased safety and performance requirements. To prevent fault-induced losses and to minimize potential risks of use, new control techniques and design approaches need to be developed to cope with system component malfunctions, while maintaining the desired degree of overall system stability, and performance levels [Jia05].

Control theory and embedded systems are typically treated separately in electrical engineering programs. The implementation of control algorithms in digital logic targets is often only indicated at the block diagram level [Ben20]. Concepts of control theory deal with the basic principles underlying the analysis and design of control systems. To *control* an object means to influence its behavior to achieve a desired goal. To implement this influence, engineers build devices that incorporate various mathematically defined techniques [Son13].

In reality, control theory and embedded systems are intertwined. A fundamental requirement of a control system is to complete all its assigned real-time tasks within design-time specified timing constraints (usually referred to as *deadlines*), even in the presence of faults [YDG04]. Embedded digital controllers are increasingly designed and built using commercial off-the-shelf (COTS) hardware and software components, due to the availability of high-performance, low-cost microcontrollers, specialized embedded processors, and various real-time operating systems (RTOSs) [Li05].

When referring to embedded systems and control theory, the merging of these topics creates two separate areas: control systems running in embedded systems and the control of embedded systems. Figure 1.1 illustrates both concepts. Control systems running in an embedded system, illustrated in Figure 1.1a, refer to an algorithm running in a computing element that controls an external system (external to the computing element). Meanwhile,

the control of embedded systems (see Figure 1.1b) refers to a computing element running a control algorithm that is responsible for managing one or more of its inner parameters, such as energy consumption, temperature control, quality-of-service (QoS), and real-time constraints of the computing element.



(a) Representation of a robotic control system running in an embedded system. (b) Representation of the control of an embedded system.

Figure 1.1: Representations of: (a) control systems running in embedded systems and (b) the control of embedded systems.

Control systems running in embedded systems and the control of embedded systems are the subject of extensive investigation. However, there are, for example, few studies on how the internal management of a computing element energy can affect a robotic application control. See Chapter 2 of this Thesis for a more extensive discussion and examples of fields where the integration embedded computing and control systems requires further research. One of the motivations of the work proposed herein is to bridge the communication gap between control systems running in embedded systems and the control of embedded systems. This motivation is further explained in Section 1.1.

While embedded systems are mostly associated with microprocessors and microcontrollers, they can encompass a variety of components employing other technologies such as digital signal processors (DSPs), complex programmable logic devices (CPLDs), application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), and graphical processing units (GPUs) [Arm10]. Furthermore, recent developments in high-speed digital communication make it possible to connect a distributed suite of these distinct high-performance components to provide a more powerful computing platform, often called a heterogeneous computing system [HJ03]. Heterogeneous systems can provide low-cost and high-performance computing whenever computational applications can be broken into tasks distributed to the various components for parallel execution [BH05].

As control theory and embedded system design are often seen as two distinct fields, most control applications in embedded systems are specifically implemented for some

given hardware architecture. Therefore, implementing control theory principles in an embedded system demands a particular knowledge, which the control engineer might not have or can cause the process of embedding the control to take a significant amount of time.

In this manner, it is noticeable a lack of a single, high-level control software framework aimed towards embedded systems in today's literature. Such a software framework can guarantee robust and stable control, and address computing machines' inner concerns, such as energy awareness, fault tolerance, and the fulfillment of real-time constraints.

The central statement behind this Thesis is that systematically applying digital control systems into a heterogeneous computing framework increases control efficiency and facilitates the opportunity to apply other techniques, such as decentralized control and controller self-adaptation. Also, this is expected to be achievable while taking into account parameters such as energy consumption, fault tolerance, and real-time constraints. The idea is to consider the digital control of dynamic systems and the control of computing components in a single unified paradigm. To demonstrate the validity of the proposed Thesis, a system composed of an heterogeneous computing environment is embedded in a Quadrotor Unmanned Aerial Vehicle (UAV), which is, by its own nature, a critical system.

## 1.1 Motivational example

It is evident in the literature that there is a mainly untapped area of research involving control theory: the adaptation of computational resources to benefit the controller implementation. The conventional way of implementing a digital controller for a dynamic system consists in defining a processing module and embed the theorized controller into the system. Later, the designer evaluates if the controller delivers an expected performance. This is usually done while ignoring other aspects of the computational environment.

Topics such as energy consumption, fault tolerance, and performance are generally treated as either a control theory or a computational problem, not as part of an intertwined system. To illustrate this point, consider the following example: a sensor fusion software is running in a single-core processor; this software is responsible for gathering data from inertial sensors and estimating a rigid body's attitude.

Figure 1.2 shows what a control engineer would look for, the performance of the software, evaluating if the quaternion representing the body's pose reflects the real-world representation with a satisfying response.

Figure 1.3 presents an estimation of the processor energy profile running the sensor fusion task for a pre-defined frequency. This is the information usually considered by computer scientists and/or computer engineers. Figure 1.4 shows the accumulated energy use throughout the process of running the task.

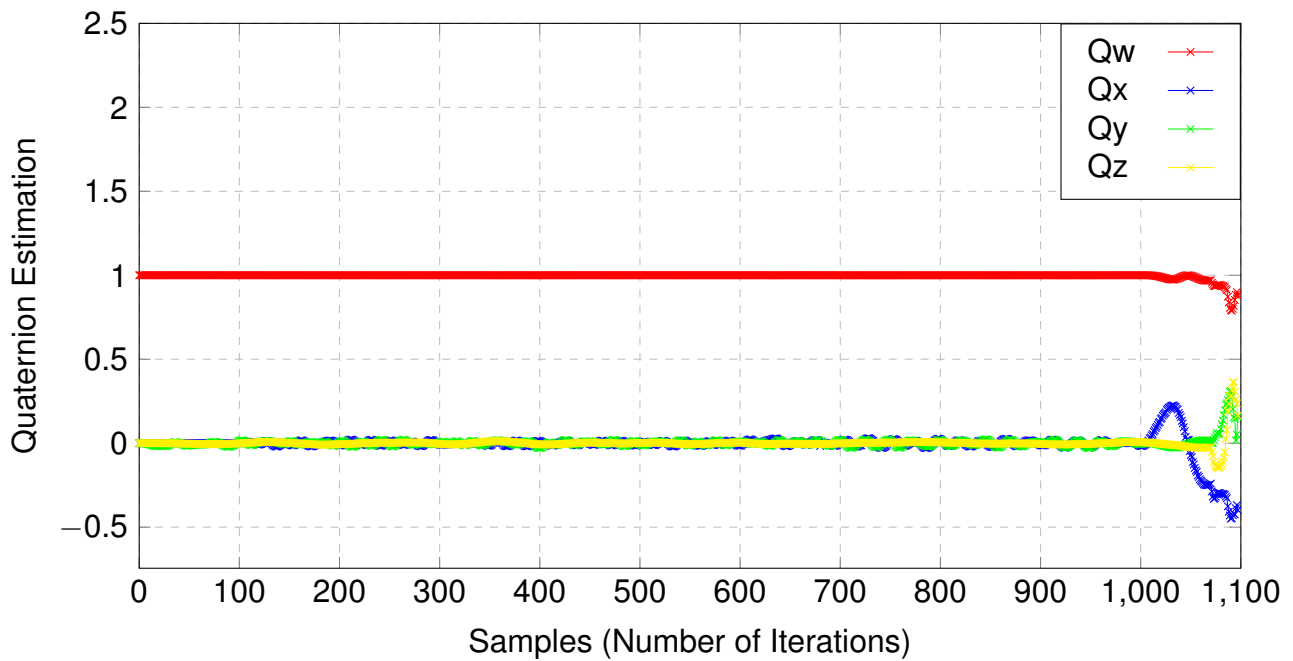


Figure 1.2: Quaternion representation of a rigid body attitude.

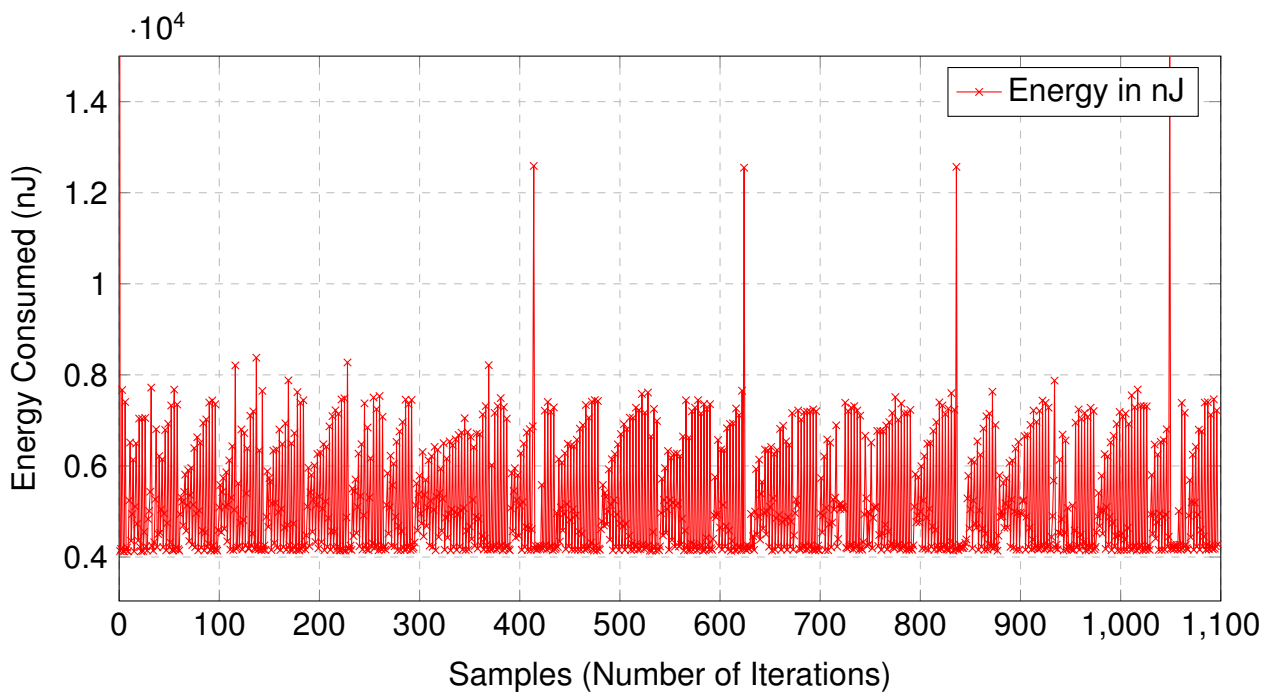


Figure 1.3: Energy consumption profile of a sensor fusion task running in a processor core.

However, what would happen to the precision of the attitude representation if the processing frequency was lowered in the interest of saving energy? Or, if it was decided to enhance performance by raising the frequency of operation? Practically, how is it possible to keep the result shown in Figure 1.2 and at the same time manage to decrease the slope of the curve presented in Figure 1.4.

In a real-time system, it is usually assumed that the computing machines are working at their maximum frequency to simplify the worst-case execution time (WCET) analysis,

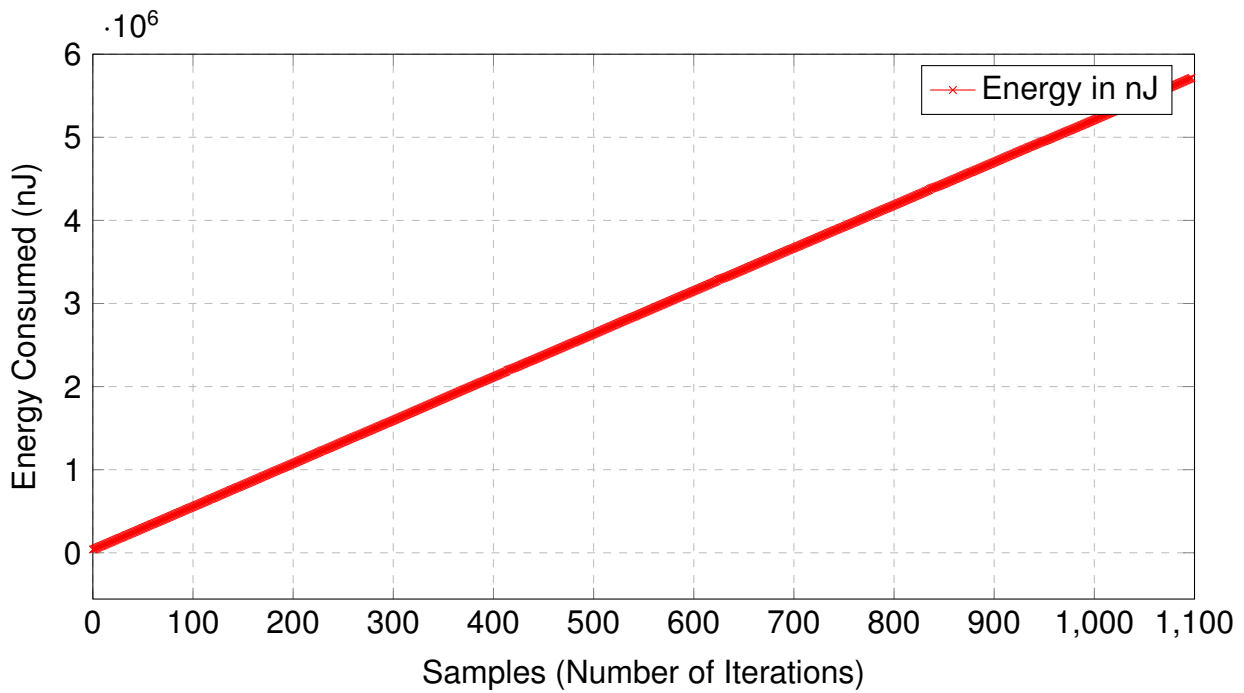


Figure 1.4: Accumulated energy consumption of a sensor fusion task running in a processor core.

to enable verifying that system deadlines are never missed. This practice can cause unintended losses elsewhere. More complex embedded systems that use Dynamic Voltage and Frequency Scaling (DVFS), when applied to real-time systems, the Operational System (OS) is set to the maximum frequency. Would not it be possible to consider the pairing of control theory and computer science to achieve an optimum result?

These topics are, mostly, unexplored.

## 1.2 Thesis Hypothesis

The central thesis hypothesis here is that using a heterogeneous computing framework aimed at control systems does not only yield better results than the usually implemented, traditional, centralized, single processing control systems, but it can simultaneously integrate different topics from research areas of control theory and embedded systems under an umbrella of a single software framework. Joining these topics together is an approach that is often overlooked in the literature.

On the one hand, issues like intra-chip energy management, fault tolerance, and other embedded systems topics can impact an implemented controller output. On the other hand, distributed computation and self-adaptive processing can positively influence the development of a control system. These two areas of study can mutually help each other and create opportunities for new and improved systems.

### 1.3 Thesis Goals

This Thesis's strategic goal is the proposition of in-depth research on the integration of two research fields: control theory and embedded systems. An expected outcome is to create a software framework for the implementation of control systems based on multi-processor systems-on-chip (MPSoCs).

The specific goals of the Thesis are as follows:

1. Develop a base software framework capable of implementing control systems into a multi-processed system on chip (MPSoC), based on an open source processor architecture;
2. Incorporate functional features related to control theory and computation into the proposed software framework. Example features that are to be included are decentralized control, online controller update, energy estimation, and control and fault detection and mitigation;
3. Link the proposed system to an operational system aimed at robotic applications;
4. Deploy the overall system into a robotic application, in this case, an UAV quadrotor, validating the Thesis propositions;
5. Compare the obtained results of the conducted tests and experiments with cases selected from the literature.

### 1.4 Thesis Contributions and Originality

The main original contributions of this Thesis are:

1. An integrated environment for the development of robotic applications targeting MP-SoCs. This environment eases the evaluation of non-functional requirements (such as energy and reliability) by combining cycle-accurate simulations from RTL models with behavioral simulations from robotics <sup>1</sup>;
2. An overall basic software framework written in the C programming language; This framework gathers a set of the most valuable functions for control applications, which allows the deployment of a large number of controllers for robotic applications;

---

<sup>1</sup>This environment was described in [VDP<sup>+</sup>20]

3. An environment that can evaluate and control the energy consumed by each processor core at runtime; this framework enables evaluating the effects of actuation in energy control, which impacts the output of the application control;
4. Application of a system capable of breaking apart a fully centralized control architecture into the proposed software framework system and running it in multiple cores, later comparing performance and efficiency;
5. Fault tolerance is another topic approached by the framework. In critical systems, a core failure can result in catastrophic consequences. This software framework proposes a task migration technique to guarantee a continuous and correct operation and also fulfilling energy consumption constraints;
6. Lastly, the decentralized nature of the MPSoC is used to create a control system that can be updated on-chip at runtime. The main application controller runs in a core, while in parallel, another task running in a different core analyses the output and updates the system.

## 1.5 Research Workflow

The research workflow, as depicted in Figure 1.5, comprises three distinct areas: literature review, implementation and evaluation.

The literature review divides the research into three sub-phases: the first consists of a state-of-the-art compilation of works regarding the core idea of this thesis: the union between heterogeneous computing systems and control theory. This phase takes the pre-defined robotic application (Quadrotor) in the second sub-phase, then research its main characteristics and related scientific works. The third sub-phase is realized by defining our overall system idea. This, takes what was considered the main ideas of the heterogeneous computing systems and control theory areas and define the target software framework architecture. This process is detailed in Chapters 2 and 3.

The Implementation phase build the target software framework, defining the necessary tools and subsystems. In the same way as the Literature Review, this area is divided into sub-phases. First phase defines computational and control system and the software basis used (C libraries and functions). Next phase ports this infrastructure into the processor's architecture chosen (RISC-V). The base simulation that validates the entire thesis is the third sub-phase, unifying the robotic simulation with the system. Lastly, all of the features devised by this thesis are implemented in this software framework. This process is detailed in chapters 4 and 5.



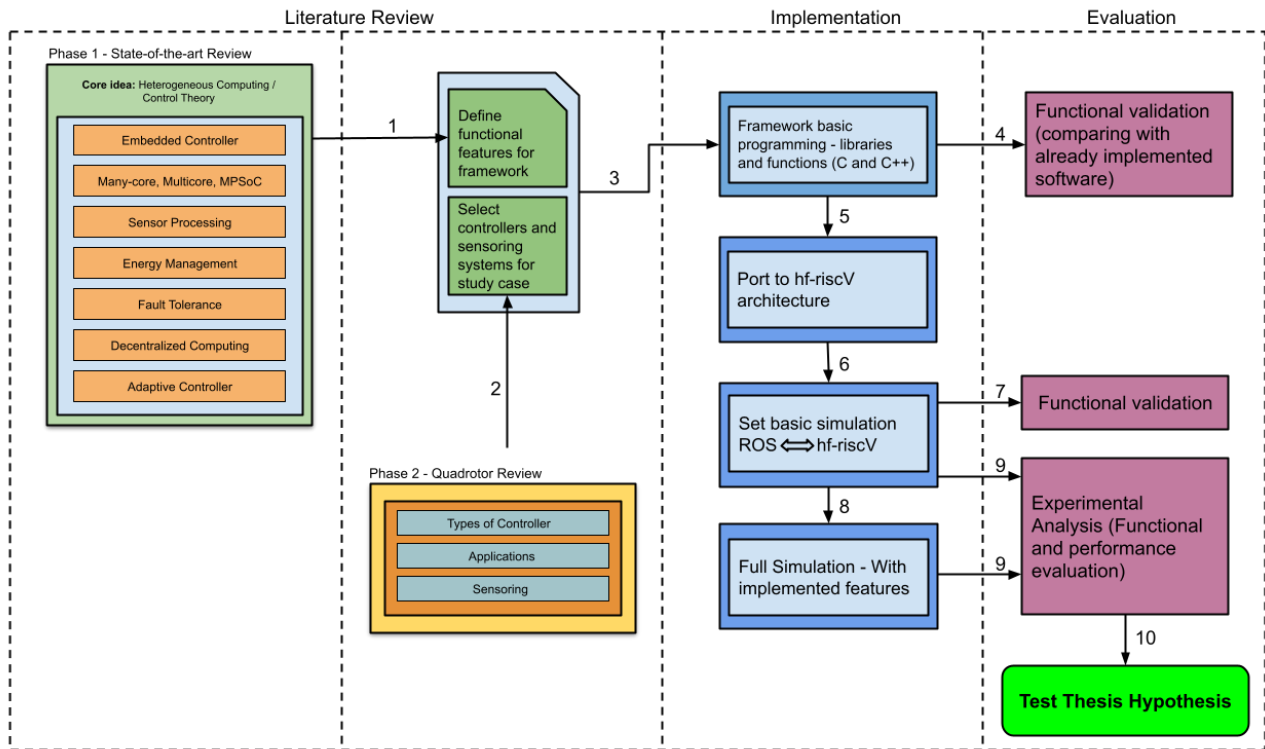


Figure 1.5: Diagram representing the research workflow

In parallel with the Implementation phase, the Validation phase evolves; as the development of the implementation progresses, the validation of the results are put in Chapter 6.

## 1.6 Document Structure

This document is structured in the following manner: Chapter 1 presents the motivation behind this thesis, present the research problem, define goals and explain its contribution and originality. Chapter 2 brings the literature review on which this thesis is based. Topics like the history of the application of control theory in embedded systems and heterogeneous computing characteristics and architectures are discussed. It also brings selected works currently being developed in related areas that are viewed as essential to contextualize this thesis (State-of-the-Art). Chapter 3 reviews all topics related to the quadrotor study case: presentation of the topics of physical dynamics, flight modes, mathematical modeling, most commonly used controllers, and available computational simulations of quadrotors. Still, Chapter 3, chooses three controllers to create our study case and apply them to the thesis. The theory behind these controllers is explained. Moreover, different theories necessary to develop this work are explained. Chapter 4 details the required infrastructure to run the software framework proposed. Presenting topics on MPSoCs, the RISC-V architecture,

the Hellfire OS, ORCA platform, URSA simulator, created C libraries, and energy estimation intrachip. Chapter 5, explains how the software framework is implemented and how the simulations are created. Also explains the entire system's architecture (Quadrotor simulation + MPSoC simulation), the experimental setup, and the implementation and description of the algorithms used in the control system. Chapter 6 contains the details and results of the experiments used to validate this thesis. All along, commenting and analyzing the results. At the end of this chapter, a recap of these results is brought, and a conclusion is given. Chapter 7 takes the entirety of this thesis, give an overall conclusion, and propose future works based on the observations made.

## 2. BACKGROUND AND STATE-OF-THE-ART

This chapter presents, in Section 2.1, background information required to understand concepts related to control theory, heterogeneous computing, and embedded systems of special importance to this work. Section 2.1.1 reviews basic concepts of control theory and how these apply to embedded systems, approaching some seminal works. Section 2.1.2 covers several concepts on heterogeneous computing and a few relevant papers. Section 2.2 reviews the state of the art in the intersection of control theory and embedded systems. The Chapter ends in Section 2.3, where a brief analysis of the revised literature takes place.

### 2.1 Background

This section brings preliminary research on control theory applied in embedded systems and topics on heterogeneous computing. It serves as a basis for the methodology used to gather works for state-of-the-art research on Section 2.2.

#### 2.1.1 Control Theory and Embedded Systems Early Work

This section reviews the basic concepts of control theory required to understand this thesis. Also, it gives historical background, reviewing some of the first works made combining control and embedded systems.

DiStefano et al. [DISW14] define a *control system* as an arrangement of physical components connected or related in such a manner as to command, direct, or regulate itself or another system. We can categorize control systems into two distinct groups in control theory: open-loop systems and closed-loop systems. An *open-loop (control) system* consists of an algorithm that performs changes to the target process solely based in the input. An open-loop control system distinguishing characteristic is that it cannot compensate for any disturbances that add to the controller driving signal [Nis20]. A representation of an open-loop control system appears in Figure 2.2.

A closed-loop control system, in simple terms, is a mechanism that seeks to minimize the difference between the actual value of a system (i.e. the process variable) and the desired value of the system (i.e. the setpoint). To better explain, the controller sends an actuator signal to the system it controls, and sensors measure the effect generated by these inputs.

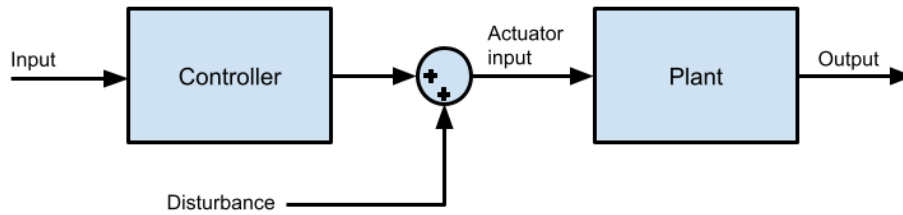


Figure 2.1: Block diagram representation of an open-loop control system.

The sensors outputs are compared to the system desired state, and that information is fed into the controller. Afterward, the cycle repeats itself. Figure 2.2 shows a block diagram of a simple closed-loop control system.

This work focus exclusively on closed-loop control systems. A simple example can illustrate the idea of such a system: consider a water tank level control. The plant is the water tank itself, the sensor is a float placed inside the tank, and the actuator is the output valve.

The controller objective is to keep the water level around a fixed value, independent of the system water flow variations. To achieve this objective, the control engineer must calculate a control law that modifies the system to satisfy the proposed constraints and the desired output.

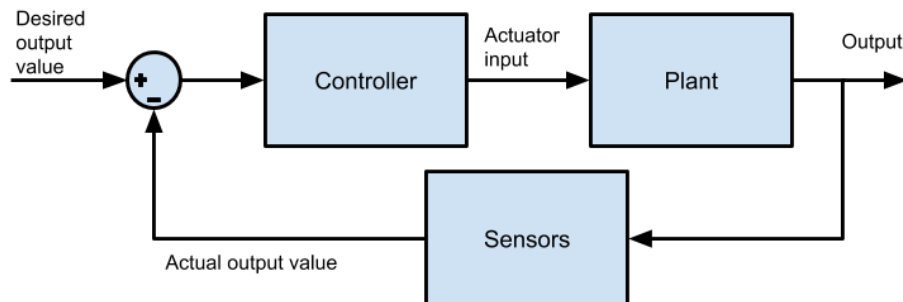


Figure 2.2: Block diagram representation of a closed-loop control system.

There are many ways and techniques to create and implement a control law, depending on the nature of the system dynamics, on the desired performance defined by the engineer, and on the available resources to apply in the controller. In the case of this work, it is assumed digital processing power is at hand.

It is possible to trace the beginning of control theory to the Hellenic period [Ben96], but here attention is placed the digital era of control theory. The idea of using digital computers for process control emerged in the mid-1950s. Serious work started in March 1956 when the aerospace company Thomson Ramo Woodridge (TRW) contacted Texaco to set

up a feasibility study. After preliminary discussions, it was decided to investigate a polymerization unit at the Port Arthur refinery in Texas. A computer-controlled system for the polymerization unit was designed based on the RW-300 computer, and the control system went on-line on March 12, 1959 [ÅW13].

The implementation of control algorithms in microcontrollers and embedded systems has been reported ever since these were introduced on the market [Ben20]. Figure 2.3 represents how usually a control system is implemented in the context of a computational processing unit. The system structure is programmed as a serial algorithm inside a single processor (the processing unit). At each end of the processing unit, there is the integration of digital and analog dynamic systems: analog-to digital (AD) and digital-to-analog (DA) converters. An AD converter takes physical measurements of sensors and translates these to digital form for sensing processing. Later, a DA converter takes the digital controller output and reintroduces the actuator signal to the analog system, that way closing the loop. Today, it is not uncommon for sensors and actuators to get or to furnish information on digital form.

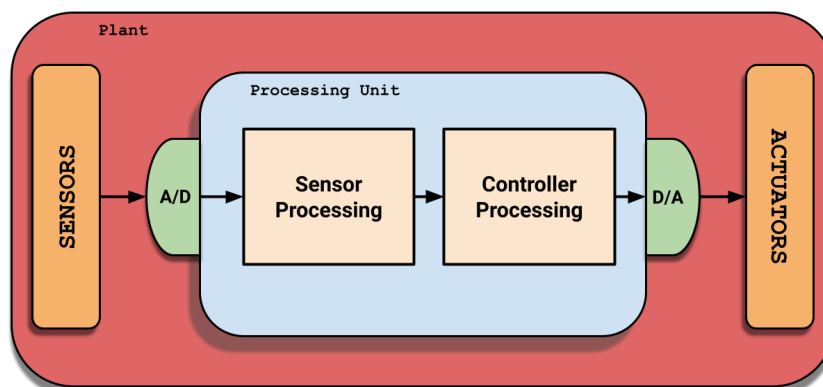


Figure 2.3: Representation of a traditional implementation of a digital control system.

It is possible to highlight a few relevant early works on embedded control. These works were selected based on their focus regarding control theory and how to apply it to embedded systems. All of these were published in the late 1970s and early 1980s.

A first paper worth mention is that by Seim [Sei75], in which a production brazing machine was adapted for embedded control in an Intel 8008 microprocessor. Even a slower processor for the time, the 8008 microprocessor was fitted for controlling a stepping motor that regulates the power applied to an induction heating coil in the brazing process. The controller chosen for this task was a Proportional-Integral-Derivative Control (PID) that ultimately proved more than adequate for the control task.

Johnson [Joh77] proposes, analyses, and simulates experiments with a discrete control algorithm for optimal control of a solar energy system for heating buildings. The dynamic system is based on two controllers utilizing a solar space heater with a conventional furnace as an auxiliary heat source. The optimal control deals with finding a control law based on minimizing a cost function modeled for the system. A MOS Technology 6502

eight-bit microprocessor computed this minimization. Moreover, as later was shown, this project was an improvement over previous work.

Reed et al. [RM77], proposed a general-purpose industrial process controller designed to use the Motorola M6800 microprocessor. A discrete form of the PID control algorithm was implemented in software and used to control a second-order system having time constants on the order of half a second.

Approaching the topic of an embedded nonlinear control system, there is the work of Klein et al. [KM79]. Authors demonstrate the implementation of the sliding mode control method with a microprocessor through the example application of controlling one leg of a six-legged walking vehicle called Hexapod. They used a computer system to control the leg three joints, and the heart of the system is an IMSAI microcomputer, which is based on the Intel 8080 microprocessor.

Furthermore, in 1982 Ohlson et al. [OWJ82] published a work based on a micro processor-based feedback controller of a medical, mechanical ventilator. Interestingly, the controlled continuous process is not a machine but a living creature. The system comprises a servo ventilator as the actuator, a CO<sub>2</sub> analyzer combined with a lung mechanics calculator serves as the measurement unit, and an IMSAI PCS 80/30 microcomputer is used to implement the control algorithm and to collect and tabulate data. The processor board is based on an Intel-8085 microprocessor.

As shown, for most Embedded Control Systems (ECSs) lifetime, controllers were based on microprocessors. Although providing most of the basic features to implement basic control systems (processor, input/output, converters), these microcontrollers usually provide low levels of computing power and only support simple applications [ACVR05]. Fast-forwarding a few decades into the future, new control applications start to appear, and these demand much higher computational complexity, functionality, and reliability than the traditional computational processing paradigm can deliver. Therefore, the search for new computational architectures leads naturally to the domain of heterogeneous computing.

### 2.1.2 Heterogeneous Computing

Moore's law describes the trend over time of the number of transistors in an integrated circuit (IC) and the associated IC cost. It was first announced as an observation by Gordon Moore and was published in 1965 [Ma19]. Moore predicted that the number of transistors integrated in a single chip would double every two years, without a significant increase on the cost of such IC. This trend followed for over thirty years [BDH<sup>+</sup>10]. In the last few years, this exponential transistor increase allows the production of ICs with several billions of transistors. However, as wires and logic gates become smaller, the advance of miniaturization becomes harder to achieve.

Reaching this physical limitation of processors, the need for a different way to continue increasing hardware performance according to increasing software demands keeps climbing [AC18]. So, one possible way to handle such performance requirements is through heterogeneous computing. This thesis employs heterogeneous computing due to:

- to the physical space availability of the IC, granting the ability to implement multiple tasks;
- the extensive repository of intellectual property to solve many problems: network access, general and specialized processing, storage, and multiple interfaces for many systems;
- the wide range of digital, computational, and control systems construction methods.

These characteristics enable the implementation of devices dedicated to composing the various parts of a control system.

A heterogeneous computing system (HCS) distributes data, processing, and program execution among different processors, such that each is best suited for one or for a few specific computational tasks. An HCS can be a super-computing system with heterogeneous cores or a cluster of heterogeneous processors or a system composed by multiple distinct processors associated to specific accelerators (such as floating or fixed point arithmetic units, dedicated neural networks, etc. A processor here can be a general-purpose processor (GPP), a special-purpose processor (e.g., a digital signal processor (DSP) or a graphics processing unit (GPU)), or a co-processor [ZWT<sup>+</sup>18]. The system is called heterogeneous because the (co-)processors are different from some main device; they may have different instruction sets, or the programming languages and environments may be different. The (co-)processors can achieve performance and energy efficiency with specialized processing capabilities to handle particular tasks [Ma19].

An HCS is generally used for the execution of a large number of applications. The advantage of HCSs lies in the different structures of multiple processing units in the system, which are suitable for executing different types of applications [HGL<sup>+</sup>20].

Today, there is a wide variety of system architectures, ranging from homogeneous multicore processors to graphics processing units (GPU) combined with CPUs. The following sections discuss some such system architecture structures.

## Multicore

The first architecture possible to highlight is the multicore. A *de facto* definition of a core, according to Zahran [Zah19] is a CPU and its level-1 caches (for instructions and for data).

Below the L1 caches are different designs. One design has a shared L2 and L3 cache, where L3 is usually the last-level cache (LLC) before going off-chip. An L2 cache is typically physically distributed and logically shared to increase scalability with the number of cores. Figure 2.4 shows a representation of the multicore architecture

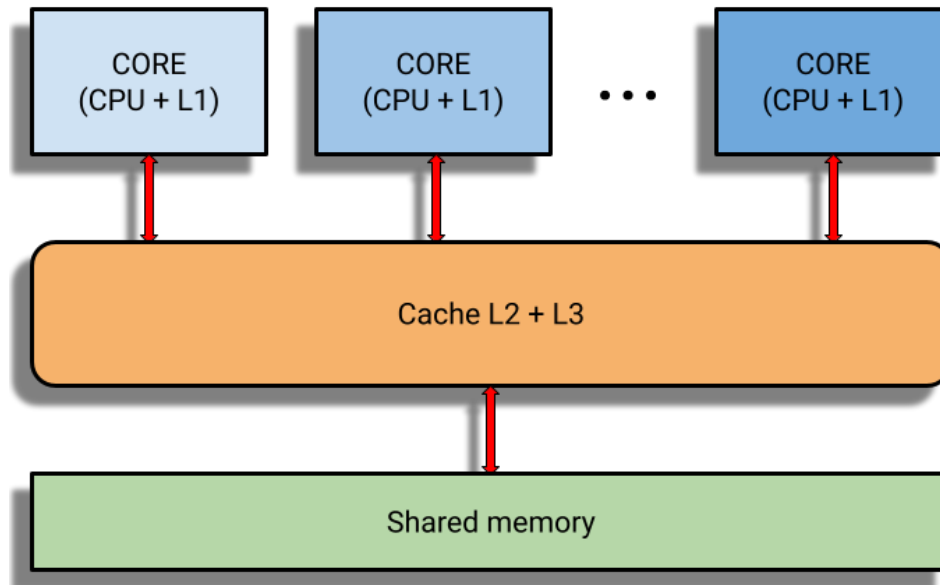


Figure 2.4: Diagram of a multicore architecture. Adapted from [Zah19].

Multicore architectures typically evolved from general-purpose cores, employing a low number of rather heavy-weight, highly complex cores and multilevel cache hierarchies with complete cache coherence across all cores in a node [BHKW12]. All multicore architectures aim to either exploit concurrency, increase computing density, handle partitioned workloads, or achieve some combination of these objectives [LC09].

In the case of multicore architecture applied to control theory, the work of Youness et al. [YMK14] serves as an example. In this paper, authors present a detailed case study of an embedded real-time(RT) self-tuning PID controller for a 1-degree-of-freedom (1DOF) aerodynamical system. To test the multicore architecture parallel processing capabilities, three control algorithms were implemented. The first one, a master task, spawns three slave tasks. Each of these computes one of proportional, integral, and derivative gains. In the second implementation, each core of the multicore architecture executes a separate PID controller, producing a multi-PID digital controller. A master thread manages up to seven worker threads. Each the worker threads implements an independent or cooperative PID control algorithm. The third algorithm comprises a self-tuning PID based on fuzzy logic. Here, a master task manages two slave tasks running in different cores: the first one is the PID controller, and the second is the fuzzy logic tuner.

Another multicore architecture application the Kalray-256 processor. It integrates 256 user cores and 32 system cores on a chip with 28nm CMOS technology. The MPPA-256 chip [DDAB<sup>+</sup>13] integrates 16 compute clusters and 4 I/O subsystems located at the



periphery of the processing array to control all the I/O devices. Each I/O subsystem contains an SMP quad-core with a shared D-cache, on-chip memory, and a DDR controller to access up to 64GB of external DDR3-1600 [DDAB<sup>+</sup>13]. Figure 2.5 shows a representation of the MPPA architecture.

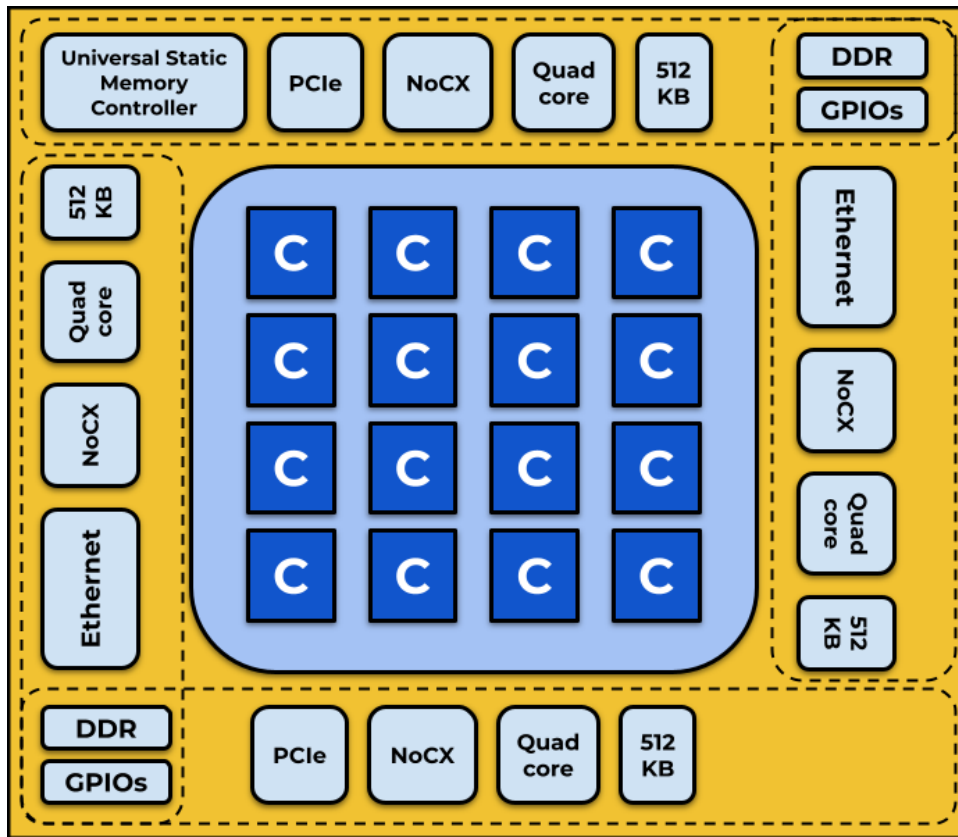


Figure 2.5: MPPA multicore architecture. Adapted from [DDAB<sup>+</sup>13].

### The big.LITTLE Architecture

The big.LITTLE architecture is designed to employ separate cores with different computing capabilities within the same CPU. Figure 2.6 illustrates the block diagram of a representative CPU designed using the big.LITTLE architecture [CECK12]. This architecture was introduced to give a better energy performance trade-off in a single silicon. Two types of cluster cores are embedded: performance-driven big core and energy efficiency-driven LITTLE core [CKC12].

At its core, the big.LITTLE architecture is the first step in the creation of heterogeneous computing. This architecture creates a trade-off between hardware and software complexity and performance. With only one ISA and hardware/software performance levels, tasks like code migration become possible. That more heterogeneous cannot deliver.

Both cores implement precisely the same processor architecture and are capable of executing the same instructions. As said before, the only difference lies in the way cores

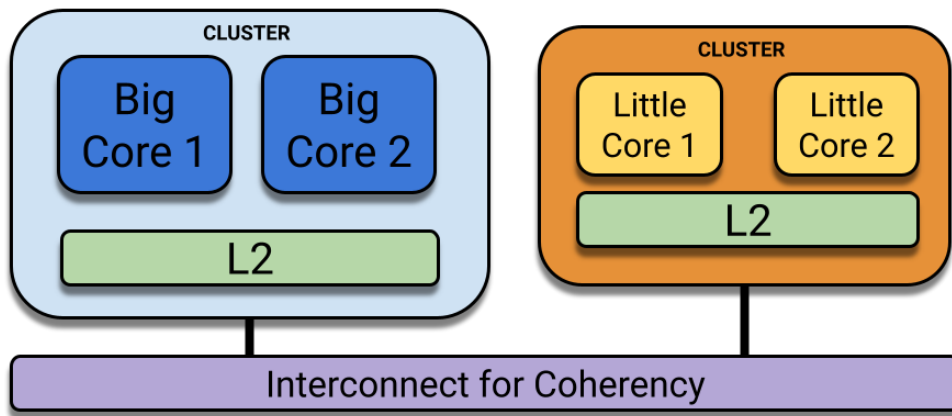


Figure 2.6: Diagram of the big.LITTLE architecture. Adapted from [CECK12].

handle execution. While the “big” core is designed with performance as its primary goal, the “LITTLE” core is designed with energy efficiency as its principal target [CECK12]. The “big” core is implemented to achieve high performance by running more instructions in parallel on a more extensive and more complex pipeline. The “LITTLE” core pipeline is simpler and targets high energy efficiency. Its performance is lower than that of the big core, but can be sufficient for most common usage scenarios executed by modern mobile devices [Gre13].

In the first big.LITTLE system from ARM, a ‘big’ ARM Cortex-A15 processor, is paired with a ‘LITTLE’ Cortex-A7 processor to create a system that can accomplish both high and low computational demand tasks in the most energy-efficient manner.

By connecting the Cortex-A15 and Cortex-A7 processors via the CCI-400 coherent interconnect, the system is flexible enough to support various big.LITTLE use models, which can be tailored to the processing requirements of the tasks [Gre11].

The work of Mishra et al. [MILH18] proposes a resource manager that learns key control parameters to meet latency requirements with minimal energy in complex, dynamic environments. Later, testing this manager’s ability to deliver reliable latency on heterogeneous ARM big.LITTLE architectures in both single and multi-application scenarios.

That goal was achieved by using learning to model complex resource interaction and control theory to manage system dynamics.

### CPU and GPU Combination

Though the primary reason for introducing GPU was for graphical purposes, it is now being used for general-purpose parallel computing [RC18]. This development’s main reason is that more transistors are devoted to data processing rather than data caching, flow control, and error correction. Furthermore, GPUs are comparatively cheap because of their high fabrication volume [BHKW12].

According to Mittal and Vetter, 2015 [MV15], computer architects, programmers, and researchers are moving towards a CPU and GPU paradigm where the best features of both can be intelligently combined to achieve even further computational gains. This paradigm aims to match each application's requirements to CPU/GPU architectures' strengths and achieve load balancing by avoiding idle time for both the Processing Units (PUs).

Fei et al. [LLW16] divides this architecture into fused CPU+GPU and discrete CPU+GPU. In a discrete system, the CPU has dependent memory space to GPU, and there is a large amount of overhead such as communication time and memory bandwidth to communicate between CPU and GPU, which make it not efficient enough. In a fused system, the CPU can share memory space with GPU, and the communication overhead is avoided. Figure 2.7 shows a GPU with 32 highly multi-threaded SIMD accelerator cores combined with a standard multicore CPU in a discrete architecture.

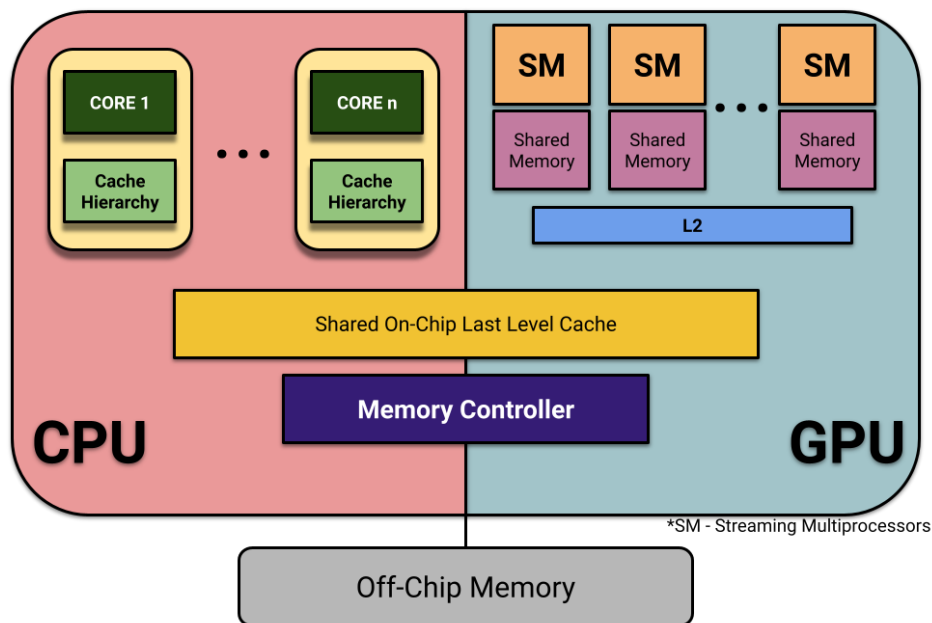


Figure 2.7: Discrete CPU + GPU architecture. Adapted from [Aro12].

In Chitchian et al. 2013, [CSvAK13] the authors present the design, analysis, and real-time implementation of a distributed computation particle filter on a graphic processing unit (GPU) architecture that is especially suited for fast real-time control applications. The controlled plant, in this case, is a three-jointed robotic arm. Each joint has a sensor to measure its angle. There is a camera mounted at the end of the arm. This camera is used for tracking an object which is moving on a monitor on a fixed  $yz$  plane. The control objective is to track a moving object with the robotic arm while it traverses the screen. The particle filter application based on the CPU-GPU paradigm is divided into a host and device side. The host refers to the CPU connected to one or more devices (i.e., the GPUs). The algorithm is

divided into four steps running in each of the computing units of the architecture's accelerator side (GPUs).

A practical example of this CPU and GPU combination is the Jetson TK1 Development Kit, designed around the 192-core NVIDIA Tegra K1 mobile processor. There are two versions of Tegra K1 mobile processors: 32-bit version and 64-bit version, which are pin-compatible [1]. The 32-bit version uses a quad-core Cortex-A15 CPU, which runs at clock rates up to 2.3GHz, is 3-way SuperScalar, and has 32KB L1 Instruction Cache and 32KB L1 Data Cache. The 64-bit version uses a custom dual-core Denver CPU, which runs at clock rates up to 2.5GHz is 7-way SuperScalar and has 128KB L1 Instruction Cache and 64KB L1 Data Cache [LSN14]. Figure 2.8 shows a high-level scheme of the NVIDIA Tegra K1 mobile processor (32-bit version).

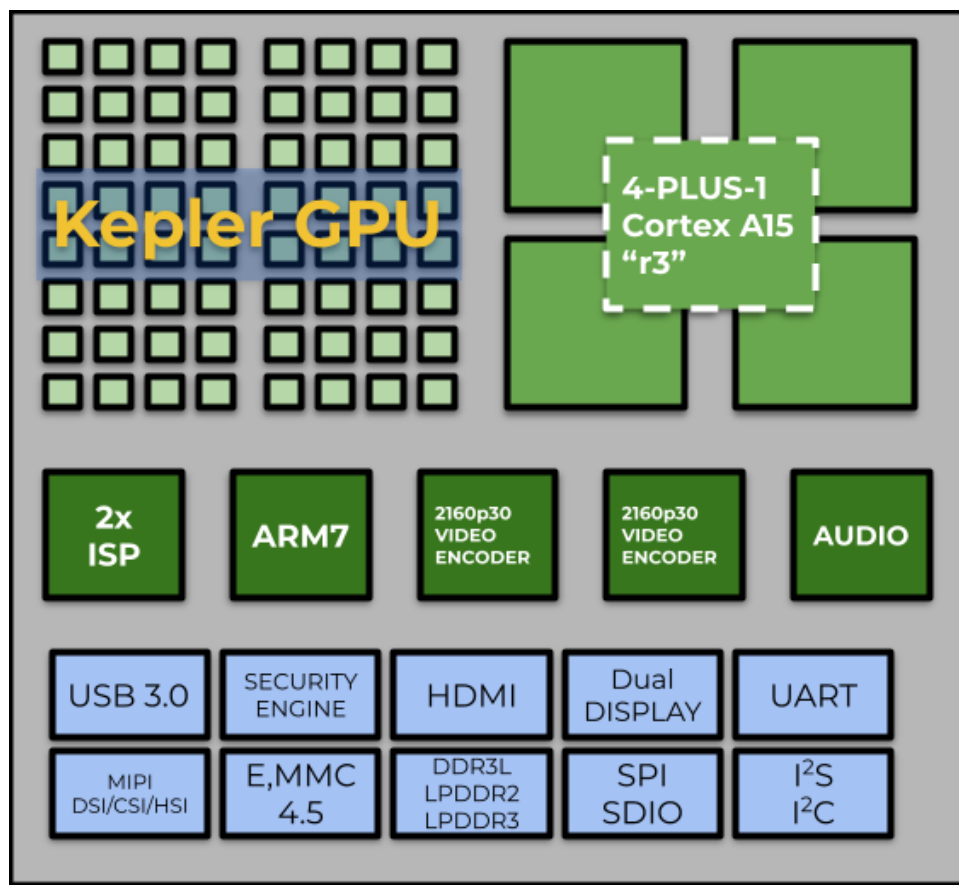


Figure 2.8: NVIDIA Tegra K1 mobile processor (32-bit version). Adapted from [LSN14].

## CPU and FPGA Combination

FPGAs can be viewed as user-defined application-specific integrated circuits (ASICs) that are reconfigurable. They offer fully deterministic performance and are designed for high throughput, for example, in telecommunication applications [BDH<sup>+</sup>10].

FPGAs are unique kinds of chips that are configurable by the end-user, but FPGA resources are of a fixed size and have limited flexibility. FPGAs are programmed using high-

level synthesis (HLS), circuit schematics, or a hardware description language like VHDL or Verilog. The essential components of FPGA are configurable logic blocks (CLBs) or logic array block (LAB) and look-up-tables (LUTs) [TDSP19].

Among various heterogeneous acceleration platforms, the FPGA-based approach is considered to be one of the most promising directions, since FPGAs provide low power and high energy efficiency and can be reprogrammed to accelerate different applications [CCF<sup>+</sup>16]. A typical PCIe-based CPU-FPGA platform features direct memory access (DMA) and private device DRAM. To interface with the device DRAM as well as the host-side CPU-attached memory, a memory controller IP and a PCIe endpoint with a DMA IP are required to be implemented on the FPGA, in addition to user-defined accelerator function units (AFUs) [CCF<sup>+</sup>19].

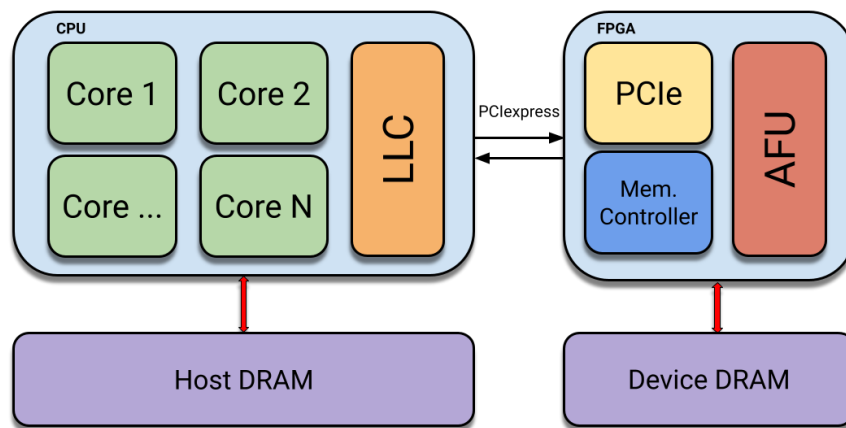


Figure 2.9: CPU + FPGA architecture. Adapted from [CCF<sup>+</sup>19].

The trend towards tighter integration of CPUs and FPGAs enables more collaborative execution between devices. Rather than executing an entire kernel on the FPGA while the CPU is idle, collaborative execution makes better use of the overall system resources by involving both CPU threads and FPGA in the execution [HCEH<sup>+</sup>19].

Rui et al. [RFXP12] present a control system designed to reduce an elevator's energy consumption by transforming the energy from the motor working at the generating state to the power grid using a Fuzzy Logic PID controller. The proposed system adopts dual-PWM control to implement the energy feedback, 32-bit ARM CPU and a 32 bit DSP CPU + FPGA structure, two CPU to exchange data in a high-speed through the parallel communication, lift master board and calling landing workstation, and ceiling assembling workstation to exchange information through the CAN bus serial.

As an example of a CPU-FPGA combination, the Xilinx Zynq-7000 device is highlighted. These products integrate a dual-core Arm Cortex<sup>TM</sup>-A9 MPCore<sup>TM</sup> based processing system (PS) and Xilinx programmable logic (PL) in a single device. The Arm Cortex-A9 MPCore CPUs are the heart of the PS, including on-chip memory, external memory inter-

faces, and a set of I/O peripherals [Xil18]. Figure 2.10 shows a high-level scheme of the ZYNQ Architecture.

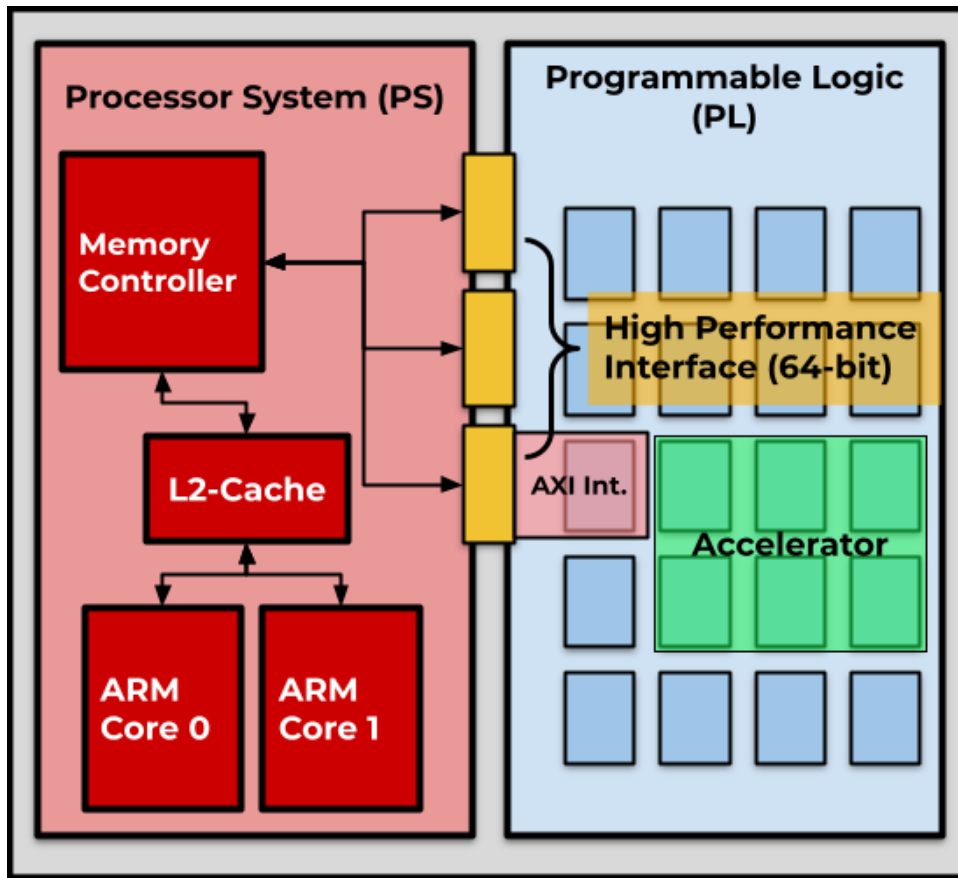


Figure 2.10: A Block Diagram of the ZYNQ Architecture. Adapted from [MWH13].

### Digital Signal Processors (DSP)

As of now, there are many others processing nodes more than what was mentioned before. One of them is the DSP (digital signal processor). These target small niches of applications, however, are not as versatile as the ones mentioned earlier [Zah17]. As with any specialized circuits, DSPs are more energy-efficient than general-purpose processors for analog signal processing applications. That is why we see DSPs in many portable devices. However, they can also be used in high-performance computing for scientific applications when needed [Zah19].

In Li et al. 2012 [LHL<sup>+</sup>12] the authors explore the DSP-GPU heterogeneous computing architecture and propose a communication implementation. A Master-slave model is designed for a heterogeneous computing system. DSP and GPU are defined as master core and slave core separately. The computing task-parallel analysis and assignment are processed in DSP, and the independent parallel computing task is transferred to the GPU unit by a message control unit. Figure 2.11 shows a representation of DSP-GPU heterogeneous computing system structure.

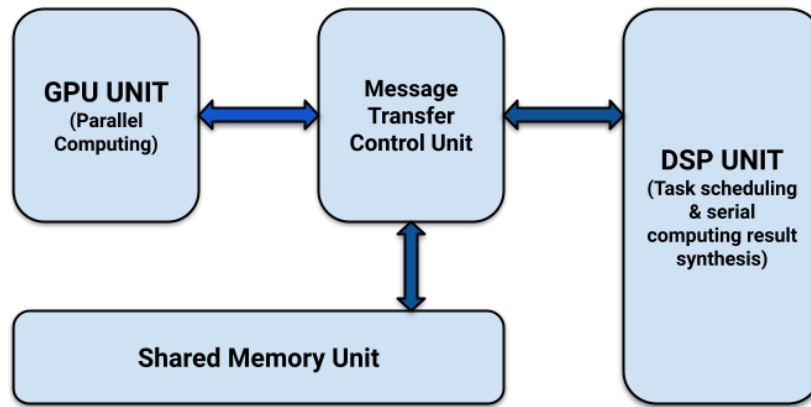


Figure 2.11: DSP-GPU heterogeneous computing system structure. Adapted from [LHL<sup>+</sup>12].

The work of Sousa and Meloni, 2012 [SM12] proposes a reconfigurable and heterogeneous computing architecture for digital signal processing on embedded systems, based on the cooperative code execution between DSP and FPGAs. Validating this approach by processing FFT (Fast Fourier Transform) and DCT (Discrete Cosine Transform) algorithms.

This work highlights the Qualcomm Snapdragon 800 as a commercial application of the DSP-CPU combination architecture. The Qualcomm Snapdragon 800 is an ARM-based SoC for tablets and smartphones. The CPU portion is based on Qualcomm's Krait architecture, which is compatible with ARMv7 ISA [Che21]. This system-on-chip has two instances of the Hexagon digital signal processor (DSP). The modem (mDSP) is dedicated and customized for modem processing, whereas the application DSP (aDSP) is used for multimedia acceleration [CAV<sup>+</sup>14]. Figure 2.12 shows a block diagram of the Snapdragon 800 Architecture.

## 2.2 State-of-the-art

This section focuses on the related works in the intersection of both fields of study (control theory and heterogeneous embedded systems), covering a vast research topic. Figure 2.13 shows which topics were selected for guiding the scope definition of this state-of-the-art review.

With the topics selected, the next step is to create our research questions.

1. Are the fields of heterogeneous embedded systems and control theory integrated?
2. Which topic of control theory, the system is applied (e.g. embedded controller, embedded sensor processing)?
3. Which heterogeneous computing architecture are being used ?

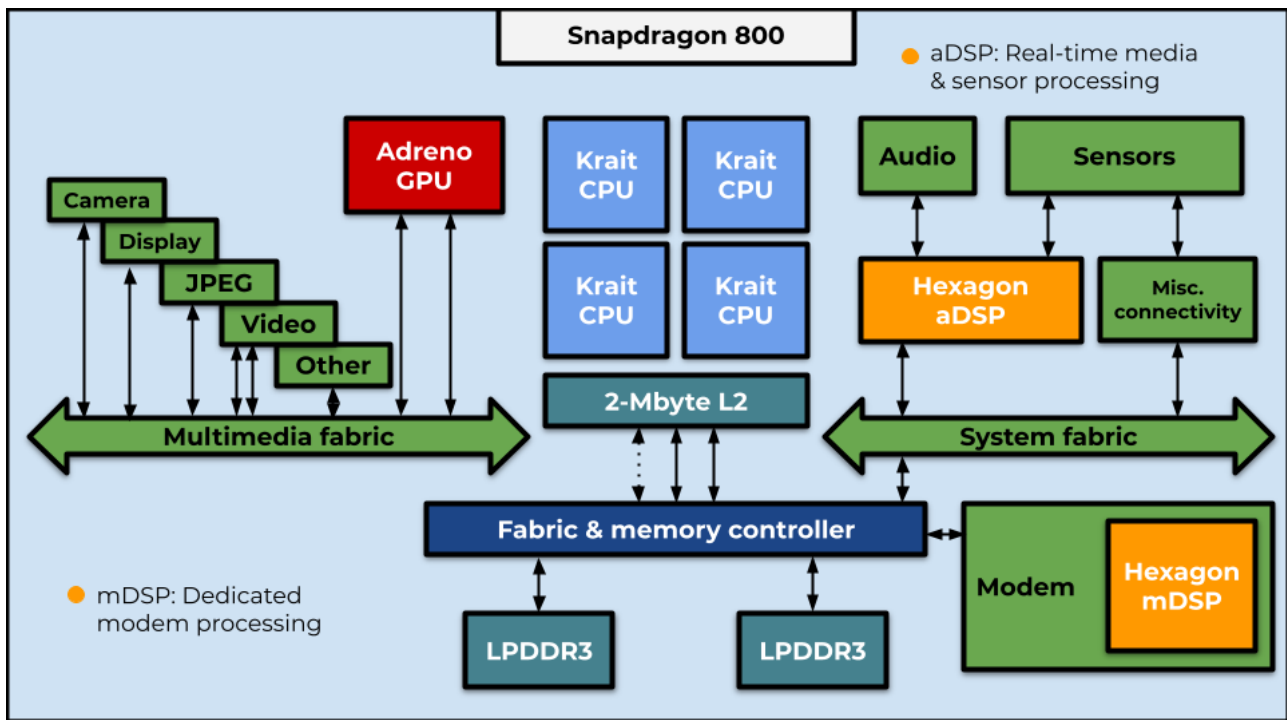


Figure 2.12: Snapdragon 800 block diagram. Adapted from [CAV<sup>+</sup>14].

4. Does the controller developed acts intra-chip? Only in an external application? Or both? See the example on Figure 1.1
5. Does the study deals with energy management? If yes, How ?
6. Does the study deals with fault tolerance? If yes, How ?
7. Does any of these studies propose any form of unifying these topics (e.g., energy management, fault tolerance) under a single processing framework? How?

The remainder of this section presents a summary of the selected works.

### 2.2.1 Bulat Khusainov et al.

The work of Khusainov et al. [KKSC18] proposes an implementation of an interior-point-based nonlinear predictive controller on a heterogeneous processor where the workload can be split between a general-purpose CPU and an FPGA. A continuous-time optimal control problem demands that the nonlinear time-invariant system be described as an ordinary differential equation (ODE). And then, to solve this control problem, two main stages must be resolved: integration and optimization.

The authors find that splitting the workload between a CPU and FPGA was the best course of action due to: the fact that integration, i.e., solving the ODE, is not desirable



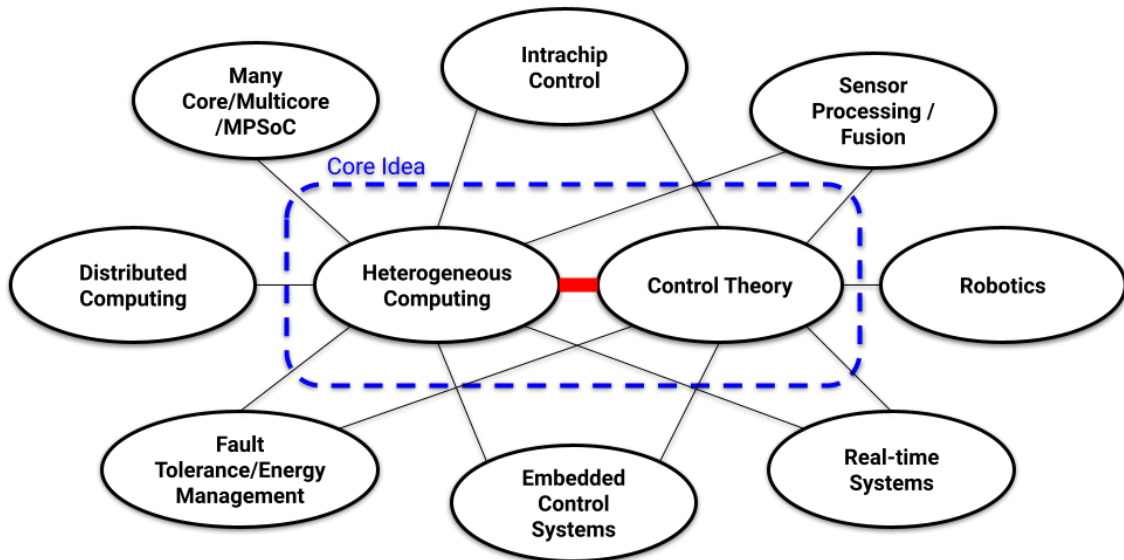


Figure 2.13: Topics selected to start the state-of-the-art research

in an FPGA because ODEs may involve mathematical expressions, like sines and square roots, that require plenty of computational resources, more resources (LUTs, DSP, internal memory) than those available in the FPGA and could be unsuitable for pipelining. On the other hand, optimization algorithms can benefit from hardware acceleration because they have an iterative nature, which is beneficial for reusing computational logic and the fact that underlying linear algebra algorithms can be efficiently mapped onto hardware.

The interior-point-based nonlinear predictive controller is based on a Primal-dual interior-point method algorithm, which uses a Minimum Residual solver. The entire algorithm is implemented in software, in an ARM Cortex-A9 processor. The matrix-associated problems, like matrix-vector multiplications and the linear system solver, are accelerated in hardware.

A proprietary software tool called Protoip was used to allow the quick prototyping and processor-in-the-loop verification of optimization algorithms on a Xilinx Zynq system-on-a-chip (SoC), which contains an ARM processor and FPGA fabric. The experimental setup represents a closed-loop system that consists of a gantry crane model and a heterogeneous computer running the predictive controller. Processor-in-the-loop testing implies simulating the controlled plant in a Matlab environment on a PC while performing computations on the embedded platform, as shown in figure 2.14.

What makes this paper relevant for the thesis is the fact that this implementation shows that the performance of a heterogeneous computer-based controller can be efficiently traded off against resource usage by shifting the computational workload between the CPU and the FPGA, while varying the amount of parallelism for a given part of an algorithm might be less efficient or even without any benefits.

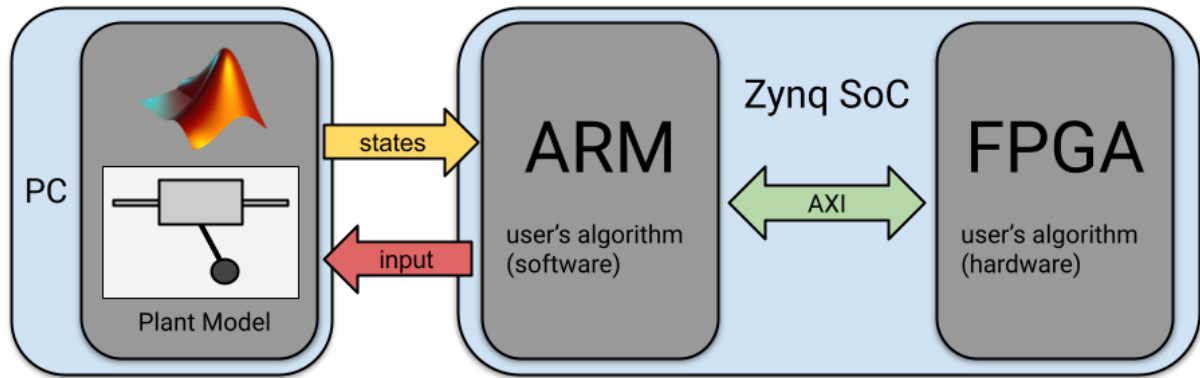


Figure 2.14: Processor-in-the-loop test. Adapted from [KKSC18].

### 2.2.2 Kasra Moazzemi et al.

Moazemmi et al. [MMY<sup>+</sup>19] explores the need for runtime resources managers to efficiently, dynamically, and robustly manage shared resources in the heterogeneous computing field. The authors addressed this through HESSLE-FREE (Heterogeneous Systems Leveraging Fuzzy Control for Runtime Resource Management): an approach leveraging fuzzy control theory that combines classical control theory's strengths with heuristics to form a runtime resource manager for heterogeneous systems.

HESSLE-FREE is a lightweight monitoring system that captures power and performance metrics from each computing unit in the system and makes runtime resource allocation and tuning decisions using fuzzy control theory with low overhead. It controls various knobs in this system, such as the operating frequency of CPU clusters and GPU separately, and decides on the active core in the heterogeneous multiprocessor. Figure 2.15 shows an abstract view of HESSLE-FREE architecture.

The platform chosen for this architecture application was an NVIDIA Jetson TX2 development board, which contains a heterogeneous multiprocessor (HMP) and an NVIDIA GPU. The HMP contains a quad-core ARM Cortex A57 cluster and a dual-core NVIDIA Denver cluster. NVIDIA Pascal CUDA cores power the GPU. The controllers used in the experiments were implemented as Linux userspace daemons that execute in the background with the applications. CPU and GPU runtime energy are separately measured on-board alongside current and voltage using sensors present on the JetsonTX2 development board. The experiment devised to validate the system was based on the CPU and GPU simultaneously execute their workloads (a face detection algorithm), while HESSLE-FREE optimizes the user metrics, like frames per second delivered by the GPU or QoS metric delivered by the CPU, as well as the energy consumption of the entire system.

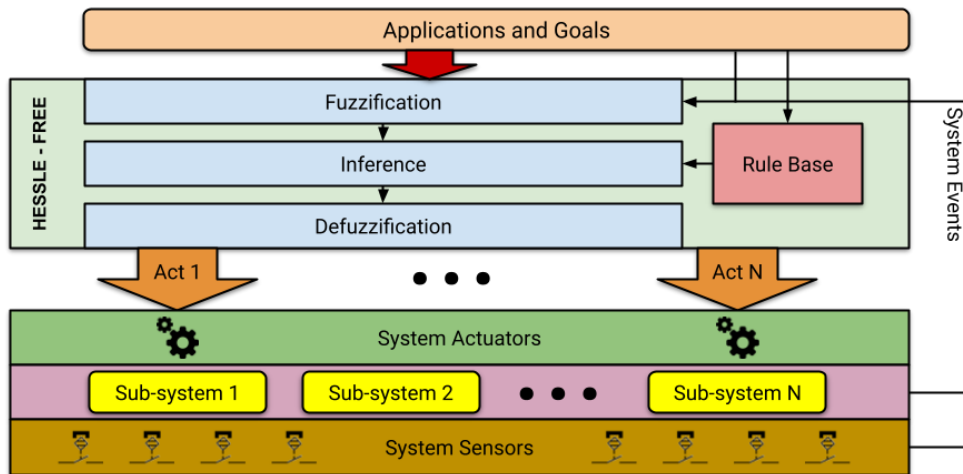


Figure 2.15: HESSLE-FREE architecture. Adapted from [MMY<sup>+</sup>19].

The article demonstrated the simplicity and effectiveness of the proposed architecture. The paper evaluations show that HESSLE-FREE successfully managed complex systems in an energy-efficient manner while achieving QoS targets. The authors claimed that their system shows the usefulness of fuzzy control for resource management in heterogeneous computer systems and their increasing complexity.

### 2.2.3 Michael Giardino et al.

Giardino et al. [GKFF20] address the topic of intra-chip resources management impacting the quality-of-service delivered by the application controller.

In the paper, the authors propose developing a software architectural framework for implementing compute-aware control systems. In this case, the term "compute-aware" describes controllers that can modify existing low-level computing platform power managers in response to the physical system controller's needs.

The framework allows bidirectional guidance between the computing system's hardware layer and the physical system controller's application layer. This means that the application layer is aware of its hardware-defined power constraints and can adjust its algorithms accordingly, and can adjust computing resources as needed. In this case, the framework tries to balance quality-of-service and energy management.

This software framework is built upon three main components: compute-aware physical system controllers, computer power and performance controller, and a quality-of-service manager (see Figure 2.16). Compute-Aware Physical System Controllers (CAPSCs) are standard software controllers reformulated into algorithms that can be stopped or reconfigured at any time due to sudden limitations in the computer resources. The QoS Manager

(QoSM) receives monitored metrics and converts this into a QoS that can be passed to the computer power and performance controller (CPPC). The CPPC takes the desired QoS, converts it into available physical power/performance modes, and relays them to the hardware/OS.

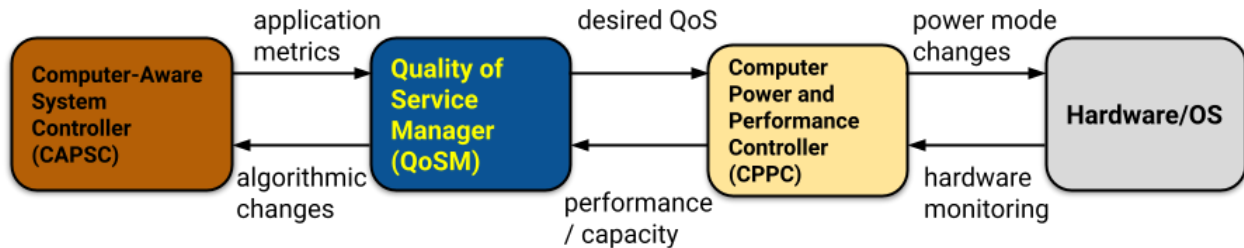


Figure 2.16: A high level view of the software framework architecture. Adapted from [GKFF20].

The performance target in this work is not a computational metric but rather the physical system's performance. In the experimental example, the performance is how closely a mobile robot can track a desired path. As mentioned before, the framework's test platform is a custom mobile robot. The computational unit is a hybrid, stacked, heterogeneous architecture consisting of two primary computational units. The lower level CPU is an ATmega328/P responsible for running the PID motor controllers and odometry; this is connected via SPI to the upper-level CPU. This is an ODROID XU4, based upon the Samsung Exynos5422 Cortex-A15/A7 Octacore SoC, responsible for the path planning, trajectory planning, and running the framework. This ARM big.LITTLE heterogeneous multi-processing architecture allows for transparent thread scheduling between high-performance higher-power cores and lower-performance lower-power cores and per-cluster DVFS.

A situation-aware (SA) governor was proposed to change the CPU clock frequency in response to situations perceived by the physical control system application. In this case, if the control system perceives that it is in a situation that requires high performance or that it is in a computationally intensive region, it sets the CPU performance state to the highest power/performance mode. In other circumstances, the SA governor attempts to reduce the power by feeding back the performance error. In other words, the algorithm puts downward pressure on the frequency of the processor, reducing the energy consumption until the reduction in performance causes an increase in the measured error, which then puts upwards pressure on the frequency, keeping the performance high enough to meet error targets.

Figure 2.17 shows the experimental results showing the obstacles and paths taken by the robot when the computing platform is running under one of four cases: The static-low-power and static-high-power settings, the situation-aware (SA) governor, and a standard Linux on-demand governor.

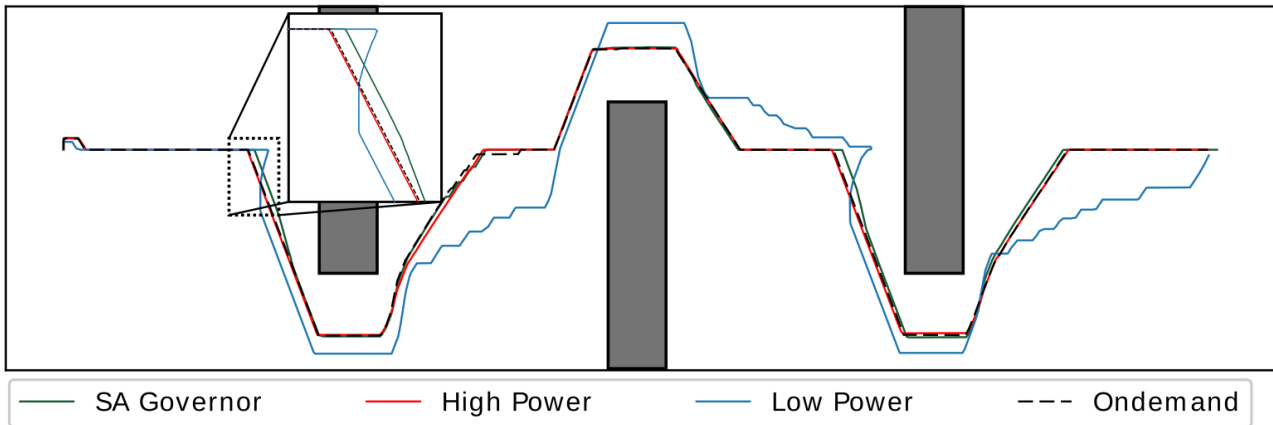


Figure 2.17: Experimental results showing the obstacles and paths taken by the robot in each case [GKFF20].

Based on the analysis of the performance and the energy profiles generated through the experiments, the authors concluded that the governor could respond to the physical system controller's needs, reducing power significantly. In a complex scenario, power was reduced by 26.9% over static high power and 20.6% of the existing power-management governor. At the same time, it maintains a performance similar to the high-power case.

#### 2.2.4 Markus Ulbricht et al.

Sensor processing is a critical part of control theory; in Ulbricht et al. [USK19], the authors propose an approach for designing a sensory system based on multicore systems that target automated driving. The system can be controlled to support low power, fail-safe, fail-operational, and distributed execution of different tasks, all while keeping the strict timing and safety constraints crucial in the automotive area, therefore also approaching the topic of fault tolerance in heterogeneous computing.

In this work, a triple-core system was chosen for the basis of the implementation. This system facilitates the ability to support many different ways of executing tasks, like in a low power mode by turning off single cores, a performance mode where tasks are executed in parallel on multiple cores, or a fault-tolerant mode, where a copy of a task runs on each core redundantly.

To determine the order in which task runs on each core, the authors determine three execution modes that determine the level of fault tolerance that the user desire (see Figure 2.18b). In the SE (single execution) mode, only a master role in an active core is running critical and non-critical tasks, and this operation mode does not allow error detection. In the FS (fail-safe) state, two active cores are running the master and active slave roles. In this mode, it is possible to detect a state mismatch by comparing the other core's system state to their own. Lastly, in the FO (fail-operational) mode, the master core runs a comparison task

that compares the results of the other core's critical tasks and performs a majority vote on the state and resetting the faulty core. Figure 2.18a shows a representation of the system's roles and phases.

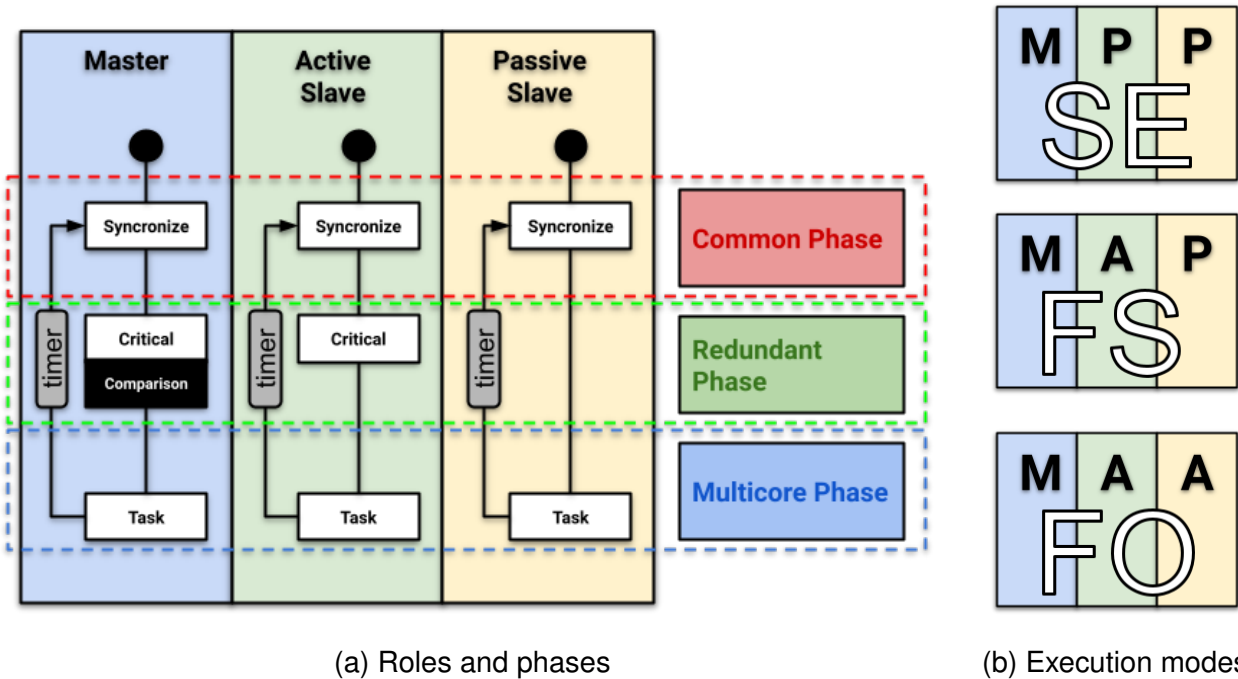


Figure 2.18: The execution modes, roles and phases of the platform. Adapted from [USK19].

The authors use Tensilica Fusion G3 cores to model the multicore system organized in a mesh network. This system is used to run a configurable fault-tolerant multicore system for optimized radar signal and data processing, aiming to realize the functions of a highly automated driving platform. This system's simulation was based on a SystemC model of the three cores simulating SE, FS, and FO state design. Several faults, like a permanent failure of a core, limited execution delay, and temporary corruption of the execution state, were inserted. Finally, the authors concluded that the system supports different grades of fault tolerance for faults in the data and control path; it also optimizes the system to be low power or high performance.

### 2.2.5 Liu Liu et al.

In Liu et al. [LTL<sup>+</sup>21] the authors focus on robotics applications enabled by heterogeneous computing. They propose  $\pi$ -RT, a robotic runtime framework to manage efficiently dynamic task executions on mobile systems with multiple accelerators and the cloud to achieve better performance and energy savings. This work is relevant for the literature review because it deals with heterogeneous computing and sensor processing, and the author considers sensor processing an essential part of control theory.

The efficient utilization of client-side heterogeneous computing resources and cloud resources,  $\pi$ -RT enables complete robotic workloads execution on mobile systems with stringent resource and energy constraints. Figure 2.19a shows the architecture of the framework.  $\pi$ -RT provides a transparent programming interface for users to submit tasks without knowing the details in heterogeneous hardware. According to a scheduling policy, the submitted tasks are appended to the queues of different processing units. Also,  $\pi$ -RT implements a callback function for each processing unit, such that, when the processing unit is done with the current task, it triggers its associated callback function; then it dispatches the first task in its associated queue to the processing unit.

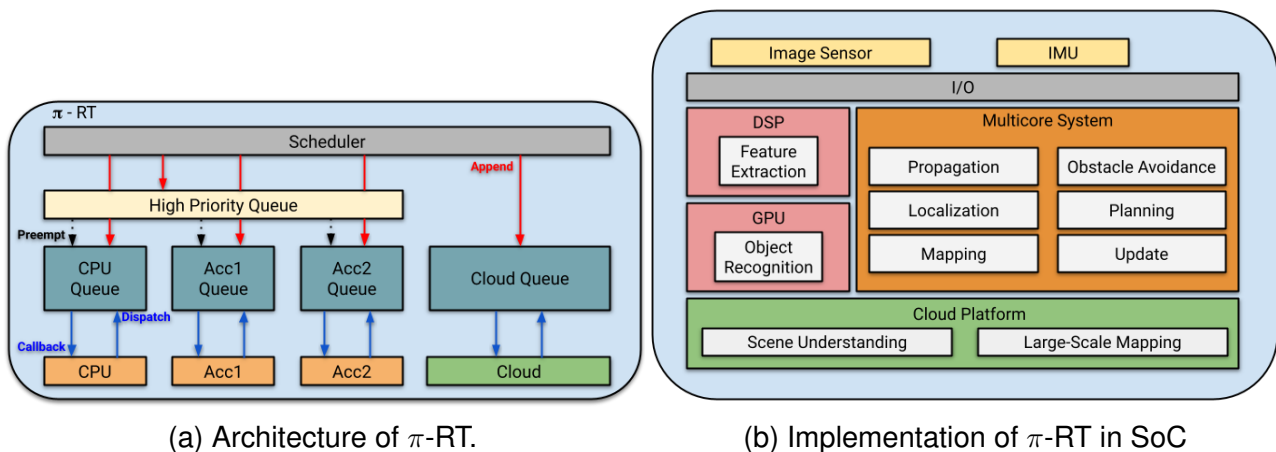


Figure 2.19: Architecture of  $\pi$ -RT and implementation of  $\pi$ -RT in the Snapdragon 820. Adapted from [LTL<sup>+</sup>21].

The central scheduler is designed to check whether this task has real-time requirements; if not, it is dispatched to the cloud. Then, it checks whether the task consumes images as input; if so,  $\pi$ -RT dispatches the task to the high priority queue. Otherwise,  $\pi$ -RT falls back to one of three sub-schedulers: The *Latency-Optimal* scheduler is responsible for offloading the tasks to the CPU based on a Round-Robin algorithm. The *Throughput-Optimal* scheduler keeps the GPU queue full of loads to achieve high throughput. And the *Energy-Optimal* uses the DSP module.

To validate the framework, the  $\pi$ -RT is implemented on the Snapdragon 820 SoC, which has a Quad-core Kryo ARMv8 processor, an Adreno 530 GPU and a Hexagon 680 DSP processor. With  $\pi$ -RT, an autonomous robot is enabled to perform simultaneously autonomous navigation, obstacle detection, route planning, large map generation, and scene understanding, all within an 11 W of the computing power envelope. Figure 2.19b shows a representation of all the robotic applications in the  $\pi$ -RT framework. In conclusion, the authors stated that  $\pi$ -RT is the first robotic runtime framework that efficiently utilizes the on-chip heterogeneous computing resources and the cloud to achieve high performance and energy efficiency.

## 2.2.6 Haoyang Deng and Toshiyuki Ohtsuka

Deng and Ohtsuka [DO18] focus on support control software for heterogeneous computing. Here the authors present a MATLAB software toolkit called ParNMPC, which can automatically generate parallel C/C++ code and carry out closed-loop simulation for nonlinear model predictive control (NMPC). NMPC, as the name suggests, is an optimal control problem for nonlinear systems, and its main drawback is a heavy demand for processing due to the need to solve a finite horizon optimal control problem at each sampling instant.

According to the authors, there are many available software toolkits for NMPC code generators, but none of them make full use of developing parallel hardware such as multi-core processors, GPUs and FPGAs. In the proposed system, the control problem is solved based on a highly parallelizable Newton-type method, which computes the optimal solution iteratively with at most  $N$  cores, where  $N$  is the number of discretization grids or the prediction horizon in the discrete-time case. Then the ParNMPC can automatically generate parallel C and C++ code.

The basis of this software is a solver function called *NMPC\_Iter* performs one iteration to solve Karush-Kuhn-Tucker (KKT) conditions. ParNMPC aims to provide an easy-to-use environment for NMPC problem formulation, initialization, parallel C/C++ code generation, and deployment. Figure 2.20 shows a high-level description of the workflow of ParNMPC.

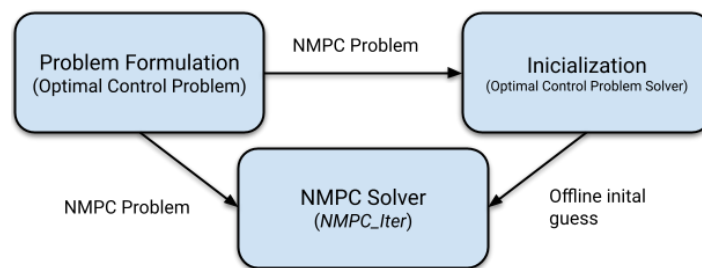


Figure 2.20: Structure and workflow of ParNMPC. Adapted from [DO18].

First, there is the problem formulation issue where the controlled plant and parameters such as the discretization method and the Hessian approximation method are defined. Then, several solvers are provided to solve the first optimal control problem offline to provide an accurate initial guess. Lastly, a parallel C/C++ code of the solver function is automatically generated and it can be easily deployed to different platforms.

To validate the system, the performance of ParNMPC was demonstrated by swinging up a double inverted pendulum and controlling a quadrotor for reference tracking. The simulation was carried out on a desktop computer with an Intel Core i7-8700K processor. The numerical results showed that ParNMPC could achieve a good control performance and



a high speed-up in computation, with a speed-up of 3.7x in the pendulum case and 4.6x in the quadrotor.

### 2.2.7 Jie Tang et al.

Tang et al. [TLG17] focus on sensor fusion in Simultaneous Localization And Mapping (SLAM) applications with heterogeneous computing. The authors propose  $\pi$  – SoC, a heterogeneous system-on-chip design that systematically optimizes the IO interface, the memory hierarchy, as well as the hardware accelerator.

The first step taken was to develop a visual-inertial SLAM system called PIRVS. This system is based on three main components: a highly optimized image processing front-end extracting image features and matching them with the 3D map. An Extended Kalman Filter based tightly-coupled visual-inertial tracking with IMU propagation and state update with 3D-2D feature correspondences. Lastly, mapping with prior poses from the tracking thread.

This preliminary test was developed to determine whether software optimization can enable PIRVS to deliver reliable performance with stringent energy constraints. The SLAM system was deployed on an ARM v8 mobile SoC. The SoC consists of a four-core CPU, running at 2 GHz; the four cores share an L2 cache. Besides the CPU, the SoC consists of a DSP, a GPU. The CPU communicates with the DSP and the GPU through shared memory.

Figure 2.21 shows a representation of the implementation of PIRVS in the ARMv8 SoC and the results from the performance tests. Due to its nature, the DSP processor was selected for the feature extraction task. However, a CPU core was still required to relay the data from the I/O to the DSP. This relay put some pressure on the processing and raising the energy consumption. As seen in the results presented in Figure 2.21, an acceptable 30 FPS was achieved, but a 9 W energy consumption was still high due to SoC architectural design constraints.

The authors state that these results indicate that the existing heterogeneous SoC architecture design is not optimized for SLAM applications. Then  $\pi$  – SoC was proposed; this is a heterogeneous SoC architecture to optimize the I/O subsystem systematically, the memory subsystem systematically, and accelerator integration to improve performance and energy consumption for visual-inertial SLAM. In the previous implementation, the DSP is not directly connected to the I/O system, and a CPU core has to act as a relay to copy image data in memory for the accelerator to consume; this wastes CPU resources, leading to extra energy consumption.

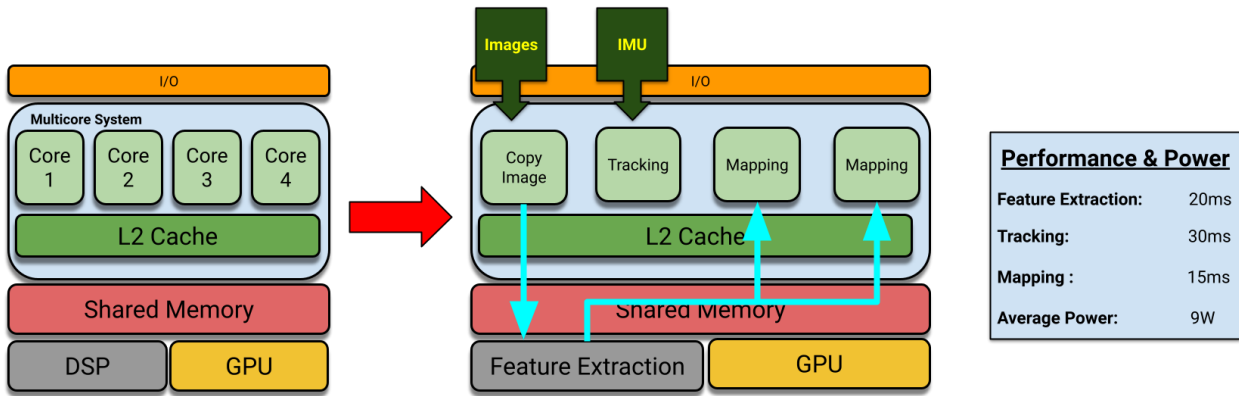


Figure 2.21: Heterogeneous implementation on an ARM v8 SoC and results. Adapted from [TLG17].

To resolve the problems, the authors propose three features for the system improvement:

1. A direct I/O such that the accelerator gets directly connected to the image sensor I/O pins, allowing the accelerator to directly consume image data without the involvement of a CPU core.
2. Design a low-latency ScratchPad memory to directly feed the extracted features to the CPU cores, significantly improving overall SLAM performance.
3. A mechanism to notify the consumers, including the update thread and the mapping thread, the SLAM trigger. This mechanism to notify the consumers works with the Feature Buffer controller that has a register to store the current filled feature memory bank ID. Once a bank is filled, the bank number is written in this register, and the Feature Buffer controller would then interrupt the CPU to notify the update thread and the mapping thread about the incoming new features.

Figure 2.22 shows the representation of the  $\pi$  – SoC architecture and a high-level description of the workflow. The workflow shows that the accelerator is connected to the image sensor through Direct I/O such that each time a new image comes in, it triggers the feature extraction task on the accelerator, and the extracted feature gets written to the ScratchPad memory. Once the extracted features fill a ScratchPad bank, the ScratchPad controller interrupts the CPU to trigger SLAM. After implemented  $\pi$  – SoC in an FPGA, the authors repeated the same tests used in the ARMv8 implementation. With that, they could find that not only the  $\pi$  – SoC outperformed in every single category but the energy consumption the FPGA implementation consumed 8 times less energy than the ARM baseline's energy efficiency.

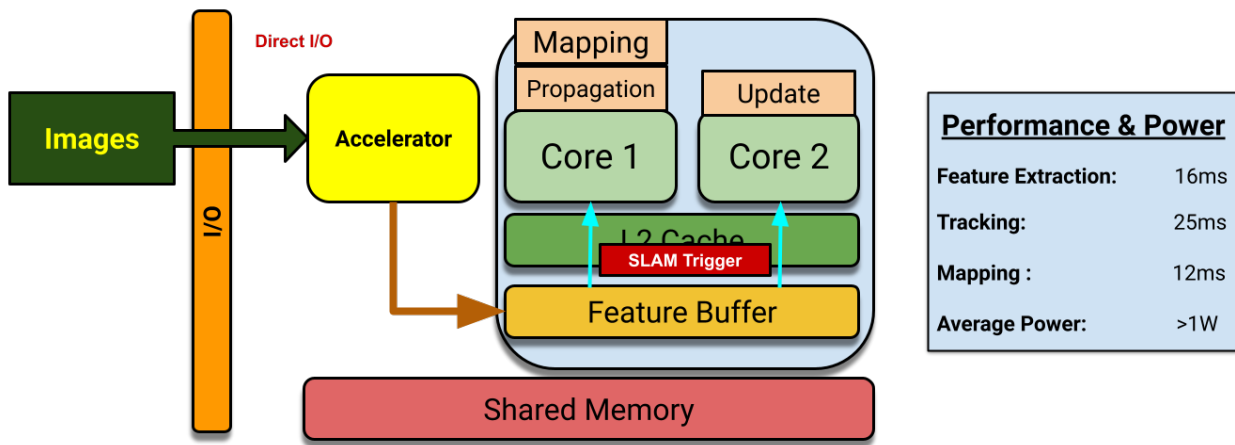


Figure 2.22:  $\pi$  – Soc heterogeneous architecture and results. Adapted from [TLG17].

### 2.2.8 Yu Ma et al.

Ma et al. [MCDH17] proposes an online framework called: reliability-aware utilization control (RUC). And is used to improve the lifetime reliability in the presence of wear caused by failure mechanisms that strongly depend on temperature and temperature variation. This framework is divided into two central systems: the first is a model-predictive controller (MPC) that keeps system utilization at the desired value. The utilization setpoint algorithm incorporates various design considerations, including real-time constraints and mean time to failure dependency on peak temperature and load balancing. The second is a heuristic algorithm to adjust the MPC sampling window length dynamically, thereby influencing the system's reliability via peak temperature and thermal cycle and the MPC computational overhead. This heuristic algorithm adjusts the sampling window length to balance these considerations.

The framework goal is to minimize the deviation of the core utilization from a set point in each time interval; this is called Sampling Window. The Sampling Window length is chosen so that a core's temperature can be considered constant within a sampling window. The RUC consists of two main components: a global utilization controller (GUC) and a sampling window controller (SWC). The GUC reduces the peak temperature by dynamically adjusting core frequencies to adhere to the utilization set point. The SWC minimizes thermal cycling wear by dynamically adjusting the length of the sampling window. Figure 2.23 shows a high-level overview of the RUC framework.

The evaluation experiments were conducted on a quad-core Cortex-A15 ARM chip: Nvidia's TK1. The quad-core ARM Cortex-A15 CPU and 192 Kepler GPU cores provide high performance with low power requirements. As a low-power chip, TK1 supports 12 different frequencies from 1.24 to 2.32 GHz. In order to evaluate RUC on different platforms, the authors used a hardware platform simulator. Then extracted the parameters from TK1 to build

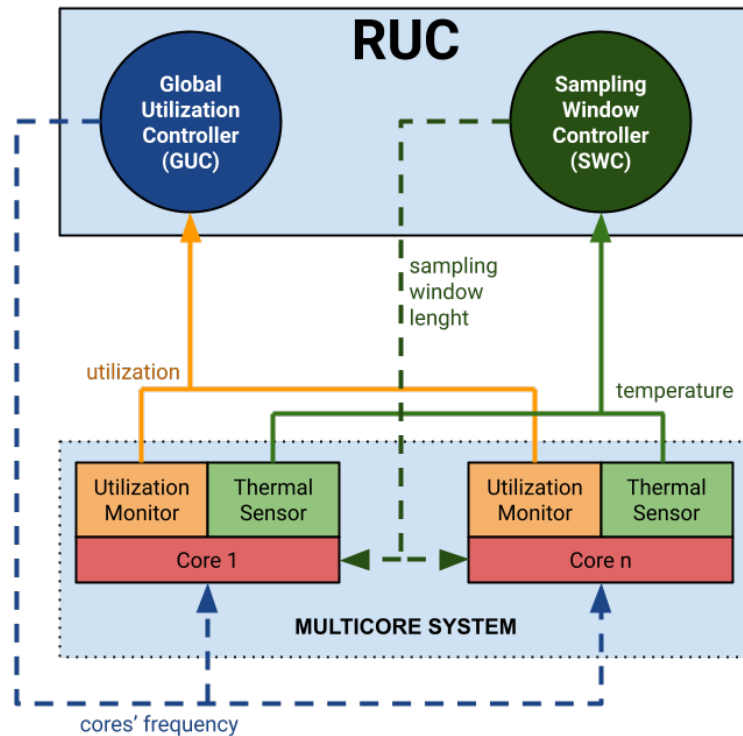


Figure 2.23: High-level overview of the RUC framework. Adapted from [MCDH17].

the power model. In the simulator, the temperature is obtained using Hotspot. The results revealed that, on all the platforms and with various task setups, this approach effectively increased the lifetime of soft real-time systems compared with existing temperature-aware and utilization control approaches.

### 2.2.9 Davide Zoni et al.

Zoni et al. [ZCF20] presents a control-theoretic scheme to design coordinated energy-budget and energy-allocation solutions for multicores, implemented into the RISC processor architecture. It was employed an online power monitoring infrastructure and a power-cap PID controller. In particular, a cascaded multi-level controller. The controller proposes a seamless integration of energy-budget and energy-allocation constraints from the applications and the Operating System, therefore addressing the popular runtime frameworks that support the self-adaptive application paradigm.

Figure 2.24 shows a representation of the structure of this control system. At the innermost level, a local control loop is implemented for each computing unit. The  $n$  numbered local controller of this loop regulates the control action to ensure the corresponding core's energy consumption follows the local setpoint value. The outer layer is a SISO global controller that generates a correction factor to the power budget starting from the difference between the global setpoint and the total consumed power in the considered time epoch.

The global power setpoint is set by either the operating system or the resource manager, and it represents the average energy consumption to enforce the required energy budget.

Outside these control loops, a supervisor algorithm implements an energy-allocation policy by modifying the SIMO box's coefficients. This controller regulates how much of the energy budget is assigned to each local controller in setpoint value. The supervisor algorithm can implement schemes or heuristics to shape a performance metric, ensuring great flexibility in the system's customization.

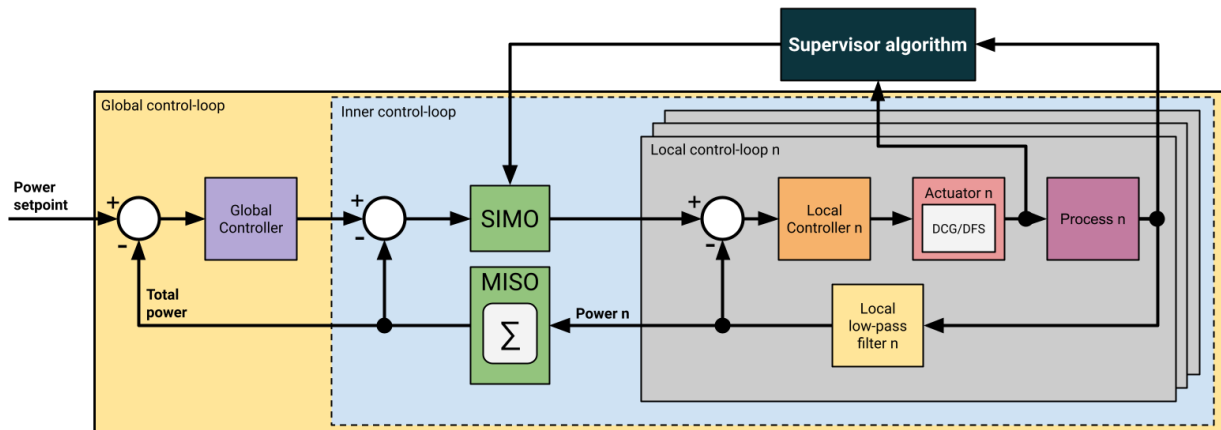


Figure 2.24: Closed-loop view of the proposed all-digital coordinated energy-budget and energy-allocation system. Adapted from [ZCF20].

As a reference computing platform, the proposed control scheme was implemented in quad- and eight-core Single Instruction Multiple Data (SIMD) RISC processors. The entire design is synthesized, implemented, and simulated at 50 MHz targeting the Digilent Nexys4-DDR board equipped with a Xilinx Artix-7 100t FPGA chip. To validate the system, three metrics were used to assess the quality of the proposed control scheme: the respect of the imposed energy budget, the performance loss due to the control scheme, and the quality of how efficiently the granted energy budget is exploited. All the obtained results showed that all of these metrics show significant improvement.

#### 2.2.10 Tiago Mück et al.

In [MDM<sup>+</sup>18], the authors present a methodology to design robust MIMO controllers with rapid response and formal guarantees for coordinated heterogeneous multicore processors (HMPs). Many times classic MIMO approach leads to controller designs that either lack robustness or manifest poor responsiveness, mainly due to unmanageable system identification complexity. An appropriate dynamic system modeling and decomposition strategy are

needed to account for the system size, heterogeneity of cores, and scope of the actuators and sensors in HMPs.

Therefore, this paper presents a methodology to design robust and responsive MIMO controllers for coordinated management of HMPs with formal guarantees. This objective is achieved by system modeling guidelines for formulating robust and responsive MIMO control of complex HMPs, including a set of properties for the system to be controllable, efficient, and robust. This enables the tuning of the controller by simplifying the identification of dynamic systems.

This paper develops a Linear Quadratic Gaussian (LQG) MIMO controller, and its main challenge is defining a system to identify and control this processor. The overall MIMO control design methodology for HMPs is proposed to achieve such a system, including guidelines for ensuring a more robust and stable closed-loop system. Figure 2.25 shows a representation of the methodology.

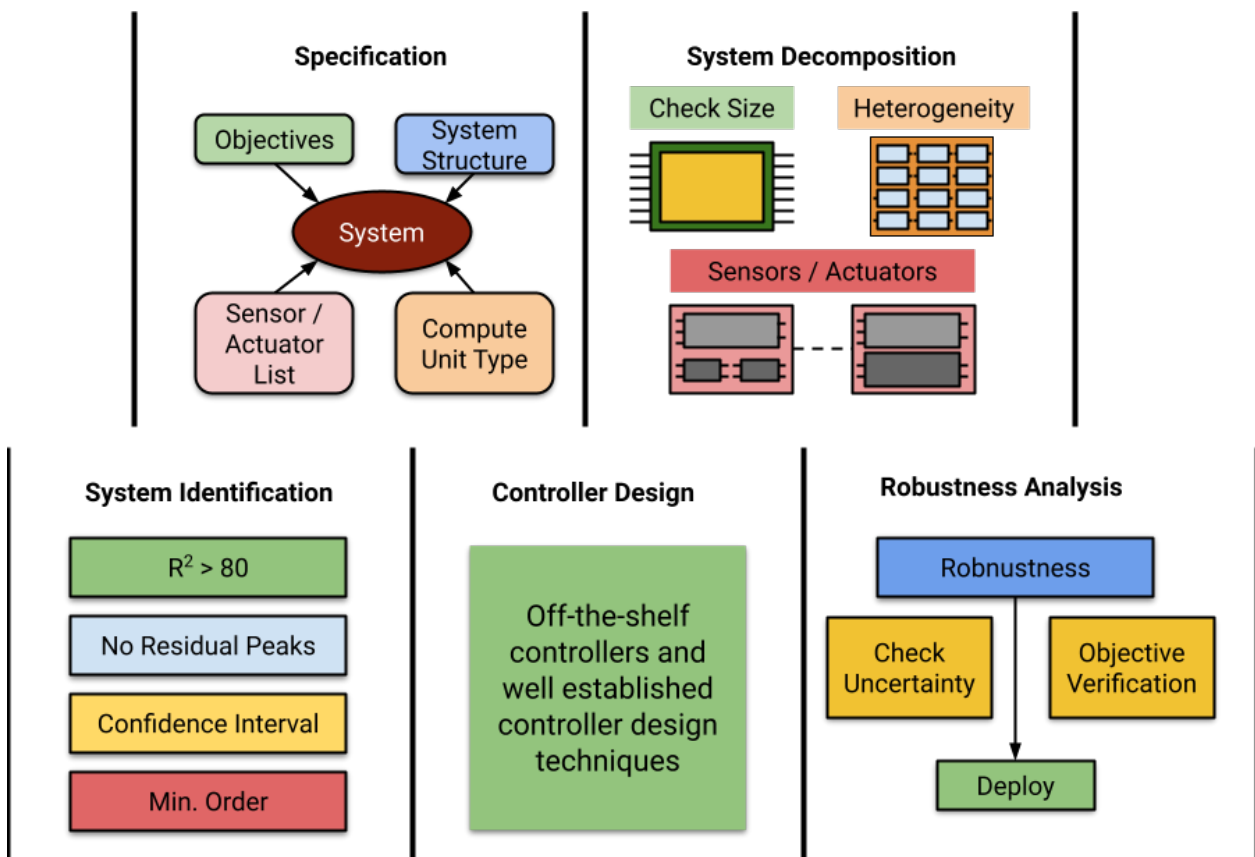


Figure 2.25: MIMO design methodology for HMPs. Adapted from [MDM<sup>+</sup>18].

This work divides the methodology into five distinct phases: Specifications, System Decomposition, System Identification, Controller Design, and Robustness Analysis. Next, the paper explains into greater details of these phases:

- **Specifications:** First, there is the necessity of specifying the management objectives of the system, the computer system structure, the compute unit description, and the list of sensors/actuators and their scope for the computer system.
- **System Decomposition:** Then, the user must find all the valid combinations of specifications that compose controllable subsystems for managing the desired objective(s). This process eliminates potential systems with an uncontrollable number of inputs and outputs or subsystems with insufficient actuators.
- **System Identification:** This is the essential step of this methodology. Here, all candidates found during system decomposition are modeled and evaluated in terms of their residual behavior and their associated fit to measured data and model order. Black-box system identification is performed to find system models exhibiting an acceptable fitting value.
- **Controller Design:** In this phase, the design of the LQG MIMO controller is implemented.
- **Robustness Analysis:** For Last, it checks if the controller can tolerate disturbance based on a defined uncertainty level while maintaining the specified confidence.

The proposed approach is evaluated using using the ODROID big.LITTLE HMP platform by following all steps of the methodology to generate predictable MIMO controllers. Then, the authors concluded that MIMO control is a promising technique for contemporary HMPs.

#### 2.2.11 Fardin Abdi et al.

In [ATR<sup>+</sup>17], the authors propose a controller that enables the system to restart and remain safe during and after the restart. This system provides fault tolerance and liveness in the presence of application-level faults and system-level faults using only one commercial off-the-shelf processing unit. This controller can keep the system inside a subset of the safety region only by updating the actuator input at least once after every system restart. After a restart, the system can restart again after only one command is applied to the actuators.

Figure 2.26 shows the logical view of this design, composed of three main components: the Base Controller (BC), Mission Controller (MC), and Decision Module (DM).

BC generates a control command that keeps the physical system stable by forcing its dynamic states to stay at a safety region where all physical constraints are respected.

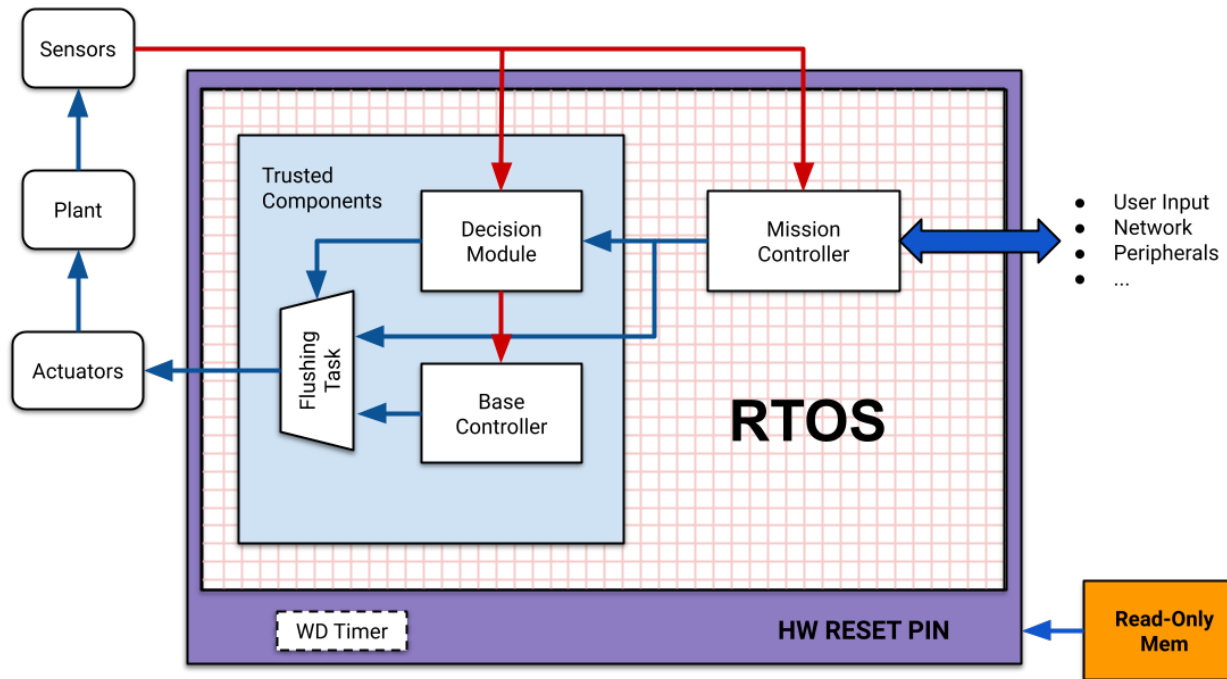


Figure 2.26: The logical view of the proposed framework. Adapted from [ATR<sup>+</sup>17].

Therefore, a BC, without any other components, can keep it that way by updating the actuator input at least once after every system restart.

The MC is the central controller of the system, which is concerned with mission-critical requirements. After MC runs and generates an output in every control cycle, the DM evaluates the safety requirements under this output and decides whether it can be applied to the actuators. Then, DM writes its output, along with the corresponding MC command and a timestamp to a fixed memory address.

At the end of each control cycle, the BC runs and generates a control signal output. Then a flushing task retrieves both control signals, from BC and MC, the DM's decision, and the corresponding timestamp from the memory. If the timestamp matches the current cycle, it updates the actuator commands with any of the signals based on the DM decision and resets the hardware watchdog timer. Non-matching time stamps indicate that one or both DM and BC tasks did not execute or missed their deadlines and triggers a restart.

To evaluate the proposed approach, the authors implemented a controller system for a three-degree-of-freedom (3DOF) helicopter and empirically verified fault-tolerance guarantees. It uses an i.MX7D SoC which provides two general-purpose ARM Cortex-A7 cores capable of running at the maximum frequency of 1 GHz and one real-time ARM Cortex-M4 core that runs at the maximum frequency of 200MHz.

Then, faults were injected in the control logic, the control application, and the operating system and demonstrated that the system remains safe, despite the faults, and recovers from these faults.



When the helicopter's state approached the states where the safety conditions were not satisfied, BC took over and prevented the helicopter from hitting the surface underneath.

### 2.2.12 Wei Li et al.

The central motivation of the study implemented in [LLZ<sup>+</sup>20] is that one of the core requirements of self-driving vehicles is environmental awareness. This feature requires mainly onboard sensors to sense surrounding environmental information and provides information support for autonomous driving car navigation and positioning, path planning, data fusion, and decision control.

This article uses the Tegra X2 embedded heterogeneous computing platform to process data coming from a LiDAR sensor, that usually presents a large number of point clouds. The structure of the Tegra X2 is represented in Figure 2.27.

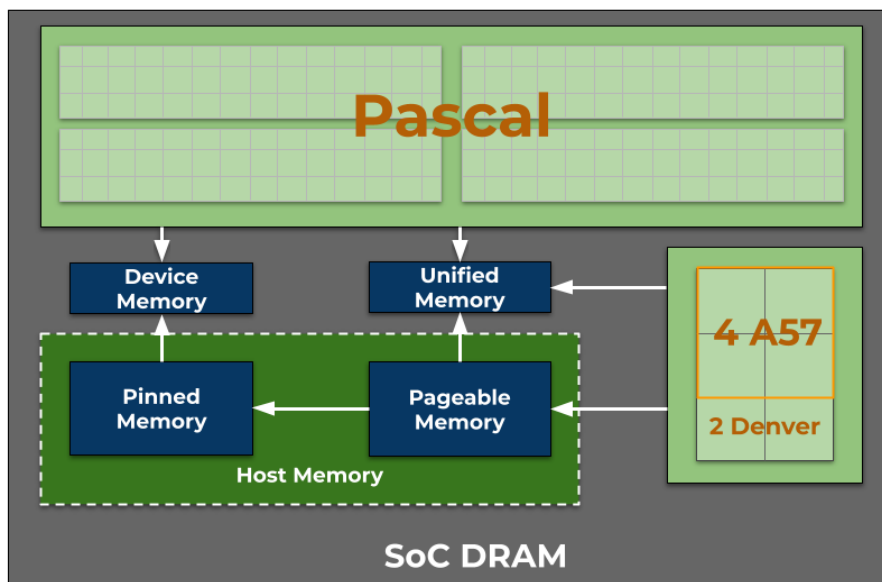


Figure 2.27: Tegra X2 architecture diagram. Adapted from [LLZ<sup>+</sup>20].

Based on Tegra X2's heterogeneous workflow of CPU and GPU collaborative asynchronous processing, this study implements offloading of LiDAR data processing algorithms and leaves the feature extraction and obstacle clustering tasks to the GPU. Moreover, the authors propose a dynamic task allocation method for thread load balancing.

The experiment devised by the authors to validate the system is the testing of the performance of the LiDAR data processing algorithm using a point cloud collected on an actual road as the experimental data. The algorithms run in the Tegra platform, and the results are compared to those obtained by an industrial personal computer (Inter(R) Core(TM), i7CPU M620). Comparing the proposed system with the industrial computer, the performance of the feature extraction step is improved by 4.5 times. Obstacle clustering enables

GPU optimisation, which improves the performance of this stage by 3.5 times. Finally, the performance of the overall algorithm is 2.3 times higher.

### 2.2.13 Zhenhua Jiang et al.

In [JB20], the authors present a multi-layer model predictive control (MPC) framework suitable for aerospace vehicles' integrated power/propulsion systems. The framework can decouple system-level energy optimization from dynamic power management in a holistic manner. This control platform can be built on distributed heterogeneous computing hardware, including real-time processors and FPGAs (field programmable gate arrays). The integral part of the MPC scheme is a real-time quadratic programming (QP) based optimizer (accelerated by FPGA) used to generate optimal control actions based on model prediction.

Figure 2.28 illustrates the proposed multi-layer model predictive control (MPC) framework. The framework decouples slow energy optimization from dynamic power management based on a heterogeneous real-time computing platform. This example considers a hybrid integrated power/propulsion system consisting of the main generator, an auxiliary power unit (APU), a battery energy storage system, main engines, and an ancillary electric propulsion system.

The control system includes:

- a host computer system as the user interface;
- a real-time computer for the higher-level MPC;
- and an FPGA (Field Programmable Gate Array) platform for the lower-level MPC.

The host computer system includes an interface communicating with real-time processors and FPGA hardware. The main component of the nonlinear MPC includes a real-time MPC computational solver that aims to find the solution to a nonlinear optimization problem based on a physical or a learning-based model.

To validate this framework, the authors chose to implement a simple model predictive controller in LabVIEW and an example application of the hybrid power system. The power system simulation model was migrated from MATLAB and redeveloped in LabVIEW. The MPC schemes perform specific computational steps in parallel to find the optimal control actions implemented in NI's CompactRIO modules and PXI FPGA modules. The focus of this system is a real-time quadratic programming (QP) based optimizer accelerated by FPGAs, used to generate the optimal control actions based on model prediction. This MPC optimizer aims to minimize a pre-defined quadratic objective function. Subjecting it to equality and inequality constraints and solving the convex optimization problem using the first-order optimality conditions.

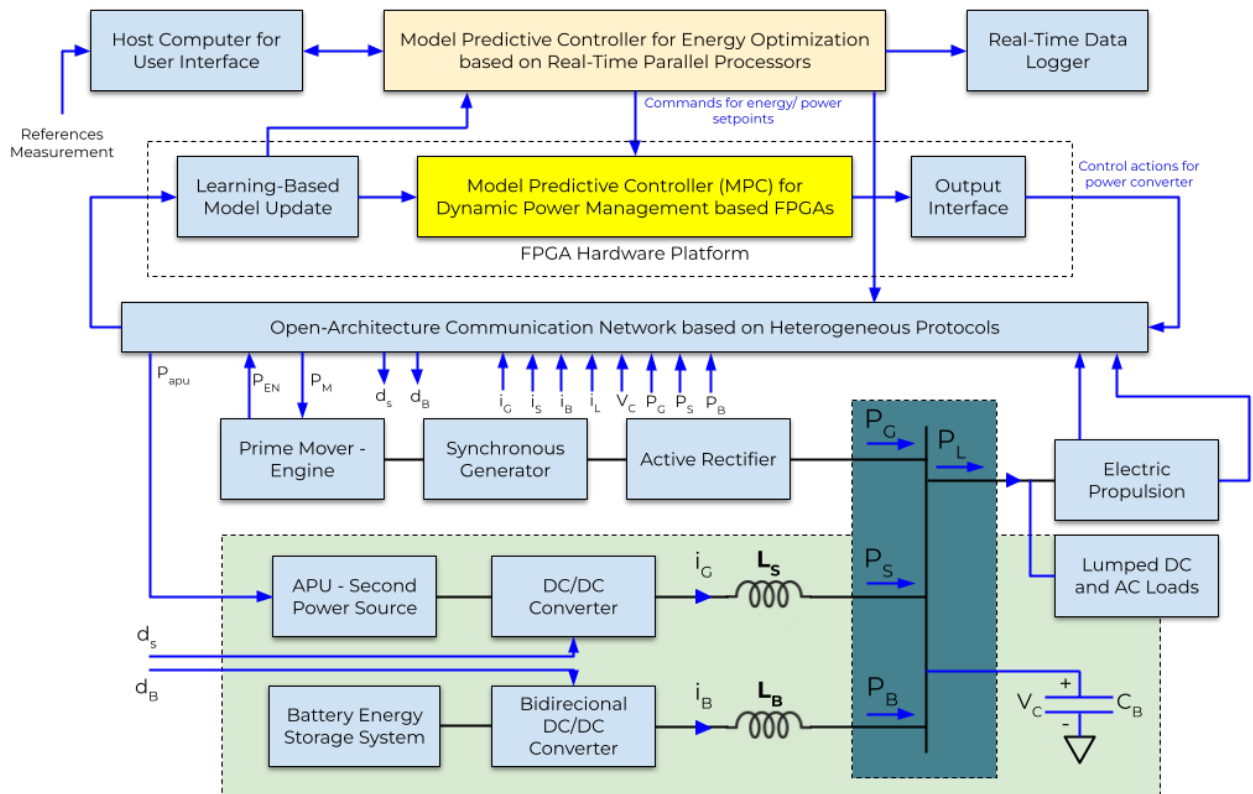


Figure 2.28: Multi-layer model predictive control framework that decouples slow energy optimization from dynamic power management based on heterogeneous real-time computing platform. Adapted from [JB20].

The hybrid power system of this study receives two power sources, each through a power converter, adapting the current and voltage to meet the load. The MPC aimed to optimize the current sharing between the two sources under reference changes and load disturbances. The results were compared with those under a PID controller.

Through graphical analysis, the experiment results showed that the MPC performs better than the PID in terms of voltage and current variations. It is also shown that the control actions (i.e., the duty cycle) are smoother with the MPC than with the PID. The simulation results show that the two-layer control system optimizes energy flow and dynamic current/voltage regulations under dynamic conditions.

#### 2.2.14 Yiming Gan et al.

Exploiting the hardware heterogeneities in mobile SoCs, the authors in [GQC+20] proposed a proactive computer vision execution model that breaks the sequential execution of the vision pipeline. Specifically, the system allows the pipeline front-end (sensing and imaging) to predict future frames. Furthermore, the pipeline back-end (vision algorithms) then predictively operates on the future frames to reduce frame latency.

Computer vision systems are bottlenecked by their long frame latency, which their serialized execution model fundamentally causes. The three major stages in a vision pipeline – sensing, imaging, and vision computation – process a frame sequentially, leading to high per-frame latencies. The authors present the PVF, a proactive mobile vision system that significantly reduces the end-to-end frame latency with lower energy consumption while mainly relying on existing mobile SoC hardware. The PVF exploits the hardware heterogeneities available on mobile SoCs, which naturally exposes different IP blocks (e.g., GPU, DSP, NPU) to execute multiple outstanding frames concurrently, mitigating resource contention.

The key idea of the predictive execution model is to allow the vision computation stage to operate speculatively on predicted future frames before the sensing and imaging stages generate the actual frames. Once an actual frame is generated, it is used to validate the predicted frame. If the predicted frame is checked to match the actual frame under specific metrics, the vision task results are likely already available and can be directly used, reducing the end-to-end frame latency. Otherwise, the speculated work is discarded, and the system executes the vision stage using the actual frame.

While the predictive execution model reduces the frame latency, it increases energy consumption for three reasons. First, speculation fundamentally trades energy for latency by performing extra work. Second, using multiple IP blocks while alleviating resource contention also increases energy consumption because CPU/GPU/DSP are less energy-efficient than NPU for executing vision algorithms. Finally, misprediction wastes energy on executing frames whose results are eventually discarded.

Figure 2.29 provides an overview of PVF. The goal of the PVF framework is to deliver a given latency target using the least energy while meeting the accuracy requirement. Accordingly, there are two main tasks of PVF: ensuring accuracy and delivering the latency target in an energy-efficient manner.

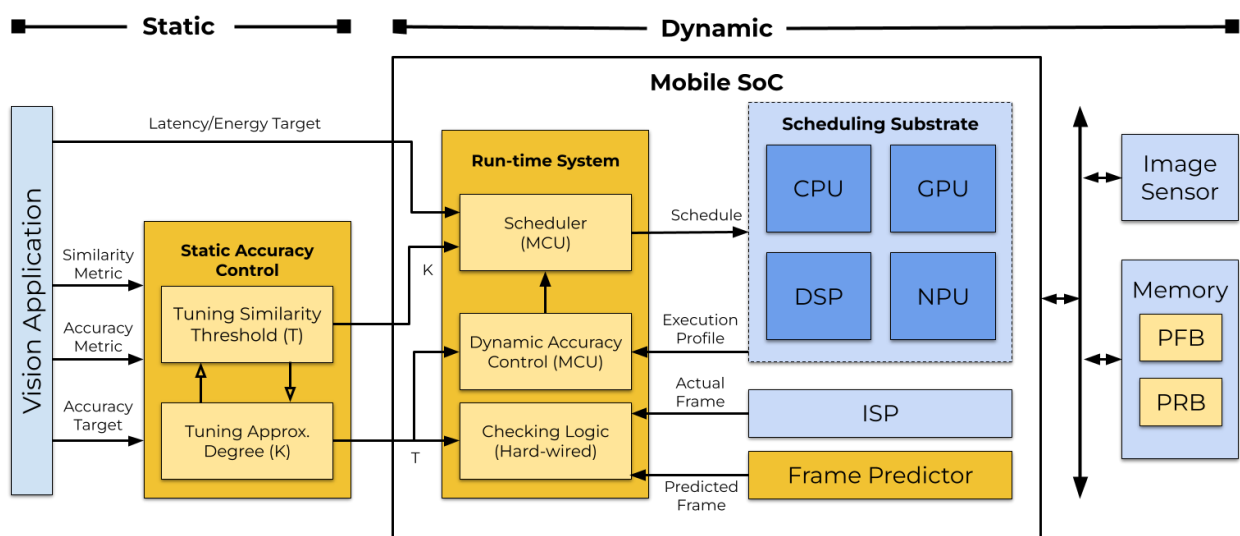


Figure 2.29: Overview of the PVF system. Adapted from [GQC<sup>+</sup>20].

At run time, a frame predictor first predicts multiple future frames, which are directly fed to the back end to perform vision tasks without waiting for the vision front-end. While a predicted frame could be executed by any IP block provided by the hardware, the authors propose a run-time design that intelligently maps the predicted frames to the IP blocks in a way that minimizes the overall frame latency while meeting a given energy budget.

The authors developed an in-house simulator with actual hardware measurements to model the continuous vision pipeline. The baseline model was a typical mobile SoC consisting of key IP blocks, including the CPU, GPU, ISP, and DSP. The ISP power is measured from the Nvidia Jetson TX2 module.

Compared to other baselines, the PVF provides significant latency reduction due to the ability to proactively execute future frames without waiting for the front end. Moreover, the PVF reduced the frame energy consumption under given latency constraints.

### 2.2.15 Jie Tang et al.

In [TLL<sup>+</sup>20], the authors proposed a system that enables multiple autonomous driving services on affordable embedded systems. The LoPECS a Low-Power Edge Computing System for real-time autonomous robots and vehicle services. According to the authors, there are three main contributions provided by this study:

- The development of a Heterogeneity-Aware Runtime Layer to schedule autonomous driving computing tasks to heterogeneous computing units for optimal real-time performance;
- The development of a vehicle-edge coordinator to dynamically offload tasks to edge cloudlet. In order to optimize user experience in autonomous driving in terms of lower power and extended battery life;
- Integration of the previous components into the proposed LoPECS system and implementation of it on Nvidia Jetson TX1.

Figure 2.30 shows the LoPECS architecture. At the application layer, LoPECS supports localization, obstacle detection, speech recognition, and more. These services support safe, efficient, and real-time driving behaviors. At the top, a layer named QoE (Quality of Experience) Oriented Service Classification receives the data coming from the applications. It can classify different autonomous driving services into QoE-Time, QoE-Insensitive, and QoE-Energy. Such grouping is based on the service's features in real-time requirements and demands for energy cost.

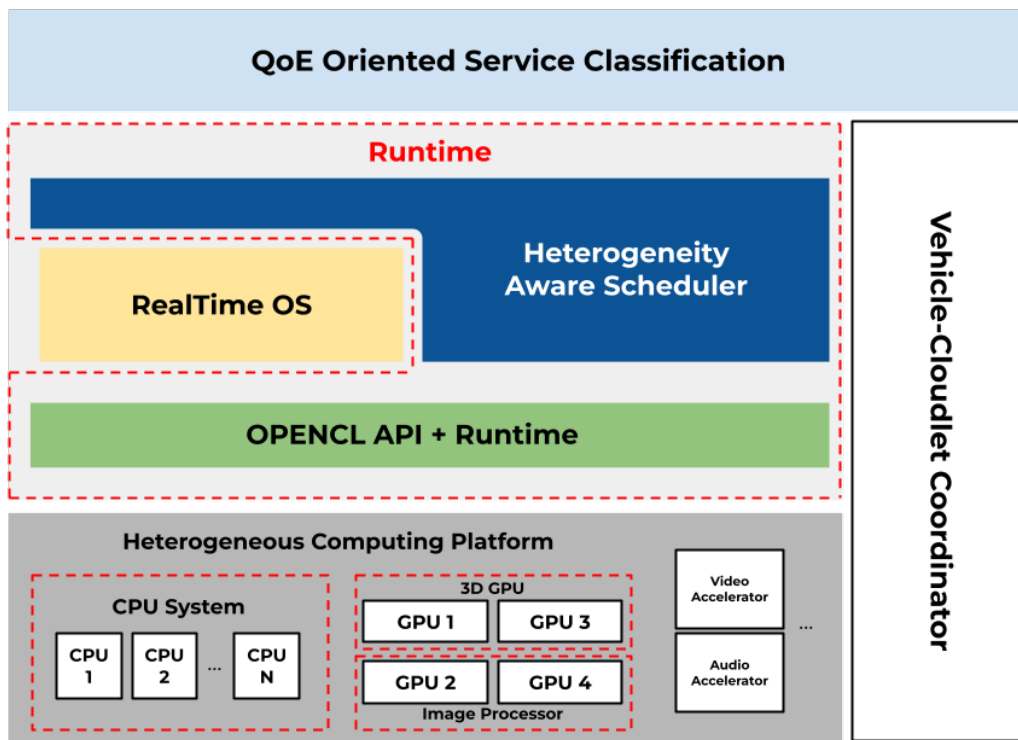


Figure 2.30: LoPECS architecture. Adapted from [TLL<sup>+</sup>20].

To integrate these services, the authors developed the Realtime OS. This lightweight operating system manages various services and facilitates communications with almost zero overheads. Realtime OS serves as the primary communication backbone.

Below Realtime OS is the LoPECS Runtime layer, which implements two functions: first, it provides an abstraction of the underlying heterogeneous computing resources through and provides acceleration operations. Furthermore, the second function implements a Heterogeneity Aware Scheduling algorithm to manage the mapping of tasks on heterogeneous hardware systems.

Also, to effectively control the energy consumption of autonomous vehicles, LoPECS contains a Vehicle-Cloudlet Coordinator to dynamically offload some tasks to the cloud to achieve optimal energy efficiency.

Figure 2.31 shows a representation of the implementation of the runtime layer to map various tasks onto the underlying heterogeneous computing units dynamically. This runtime layer is crucial to simultaneously enable multiple autonomous driving tasks on computing and energy resource-constrained edge computing systems.

To manage heterogeneous computing resources was utilized OpenCL. An open standard for cross-platform, parallel programming of various computing units. OpenCL provides the interface for LoPECS to dispatch various applications to the underlying computing resources. On top of OpenCL, it was designed and implemented a Heterogeneity Aware Scheduler to manage and dynamically dispatch incoming tasks.

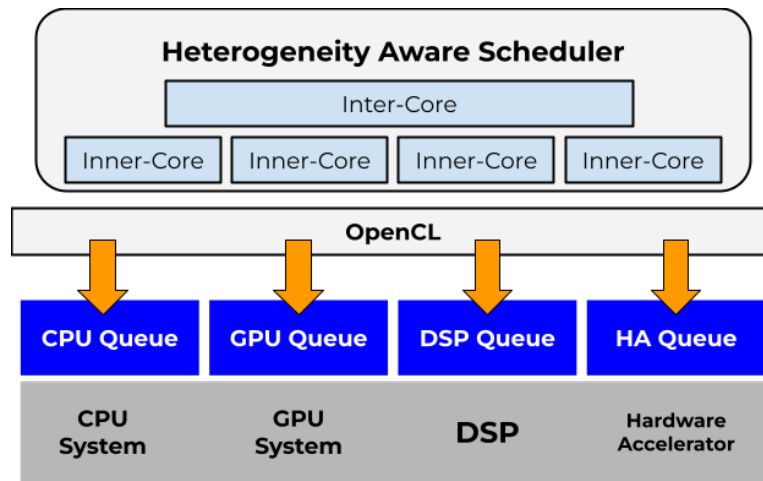


Figure 2.31: The LoPECS runtime design. Adapted from [TLL<sup>+</sup>20].

To validate this study, as stated before, the LoPECS system was implemented on an Nvidia Jetson TX1. The experiment consisted of processing three types of sensors: an IMU, a camera, and a microphone. Later, the system ran SLAM, speech recognition, and computer vision algorithms. The authors considered the experiments successful based on:

- The SLAM pipeline can process images at 18 FPS. The authors consider that once the SLAM pipeline can process images at more than 15 FPS, the localization service is stable;
- 10 FPS achieved in image recognition. For low-speed autonomous driving applications, the authors consider 10 FPS satisfactory;
- For speech recognition, the system can convert an audio stream into words with 100 ms latency. The authors determined a 500 ms latency baseline for such tasks.

## 2.3 State-of-the-art Analysis

Tables 2.1 and 2.2 summarizes the reviewed works according the thesis core idea and subtopics showed in Figure 2.13 and the research questions proposed in Section 2.2. The second column presents the heterogeneous architecture (Multicore, CPU-GPU, big.LITTLE, and more) used in each work. The table columns are described as:

- *3rd Col:* lists the different techniques applied related to control theory;
- *4th Col:* the applications selected by each work for the study cases;
- *5th Col:* highlights if the work approach intra-chip energy management;

- *6th Col:* classifies if the proposal implements some form of distributed processing regarding the control environment;
- *7th Col:* highlights any form of support software framework or library used to construct the system in question;
- *8th Col:* shows if the work applies any fault tolerance method.

The literature presents an almost even distribution of heterogeneous architecture between works, as shown in Tables 2.1 and 2.2 (2<sup>nd</sup> column). It exemplifies the strength of heterogeneous computing, how many forms and arrangements are possible under this methodology. This thesis opted for a MPSoC architecture as the platform where the framework was built upon. Like the works in [DO18] and [ZCF20].

The control implementation column (3<sup>rd</sup> column) exhibits the sense-control-act paradigm module highlighted in each work. To exemplify this point, we can point at the works of Kusainov et al. [KKSC18], where a nonlinear model predictive control and how it is processed is the focal point of the paper, whereas Tang et al. [TLG17] focus on the distributed processing of a robotic sensory system.

In the control side of this literature review, we can divide the controller by type: linear control, nonlinear control, and learning-based control. Giardino et al. [GKFF20] and Zoni et al. [ZCF20] approaches the linear types by applying some form of PI - PID control. The works of Khusainov et al. [KKSC18], Deng and Ohtsuka [DO18], Jiang et al. [JB20] and Ma et al. [MCDH17] opted for MPC nonlinear control. Moazemmi et al. [MMY<sup>+</sup>19] implement a Fuzzy controller for intra-chip resource management in the learning-based control category. Lastly, Abdi et al. [ATR<sup>+</sup>17] does not define a specific controller; the authors give a requirements guideline for the controller implementation.

Ulbricht et al. [USK19] propose a fault-tolerant multicore system for optimized sensor processing using DSP cores to guarantee different levels of fault tolerance. In Liu et al. [LTL<sup>+</sup>21] the authors implement a computer vision and inertial sensors fusion to perform autonomous navigation based Snapdragon 820 processor. Furthermore, Tang et al. [TLG17] implements a SLAM algorithm in a mobile heterogeneous SoC balancing performance and energy consumption.

The control application column (4<sup>th</sup> column) shows which system is controlled in each work. Each application represents the plant block showed in Figure 2.2. Inside this category, we can divide it into two distinct subcategories: outer-chip applications and intra-chip applications. To better explain this choice of categorization, we can describe the outer-chip category as the "traditional" control paradigm (see Figure 2.3), where the processing unit only role is to process the sensor's and controller data, the plant is a dynamic system outside of the realm of processors. In this review, we can highlight the works that refer to these kinds of controlled systems. Liu et al. [LTL<sup>+</sup>21] and Tang et al. [TLG17] use



heterogeneous processing to control mobile robots in their research. As shown in the Tables 2.1 and 2.2, there are many other implementations of this nature.

In contrast, the intra-chip control makes the processing unit itself the plant to be controlled. Moazemmi et al. [MMY<sup>+</sup>19] propose a system based on an NVIDIA Jetson TX2 platform capable of managing its resources, like the numbers of active cores or its frequencies, using fuzzy control. Ma et al. [MCDH17] also propose a control system to manage inner processor resources. Zoni et al. [ZCF20] and Mück et al. [MDM<sup>+</sup>18] propose intra-chip energy management control systems.

Combining these two application categories, we can highlight the work of Giardino et al. [GKFF20] where the authors propose two distinct yet intertwined control systems. The processor energy budget is controlled, and it is investigated the effects has on the navigation control of a mobile robot. This thesis has a similar approach to how internal chip control affects the plant's control quality-of-service (QoS).

The fifth column shows which papers approach the topic of energy management. As shown in the fourth column, the works of Giardino et al. [GKFF20], Zoni et al. [ZCF20] and Mück et al. [MDM<sup>+</sup>18] focus solely in energy intra-chip control. The works of Moazemmi et al. [MMY<sup>+</sup>19] and Ma et al. [MCDH17] approach the topic of energy control among other resources. Moreover, the research of Liu et al. [LTL<sup>+</sup>21] does not apply energy control, but it uses energy sensing to evaluate the proposed system.

In the decentralized control processing (6<sup>th</sup> column), we arrange the works by how the controller or the whole control system is processed. For example: in Khusainov et al. [KKSC18] the authors break the model predictive control problem into two algorithms, one running in the CPU and another in an FPGA. In Liu et al. [LTL<sup>+</sup>21] the sensor fusion for the robotic system is distributed over DSP, GPU, and multicore system.

The seventh column shows if any of these works use or propose any auxiliary software or libraries to implement their control systems. In Deng and Ohtsuka [DO18] the authors presented a MATLAB-based software that generates parallel C/C++ code and carry out closed-loop simulation for nonlinear model predictive control, built for shared-memory multicore processors. Giardino et al. [GKFF20] use a C library for scientific computing called GNU Scientific Library for the mathematical functions and matrix operations needed by the path planner and trajectory tracking. In this thesis, we propose combining these two examples to create a software framework to implement different controllers intra and outer-chip of a multicore system.

Finally, the last column (8<sup>th</sup> column) presents which work approaches the topic of fault-tolerance. Ulbricht et al. [USK19] propose a sensory system that targets automated driving and is fail-safe, fail-operational, and distributed execution of different tasks. All while keeping the strict timing and safety constraints. Furthermore, Abdi et al. [ATR<sup>+</sup>17] propose a controller design that enables the system to restart and remain safe during and after the restart. This design tolerates faults in the underlying software layers such as RTOS and mid-

deware and recovers from them through system-level restarts that reinitialize the software from a read-only storage.

In this thesis, we chose these categories, spread out into these columns, to propose a software framework that partially addresses all of these characteristics into one system, shown in the subsequent sections.

Table 2.1: State-of-the-art Analysis Table

Proposal	Hetero. Arch.	Control Implementation	Application	Energy Management?	Decentralized Control?	Software Support?	Fault Tolerance?
Khusainov et al. [KKSC18]	CPU-FPGA	Nonlinear Model Predictive Control (NMPC)	Gantry Crane System	✗	✓	Protoip, MATLAB	✗
Moazemmi et al. [MMY <sup>+</sup> 19]	CPU-GPU	Fuzzy Control	Intrachip Resource Management	✓	✗	MATLAB	✗
Giardino et al. [GKFF20]	big.LITTLE	PI Controller	Mobile Robot/ Intrachip Energy Management	✓	✓	GNU Scientific Library	✗
Shahhosseini et al. [SMRD18]	Multicore	PI Controller	Intrachip Energy Management	✓	✗	SNIPER Simulator	✗
Ulbricht et al. [USK19]	DSP-Multicore	Sensor Processing	Automated Driving	✗	✓	✗	✓
Liu et al. [LTL <sup>+</sup> 21]	DSP-GPU-CPU	Sensor Fusion	Mobile Robot	✓	✓	✗	✗
Deng and Ohtsuka [DO18]	Multicore	Nonlinear Model Predictive Control (NMPC)	Double Inverted Pendulum/ Quadrotor	✗	✓	ParNMPC	✗
Tang et al. [TLG17]	DSP-GPU-CPU	Sensor Processing	Mobile Robot	✗	✓	✗	✗
Ma et al. [MCDH17]	CPU-GPU	Model Predictive Control (MPC)	Intrachip Resource Management	✓	✓	✗	✗
Zoni et al. [ZCF20]	Multicore	PID Controller/ Cascade Control	Intrachip Energy Management	✓	✓	✗	✗
Mück et al. [MDM <sup>+</sup> 18]	big.LITTLE	Linear Quadratic Gaussian Control (LQG)	Intrachip Energy Management	✓	✗	✗	✗
Abdi et al. [ATR <sup>+</sup> 17]	Multicore	Not Specified	3DOF Helicopter	✗	✓	FreeRTOS	✓

Table 2.2: State-of-the-art Analysis Table (Continuation)

Proposal	Hetero. Arch.	Control Implementation	Application	Energy Management?	Decentralized Control?	Software Support?	Fault Tolerance?
Jiang et al. [JB20]	Real-Time processors and FPGAs	MPC	Propulsion System of Aerospace Systems	✓	✓	LabVIEW, MATLAB	✗
Li et al. [LLZ+20]	CPU-GPU	Sensor Processing	Self-Driving Car	✗	✓	✗	✗
Gan et al. [GQC+20]	CPU - GPU - DSP	Sensor Processing	Computer Vision	✓	✗	In-house Simulator	✗
Jie Tang et al. [TLL+20]	CPU - GPU - DSP	Sensor Processing	Autonomous Vehicle	✓	✓	LoPECS	✗
<b>This Thesis</b>	Multicore	PID / Linear Quadratic Integral Control (LQI)	Intrachip Energy Management/ Quadrotorr	✓	✓	Original Work	✓

### **3. THE QUADROTOR CASE STUDY**

The quadrotor control is chosen as a case study, although the proposed software framework is developed to have the capability to be implemented in many other robotic applications; this chapter presents various quadrotor-related topics, ranging from the quadrotor's fundamental physics to the theory of the algorithms used in the control systems. Section 3.1 gives an introduction, presenting a basic overview of the quadrotor, defining concepts and motivations. Section 3.2 presents a description of the physics of the quadrotor and its possible flight modes. Once it was chosen to implement the proposed control system in a simulated quadrotor, Section 3.3 lists the available quadrotor simulators and explains why the selected simulator was chosen. Then, Section 3.4 shows the structure of the robotic operating system (ROS) on which the selected simulation is based.

As an important aspect of a control system, the topic of sensing, is addressed in Sections 3.5 and 3.6. Section 3.5 lists the most used sensors in the quadrotor and explains why specific sensors were chosen for this work. Then Section 3.6 presents the theory behind the sensor fusion used in the system. Moreover, before developing the control system, it is necessary to model the quadrotor system mathematically. Section 3.7 presents the modeling in which the controllers are based followed by Section 3.8 lists the most commonly used controllers for quadrotors found in the literature. Section 3.9 presents the theory of the selected controllers used in this work. Therefore, this chapter aims to link the theory related to the quadrotor with the control theory. These relations are detailed in Section 3.10.

#### **3.1 Introduction to Quadrotors**

In recent years there has been a lot of discussion about driver-less cars roaming our streets, flying drones that deliver our internet purchases right to our doors and autonomous submarines capable of protecting our shores. But what exactly are these autonomous vehicles? The most common definition of an autonomous vehicle is a transport machine able to operate itself, perform its required functions without human intervention, and sense and react to its surroundings. The main advantage of these systems can be seen as the lack of human interference in its operation. Humans are prone to errors and misjudgments. Such system can reduce failures.

There is a wide array of possible applications for autonomous vehicles, either moving over land, across the water or through air. In the ground, driver-less cars take people from their house to work. Small vehicular robots traverse the factory floor transporting resources and products. In the water, autonomous boats map the shoreline, and sub-aquatic

vessels inspect pipelines. That been said, one of the main topics in this Thesis is in the aerial space of autonomous actuation, more specifically in unmanned aerial vehicles or UAVs.

As its acronym suggests, UAVs are aerial vehicles that do not require a human on board to control it. Even though a remotely-controlled system can sometimes be called a UAV, in this work considers only autonomous UAVs.

There are many forms and configurations to such vehicles: blimps, balloons, fixed-wing aircraft, rotary-wing aircrafts and even flapping-wing aircrafts. Again, this work limits attention to rotary-wing UAVs, more specifically the quadrotor type/kind.

A quadrotor is an aircraft lifted and propelled by four rotors. Its growing popularity can be attributed to many advantages presented over counterparts, due to reduced mechanical complexity, hover capability, high maneuverability, VTOL (Vertical Take-Off and Landing) ability, and easy maintenance. The quadrotor is a straightforward mechanical structure due to its configuration that consists of a rigid cross frame with a rotor placed in each frame end. The quadrotor fundamental dynamic dictates that the vertical motion is generated by the increase (upwards) and decrease (downwards) of each rotor at equal speeds. This dynamic is possible since each spinning rotor creates a perpendicular force in the vertical ( $z$ ) axis.

Then, what is the motivation behind choosing the quadrotor as a case study for this Thesis? There is a fourfold answer to this question. First, the author chose a system that demands fast processing, a feature critical in a quadrotor due to its fast dynamic nature. Fault tolerance is the second reason to justify the use of quadrotor. Even still, the quadrotor has its rotors in full actuation, and any fault during flight can result in severe consequences. The third reason is energy management. As said before, the quadrotor, while on the air, does not stop consuming energy. In a system where any extra weight put a stranglehold on performance and adding batteries are not a solution, energy management is crucial.

According to Morbidi et al. [MCL16], the quadrotor system suffers from a significant limitation: reduced flight endurance, typically between 15 and 30 minutes. Some promising new applications (package delivery, cinematography, aerial manipulation) have lately emerged: however, the limited runtime of the existing lithium-ion polymer (LiPo) batteries strongly restricts the class of missions that a rotorcraft can successfully carry out.

Lastly, the fourth reason is tied to the dynamical modeling of the quadrotor. Due to its non-linearities, robust non-linear control is necessary to guarantee a safe and effective performance.

One of the few disadvantages of UAVs, unlike ground vehicles, is that the former never stop consuming energy, because they must maintain their own weight in the air the entire time. This is even more critical in rotary-wing UAVs, since they do not have enough wing surfaces providing a lift phenomenon (as in fixed-wing UAVs), and all the thrust must be generated by the propulsion system [GSBT16]. Combined with the energy consumption

of the processing elements responsible for the vehicle's control, there is a serious problem that influences a quadrotor design and construction.

The central goal of this thesis is to develop a software framework for the implementation of controllers in a MPSoC, in the field of robotics. Therefore, this system must address fault tolerance, energy management, decentralized processing, and high demand for processing large quantities of sensor data. So, considering the quadrotor's characteristics listed above, the quadrotor is an adequate study case for this work.

## 3.2 Quadrotor Dynamics

The quadrotor holds a very simple mechanical structure, while conventional helicopters normally retain variable pitch rotors and extremely complex mechanical control structure [EN18]. The quadrotor configuration consists of a rigid cross frame with a rotor placed in each of the frame's end showed in Fig. 3.1. The fundamental dynamics of the quadrotor dictates that the vertical motion is generated by the increase (upwards) and decrease (downwards) of each rotor at equal speeds. This dynamic is possible since each spinning rotor creates a perpendicular force in the vertical ( $z$ ) axis.

To cancel the yaw drift movement created by four rotors spinning in the same direction, in the quadrotor, two rotors placed at opposite ends spin at one direction, and the other pair spin at the other. To achieve the desired yaw movement while maintaining height, the pair of rotors responsible for a particular spin direction, increase their speeds equally. In contrast, the other pair decrease their speeds, all the while the total thrust must be the same to avoid the up-down movement.

The roll movement ( $\phi$ ) is done by the rotor pair located in the  $y$  axis, one of the rotors increases its velocity while the other decreases. This dynamic makes the quadrotor spin around the  $x$  axis. Similarly, the pitch movement ( $\theta$ ) is performed by the rotor pair located in the  $x$  axis, one of the rotors increases its velocity while the other decreases. This dynamic makes the quadrotor spin around the  $y$  axis.

A quadrotor has six degrees of freedom and only four actuators, making it an underactuated mechanical system with a degree of underactuation of two [EN18]. Therefore, a single controller could not control every state.

### 3.2.1 Flight Modes

This section presents the operating modes of the vehicle that the author deems to be the most relevant to represent a fully operational quadrotor. Six main categories de-

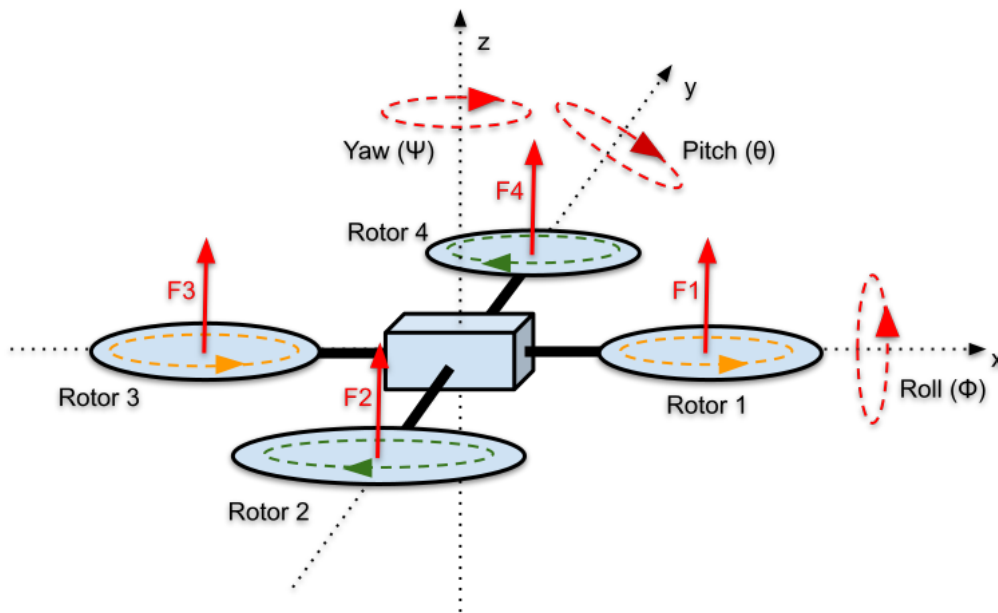


Figure 3.1: Diagram of forces and movements of a Quadrotor.

scribe how the vehicle should behave, from the more straightforward modes like hover and trajectory following to the most complex, like crash recovery and rejection of disturbances. Ultimately, this Thesis focus only in the hover mode and line trajectory, due to their simplicity.

- **Hover mode:** In this mode the aircraft hovers above the ground in a controlled environment, so disturbances like ground effect, chimney effect and wind gusts are not factors to be considered in the project phase of the control system implementation.
- **Line trajectory and circular trajectory:** Here the main concern is to guarantee that the quadrotor can follow a straight line smoothly and a circular trajectory around a given point in a Cartesian plane.
- **Collision avoidance/recovery:** A critical feature of an autonomous vehicle is the ability to navigate a cluttered environment avoiding obstacles and possible hazards. This capability generally requires that the quadrotor perform some complex maneuvers rapidly, demanding specific control schemes. Another essential feature that a quadrotor project should consider is in the event of a crash, in which the form of control is more suitable to recover the stability of the drone, considering two scenarios: when damage to the system has happened (i.e. motor failure), and when the system remains intact.
- **Cargo load/moving/landing:** As the research on quadrotors advances, grows the demand for the use of drones to carry payloads for a number of reasons. This means drastically changing the quadrotor's dynamic model. These changes must be dealt with by the controller in a way that guarantees a satisfactory stability to the system independent of the weight, size or the way that the cargo is transported.



- **Ground effect/Chimney effect:** Operating a quadrotor near ground introduces a new difficulty in controlling the vehicle that is known as *ground effect*. Ground effect is an aerodynamic phenomena which reduces the induced drag of an aircraft and thereby increases its lift-to-drag ratio. It is the lift increase generated by rotors when an aircraft is close to a surface [DYZG15]. The wall effect on the aerodynamics of the propulsion system is similar to the ground effects. The chimney effect happens when a quadrotor is fully surrounded by walls. Depending on the distance from the wall, the dynamic of the quadrotor varies due to the changes in aerodynamic coefficient [GAAA16].
- **Hover, takeoff and landing under harsh conditions:** Besides the operating modes previously mentioned, it is important to consider that the majority of applications is not be carried out in a controlled environment that means uncertainties and disturbances like wind gusts is a factor in the quadrotor control system.

### 3.3 Computational Simulations

When dealing with a new project with new technologies, it is wise to rely on computer simulations as the first step of development. There are many reasons to use simulations instead of straight prototyping. Simulating guarantees a better design exploration; it is easier to set up, significantly less expensive, faster and eliminates the risk of possible injuries due to accidents.

Today, there is a great variety in of choices concerning resources for quadrotors simulation, each with its strengths and inconvenients. Figure 3.2 abstracts the main simulator resources available for quadrotors at the time of the Thesis writing. Simulation resources are classified here in four categories: (i) those based on Matlab; (ii) the ones based on ROS; (iii) those based on the *Unreal Engine* [Eng20]; (iv) other simulation resources.

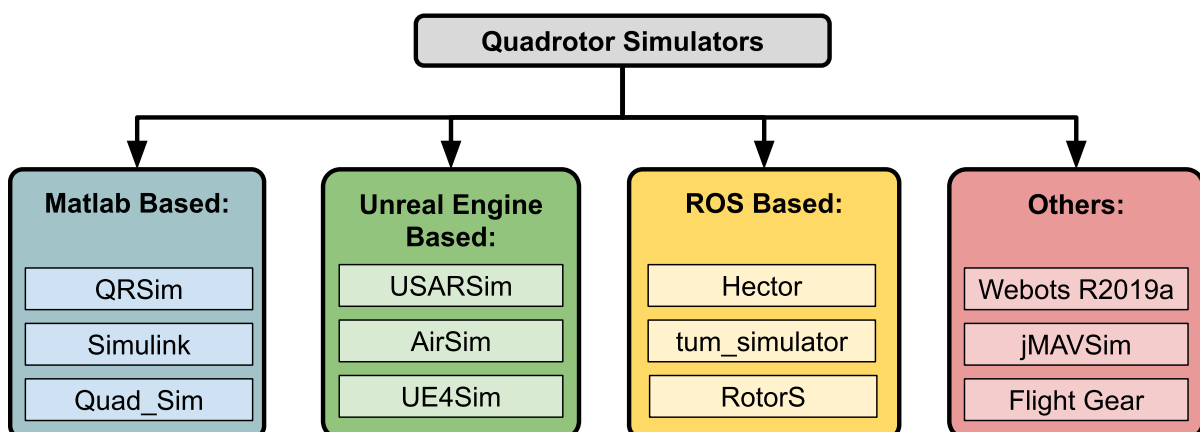


Figure 3.2: Main simulators of quadrotors.

Matlab is a language for technical computing; it integrates computation, visualization, and programming in a simple environment describing problems and solutions in mathematical notation [Mat05]. In control engineering, Matlab is frequently used due to its simplicity to describe, model, and simulate dynamic systems. Matlab also enables a simple interface for data analysis, exploration, and visualization. In the field of quadrotor simulation, the Simulink Toolbox is an intuitive way to describe the quadrotor dynamic model and also implement a control system, as seen in [KO19]. Other implementations of quadrotor simulators are QRSim [De 13] and Quad\_Sim [HLM<sup>+</sup>14], These are implemented in a series of m-files that describe the dynamic models, sensors and environments. This work does not consider any of these simulators, because although their scripts are open source, Matlab is not. Another factor that influenced the decision is the computation cost that Matlab puts in the overall simulation system.

A second category of considered for robotic simulations are those based on video game engines, more specifically, the *Unreal Engine*, produced by Epic Games, mainly to create first-person shooter games. As video game technology progresses, each year, more realistic games are developed, improving on graphics and physics, it is natural that researchers are reaching for this kind of technology to conduct robotic simulations. Game engines are general-purpose simulators, the base framework for the development of a variety of games. In [CLW<sup>+</sup>07], authors propose a high fidelity robotic simulator built upon the *Unreal Engine 2.0*. The simulator is called USARSim (Urban Search and Rescue Simulation). Although it was created for ground vehicles, there are many robotic models implemented using it, including underwater vehicles, humanoid robots, and UAVs. In February of 2017, Microsoft researchers released a simulator for autonomous quadrotors focused mainly on machine learning called AirSim [SDLK18]. This simulator is built upon the *Unreal Engine 4.0*, taking advantage of its visual high fidelity to implement sensors like depth-cameras that produce data for deep learning applications. [MCL<sup>+</sup>18] and AirSim [SDLK18] propose a photo-realistic simulator based on *Unreal Engine 4.0* for computer vision applications such as object tracking, object detection, autonomous navigation, and multi-agent collaboration. The reason for not considering using simulators based on video game engines is the fact that this kind of simulator demands host systems with heavy-duty GPUs for executing. Also, the internal structures of these engines are proprietary and third-party interface software is needed for information exchange.

The Robot Operating System (ROS) [QCG<sup>+</sup>09] became a *de facto* standard for the development of robotic systems. ROS-based systems implement the publish-subscribe pattern [CDK05], whose organization consists of nodes and topics. A node is a software that can subscribe to (or publish to) data channels called topics. Typical setups for robot simulation combine ROS with Gazebo [Fou14]. Gazebo is a robot simulator capable of simulating environmental physics, e.g., gravity, wind, terrain, and sunlight. Robots interact with the environment by sensors, which capture data based on the stimuli of the simulated envi-

ronment. Gazebo can be extended to push data into ROS so that applications can access sensing information. Then, the behavior of the robot can be programmed and validated as if the software were deployed to a real robotic platform. In [FBAS16] is shown a multi-rotor Gazebo simulator developed by Autonomous Systems Lab, ETH Zurich called RotorS. This simulator simulates a few models of multi-rotor helicopters, such as AscTec Hummingbird and Crazyflie. It is designed in a modular way that different controllers and state estimators can be used interchangeably. In [NR19] uses RotorS and Matlab Simulink to implement a hover control and tracking trajectory. In [HS14], the authors developed an AR.Drone simulator based on the ROS package shown in [MSK<sup>+</sup>12].

Other simulations that were also considered for this work that doesn't fit in the first three categories. In [Oli04] is presented an Open Dynamics Engine (ODE) based simulator called Webots developed by Cyberbotics Ltd. The FlightGear flight simulator [OMT97] is an open-source project developed by volunteers since 1997. It supports a variety of popular platforms (Windows, Mac, Linux, etc.), and its source code for the entire project is available and licensed under the GNU General Public License. The last simulator considered is the PX4 developed jMAVSim [Fou20]. This is a simple quadrotor simulator that uses SITL (Software-In-The-Loop) PX4 autopilot software.

In the next section the author defines and justifies the quadrotor simulation chosen.

### 3.3.1 Hector Quadrotor

In [MSK<sup>+</sup>12] is the detailed description of the modeling and implementation of this simulator. In this work, the author gives an overall explanation of the system and how it was implemented.

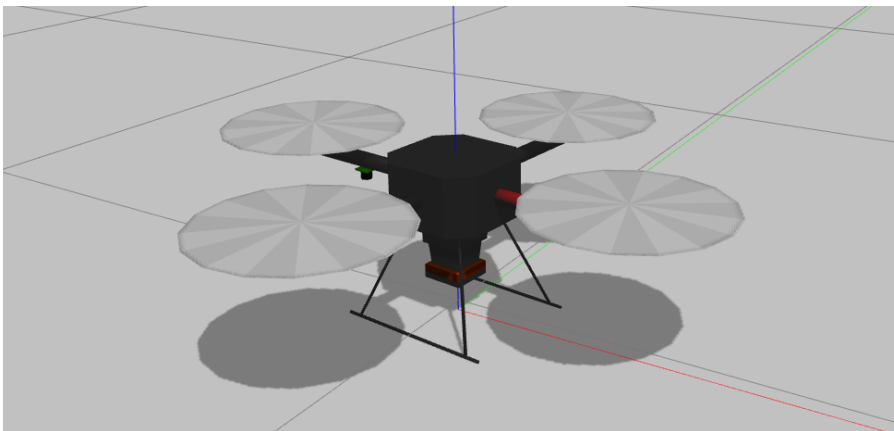


Figure 3.3: Quadrotor mesh-based model - Gazebo screenshot.

### 3.4 The Robot Operating System (ROS)

According to [QGS15] the Robot Operating System (ROS) is a loosely-defined framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

The authors in [JC18] refer to ROS as a meta-operating system, since it offers not only tools and libraries but even OS-like functions, such as hardware abstraction, package management, and a developer toolchain. Like a real operating system, ROS files are organized on the hard disk in a particular manner, as seen in Figure 3.4.

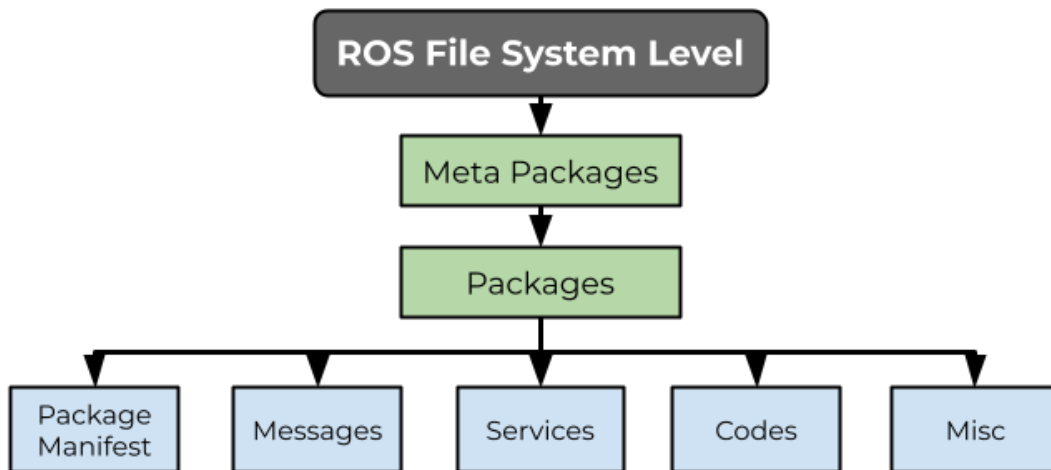


Figure 3.4: ROS file system level. Adapted from [JC18].

The authors briefly explain each block of the file system:

- **Packages:** The ROS packages are the most basic unit of the ROS software. They contain one or more ROS programs (nodes), libraries, configuration files, and so on, which are organized together as a single unit. Packages are the atomic build item and release item in the ROS software.
- **Package manifest:** The package manifest file is inside a package that contains information about the package, author, license, dependencies, compilation flags, and so on. The `package.xml` file inside the ROS package is the manifest file of that package.
- **Metapackages:** The term metapackage refers to one or more related packages which can be loosely grouped together. In principle, metapackages are virtual packages that don't contain any source code or typical files usually found in packages.

- **Metapackages manifest:** The metapackage manifest is similar to the package manifest, the difference being that it might include packages inside it as runtime dependencies and declare an `export` tag.
- **Messages:** The ROS messages are a type of information that is sent from one ROS process to the other. We can define a custom message inside the `msg` folder inside a package (`my_package/msg/MyMessageType.msg`). The extension of the message file is `.msg`.
- **Services:** The ROS service is a kind of request/reply interaction between processes. The reply and request data types can be defined inside the `srv` folder inside the package (`my_package/srv/MyServiceType.srv`).
- **Repositories:** Most of the ROS packages are maintained using a Version Control System (VCS), such as Git, Subversion (svn), Mercurial (hg), and so on. The collection of packages that share a common VCS can be called repositories. The package in the repositories can be released using a catkin release automation tool called bloom.

A process in ROS is called a node, which is responsible for performing computations and processing data collected from sensors. A ROS system is typically composed of several nodes (i.e. processes), where each node processes a certain data [Kou15].

The most basic architecture of a ROS system (see Figure 3.5) is composed of three nodes, which is where processes perform computations. ROS Master is aware of all existing nodes in the architecture and coordinates them accordingly. A Publisher has the ability to broadcast messages over a topic for other nodes to receive. A Subscriber is able to receive a message if a compatible topic is available [PG19].

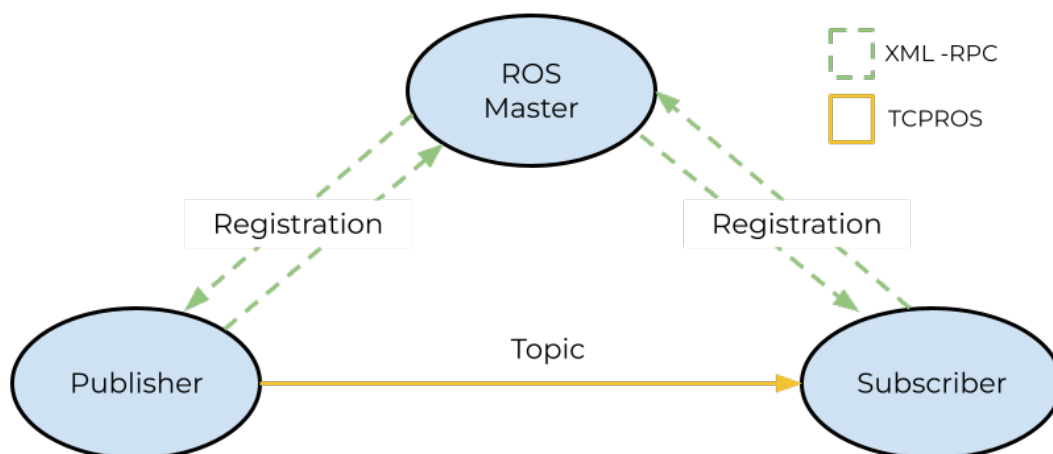


Figure 3.5: Basic ROS architecture. Adapted from [PG19].

Nodes communicate between each other via topics, which are composed of pre-formatted messages. They can be standard ones provided by ROS, off-the-shelf ones included in any given ROS package or custom ones designed by the user. There are two

communication protocols in place: to register new nodes in the system (XML-RPC) and to exchange data between them (TCPROS) [PG19].

### 3.4.1 Gazebo Simulator

Gazebo, proposed in [KH04], began as a venture in the University of Southern California. Later on, John Hsu (ROS maintainer) integrated it with the ROS framework. From then, the Open Source Robotics Foundation (OSRF) has maintained Gazebo along with ROS [NPRC17].

Gazebo is a 3D simulator that provides robots, sensors, environment models for 3D simulation required for robot development, and offers realistic simulation with its physics engine. It works as a stand-alone program, but there is also availability to establish a connection with Gazebo using a different type of Application Programmer Interfaces (APIs) and libraries. ROS is considered one of the most popular API that can be used to connect with Gazebo and which forms a powerful tool in the robotic field. In order to do that, ROS uses a set of packages named `gazebo_ros_pkgs` that provides wrappers around the stand-alone Gazebo and which can achieve integration with different ROS components. These packages provide the necessary interfaces to simulate a robot in Gazebo using ROS messages, topics, and services and give the ability to communicate with available sensor models included in Gazebo such as sonar, scanning laser range-finders and GPS [YB19]. Figure 3.6 shows a screenshot of the Gazebo environment running the Hector Quadrotor simulation.

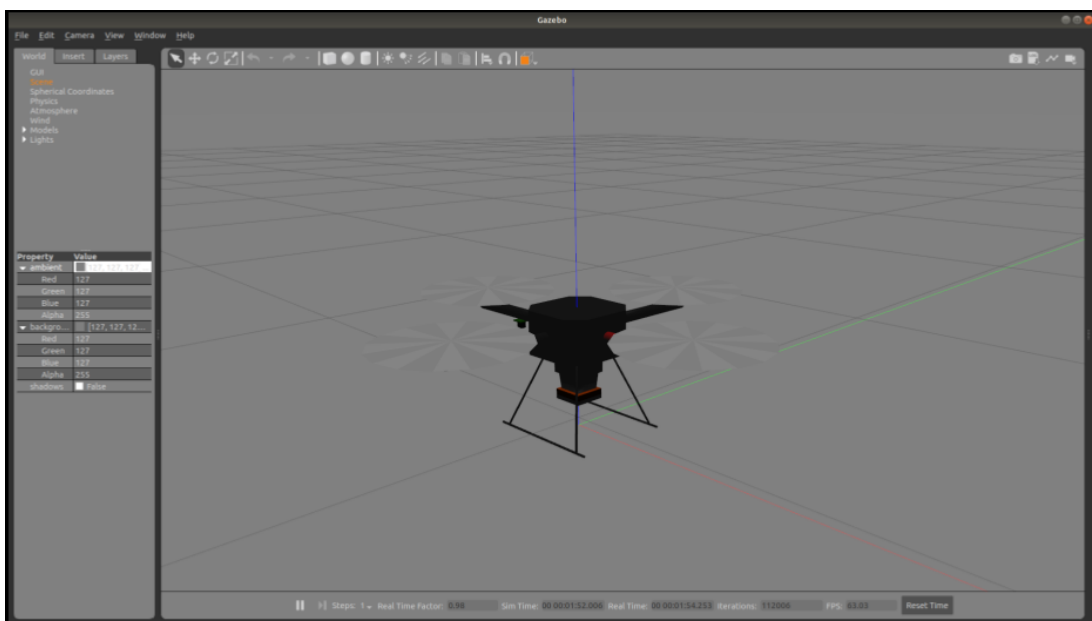


Figure 3.6: Gazebo simulator running Hector quadrotor.

### 3.5 Sensors

Despite its usually small size, quadrotors are equipped with an array of sensors. These sensors are typically used to determine the vehicle's states related to its position in the world frame, like absolute position and attitude. Other uses for the quadrotors' sensors can be highlighted, like obstacle detection and environment mapping. The most common sensors available for quadrotors include:

- **Global Positional System (GPS):** for absolute position measurement  $(x, y, z)$ ;
- **Gyroscope:** for angular rates measurement  $(g_x, g_y, g_z)$ ;
- **Accelerometer:** for accelerations measurement  $(a_x, a_y, a_z)$
- **Magnetometer:** for magnetic field measurement  $(m_x, m_y, m_z)$ , used for heading determination;
- **Sonar:** for the measurement of the relative height above the ground  $(h)$ ;
- **Barometer:** to measure the air speed or the altitude;
- **LIDAR (Light Detection And Ranging):** it can be used to measure height, the immediate surroundings, and mapping;
- **Visual Sensors:** the most versatile sensor can be used to determine absolute position, attitude, height, heading, environment mapping, and collision avoidance. Its drawback is the high computational cost it demands.

For this project, five sensors were chosen to determine the quadrotor's attitude, height, and  $xy$  position. An accelerometer, gyroscope, and magnetometer are used to determine the attitude, a sonar is used to determine the relative to the ground height, and a GPS (Global Positioning System) coupled with the accelerometer gives the position. These sensors were chosen due to their simplicity, information data size, and availability in the Hector-Quadrotor simulator.

### 3.6 Sensor Fusion

In this work, the author uses a state observer that is capable of performing the fusion of Inertial Measurement Unit (IMU), and magnetometers information, at the same time filtering the sensors' undesirable noises. With the use of an Extended Kalman Filter (EKF) based on quaternions, the attitude of the quadrotor is determined. The use of quaternions

for the description of the dynamic equations simplifies the filter equations and eliminates the need for trigonometric functions that significantly increases the computational cost.

One of the most known techniques for the state estimation of dynamic systems is the Kalman Filter (KF) [Kal60]. Explaining simplistically, the Kalman Filter gives a recursive method of state estimation of a dynamic system in the presence of noise. A vital aspect of this algorithm is that the Kalman Filter keeps estimation of the states vector ( $\bar{x}$ ) as well of the covariance matrix of the estimated error. It is possible to assert that the Kalman Filter is a Gaussian Probability Density Function (PDF) with mean ( $\bar{x}$ ) and covariance ( $P$ ) [CHL<sup>+</sup>05].

The algorithm of the Kalman Filter performs the state estimation in two distinct phases. The first phase is the prediction phase, where the mathematical model is used to predict the system's state in a sample  $k$ . In the second phase, the correction is performed, where the sensors' gathered information is used to correct the estimation done in the first phase. This method of estimation can be classified as a parametric Bayesian Filter, that is, the Kalman Filter parametrizes the system's and the sensors' uncertainties as Gaussian distributions with its means and covariances [TBF05].

### 3.6.1 Kalman Filter

To build a Kalman Filter (shown in Algorithm 3.1) is necessary for the system to be estimated to be linear, time-invariant, and effected by additive Gaussian noise. Consider the linear system in equation 3.1.

$$\begin{aligned}x_k &= A_k x_{k-1} + B_k u_k + \epsilon_k \\z_k &= C_k x_k + \delta_k\end{aligned}\tag{3.1}$$

The variable  $x_k \in \mathbb{R}^n$  represents the system's states,  $u_k$  represents the inputs. The variable  $z_k \in \mathbb{R}^p$  represents the measurements coming from the sensors and the matrices  $A_k$ ,  $B_k$  and  $C_k$  describe the linear system. The element  $\epsilon_k$  is a Gaussian signal of mean equal to zero and covariance  $R_k$  that describe the uncertainties of the model. The variable  $\delta_k$  represents the noise that affects the sensors, and this is also a Gaussian signal of zero mean and has a  $Q_k$  covariance.

Algorithm 3.1: KF Algorithm.

```

1 algorithm KF ( $x, u, Q, R, P$ ) :
2    $\bar{x}_k = A_k x_{k-1} + B_k u_k$ 
3    $\bar{P}_k = A_k P_{k-1} A_k' + R_k$ 
4    $K_k = \bar{P}_k C_k' (C_k \bar{P}_k C_k' + Q_k)^{-1}$ 
5    $x_k = \bar{x}_k + K_k (z_k - C_k (\bar{x}_k))$ 
6    $P_k = (I - K_k C_k) \bar{P}_k$ 
7 return ( $P_k, x_k$ )

```



Analyzing the algorithm is possible to conclude that in the equations 3.2 and 3.3 the state's estimation and its covariance  $\bar{P}_k$  are calculated *a priori*, meaning, prior to a new measurement.

$$\bar{x}_k = A_k x_{k-1} + B_k u_k \quad (3.2)$$

$$\bar{P}_k = A_k P_{k-1} A_k' + R_k \quad (3.3)$$

The term  $K_k$  represents the correction gain of the Kalman Filter and is calculated in equation 3.4.

$$K_k = \bar{P}_k C_k' (C_k \bar{P}_k C_k' + Q_k)^{-1} \quad (3.4)$$

The correction phase, the final estimation of the system's state  $x_k$  and its covariance  $P_k$  are calculated in the equations 3.5 and 3.6.

$$x_k = \bar{x}_k + K_k (z_k - C_k(\bar{x}_k)) \quad (3.5)$$

$$P_k = (I - K_k C_k) \bar{P}_k \quad (3.6)$$

### 3.6.2 Extended Kalman Filter

The assumptions of linear state transitions and linear measurements with added Gaussian noise are rarely fulfilled in practice. Linear next state transitions cannot describe most of the robotic vehicles. Then, simple Kalman filters do not apply to most trivial robotics problems. For this thesis, the Extended Kalman Filter (EKF, shown in Algorithm 3.2) is used instead, since it allows nonlinear equations.

The space-state description shown in equation 3.7 is now represented as,

$$\begin{aligned} x_k &= f(u_k, x_{k-1}) + \epsilon_k \\ z_k &= h(x_k) + \delta_k. \end{aligned} \quad (3.7)$$

To run the EKF is necessary to perform a linearization in equation 3.7. For this, the  $f$  function is approximated as a first-order Taylor series, shown in equation 3.8.

$$f(u_k, x_{k-1}) \approx f(u_k, \bar{x}_{k-1}) + \underbrace{\frac{\partial f(u_k, \bar{x}_{k-1})}{\partial \bar{x}_{k-1}}}_{:=F_k} (x_{k-1} - \bar{x}_{k-1}) \quad (3.8)$$

The  $F_k$  matrix is known as the jacobian matrix, and its values depend on  $u_k$  and  $\bar{x}_k$ . This dependence means that  $F_k$  is not constant. The same linearization is applied to the function related to the sensors, as shown in equation 3.9.

$$h(x_k) \approx h(\bar{x}_k) + \underbrace{\frac{\partial h(\bar{x}_k)}{\partial \bar{x}_k}}_{:=H_k} (x_k - \bar{x}_k) \quad (3.9)$$

### Algorithm 3.2: EKF Algorithm.

```

1 algorithm EKF( $x, u, Q, R, P$ ):
2    $\bar{x}_k = f(u_k, x_{k-1})$ 
3    $\bar{P}_k = F_k P_{k-1} F_k' + R_k$ 
4    $K_k = \bar{P}_k H_k' (H_k \bar{P}_k H_k' + Q_k)^{-1}$ 
5    $x_k = \bar{x}_k + K_k (z_k - h(\bar{x}_k))$ 
6    $P_k = (I - K_k H_k) \bar{P}_k$ 
7 return ( $P_k, x_k$ )

```

Comparing Algorithms 3.1 and 3.2 is possible to identify that both algorithms follow the same basic structure. Lines 1 and 2 are responsible for the state's prediction; line 3 calculates the Kalman gain, and lines 4 and 5 perform the correction phase.

### 3.6.3 Quaternion Representation

In the field of mathematical modeling of dynamical systems, quaternion representation is a valuable tool as they can represent rotation matrices in three dimensions. As [KNG05] stated, a rotation of  $\theta$  radians over an arbitrary unitary vector  $\mathbf{k} \in \mathbb{R}^3$  can be parametrized as shown in equation 3.11<sup>1</sup>.

$$R(\mathbf{q}) = I_3 + 2\eta S(\varepsilon) + 2S(\varepsilon)^2, \quad R(\mathbf{q}) \in \mathbb{R}^{3 \times 3} \quad (3.11)$$

In equation 3.11,  $I_3$  is the identity matrix of the third order,  $\eta = \cos(\theta/2)$ ,  $\varepsilon = \mathbf{k} \cdot \sin(\theta/2) \in \mathbb{R}^3$ , and  $\mathbf{q} \in \mathbb{R}^4$  is defined in equation 3.12.

<sup>1</sup>The operator  $S(\varepsilon)$  is the skew-symmetric matrix, represented as:

$$S(\varepsilon) = \begin{bmatrix} 0 & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & 0 & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & 0 \end{bmatrix} \therefore S(\varepsilon)\varepsilon = 0. \quad (3.10)$$

$$\mathbf{q} := \begin{bmatrix} \eta \\ \varepsilon \end{bmatrix} \quad (3.12)$$

This way, the  $\mathbf{q}$  vector represents the orientation of a rigid body in relation to the global coordinate system. Quaternion representation is not just more computationally efficient than Euler angles, but it also avoids kinematic singularities and discontinuities [Sze10]. Deriving the rotation matrix  $R(\mathbf{q})$  is obtained the dynamic equations that describe the system rotation in quaternions, shown in equations 3.13 and 3.14 [XM15].

$$\dot{\eta} = -\frac{1}{2}\varepsilon^T \omega \quad (3.13)$$

$$\dot{\varepsilon} = \frac{1}{2}[\eta I_3 + S(\varepsilon)]\omega \quad (3.14)$$

In equations 3.13 and 3.14,  $\omega \in \mathbb{R}^3$  is the angular velocity vector in the local coordinates. This vector is directly measured by the gyroscope attached to the quadrotor body.

For the sake of simplicity, we can represent the rotation matrix shown in equation 3.11 as

$$R(\mathbf{q}) = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \quad (3.15)$$

Then, to get back to the quaternion representation. It is used the algorithm proposed by [She78] which yields four possible results as seen in equations 3.16, 3.17, 3.18 and 3.19.

$$q_1 = \frac{1}{2} \begin{bmatrix} \sqrt{1 + R_{11} + R_{22} + R_{33}} \\ (R_{32} - R_{23})/(\sqrt{1 + R_{11} + R_{22} + R_{33}}) \\ (R_{13} - R_{31})/(\sqrt{1 + R_{11} + R_{22} + R_{33}}) \\ (R_{21} - R_{12})/(\sqrt{1 + R_{11} + R_{22} + R_{33}}) \end{bmatrix} \quad (3.16)$$

$$q_2 = \frac{1}{2} \begin{bmatrix} (R_{32} - R_{23})/(\sqrt{1 + R_{11} - R_{22} - R_{33}}) \\ \sqrt{1 + R_{11} - R_{22} - R_{33}} \\ (R_{12} + R_{21})/(\sqrt{1 + R_{11} - R_{22} - R_{33}}) \\ (R_{31} + R_{13})/(\sqrt{1 + R_{11} - R_{22} - R_{33}}) \end{bmatrix} \quad (3.17)$$

$$q_3 = \frac{1}{2} \begin{bmatrix} (R_{13} - R_{31})/(\sqrt{1 - R_{11} + R_{22} - R_{33}}) \\ (R_{12} + R_{21})/(\sqrt{1 - R_{11} + R_{22} - R_{33}}) \\ \sqrt{1 - R_{11} + R_{22} - R_{33}} \\ (R_{23} + R_{32})/(\sqrt{1 - R_{11} + R_{22} - R_{33}}) \end{bmatrix} \quad (3.18)$$

$$q_4 = \frac{1}{2} \begin{bmatrix} (R_{21} - R_{12})/(\sqrt{1 - R_{11} - R_{22} + R_{33}}) \\ (R_{31} + R_{13})/(\sqrt{1 - R_{11} - R_{22} + R_{33}}) \\ (R_{32} + R_{23})/(\sqrt{1 - R_{11} - R_{22} + R_{33}}) \\ \sqrt{1 - R_{11} - R_{22} + R_{33}} \end{bmatrix} \quad (3.19)$$

To obtain the best possible result, avoid division by zero or a minimal number, or perform the square root of a negative number. The whichever position on vector  $v = [(R_{11} + R_{22} + R_{33}), R_{11}, R_{22}, R_{33}]$  has the largest number that correspond to which of the  $q_i$  is chosen. For example: if  $v[1] > v[2]$  and  $v[1] > v[3]$  and  $v[1] > v[4]$ , then the better result comes from  $q_1$ .

### Quaternion to Euler Angles Conversion

The PID controller, in this case, calculates each control law based on the Euler angles of the current quadrotor state. Since the EKF gives the system an estimation of the attitude based on quaternions, the PID must convert the quaternion into the Euler angles representation. The reason behind this is two-fold: first of all, using Euler angles simplifies the construction of the control laws of Equation 5.57, and second, since the system is based on fixed-point arithmetic, the range of possible numbers between 0 and 1 becomes restricted, making the controller less efficient due to the loss of information. The equations 3.20, 3.21 and 3.22, shows the calculation for the euler angles for the x, y and z respectively.

$$ea_x = asin\left(\frac{-2(q_x q_z - q_y q_w)}{q_w^2 + q_x^2 + q_y^2 + q_z^2}\right) \quad (3.20)$$

$$ea_y = atan2(2(q_y q_z + q_x q_w), -(q_x^2 - q_y^2 + q_z^2 + q_w^2)) \quad (3.21)$$

$$ea_z = atan2(2(q_x q_y + q_z q_w), -(q_x^2 - q_y^2 - q_z^2 + q_w^2)) \quad (3.22)$$

## 3.7 State Space Representation

In Section 3.2 we gave an overview of the quadrotor dynamics. This section gives a mathematical representation of these dynamics. In a 6DOF quadrotor system, we can define the state vector as a  $X \in \mathbb{R}^{12 \times 1}$  matrix, as seen in equation 3.23.

$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} \end{bmatrix}^T \quad (3.23)$$

The representation of equation 3.23 is mapped to the degrees of freedom of the quadrotor as shown in equation 3.24.

$$X = [\phi \ \dot{\phi} \ \theta \ \dot{\theta} \ \psi \ \dot{\psi} \ z \ \dot{z} \ x \ \dot{x} \ y \ \dot{y}]^T \quad (3.24)$$

Where the vector  $\omega = [\phi \ \theta \ \psi]^T$  represents the attitude states: roll, pitch and yaw respectively. Vector  $\dot{\omega} = [\dot{\phi} \ \dot{\theta} \ \dot{\psi}]^T$  represents the rate of change of the attitude vector. The vector  $p = [x \ y \ z]^T$  represents the 3D global position states: x position, y position and height (z position). Vector  $\dot{p} = [\dot{x} \ \dot{y} \ \dot{z}]^T$  represents the rate of change of the position vector.

The control input vector of the system  $U$  consists of four inputs,  $U_1$ ,  $U_2$ ,  $U_3$  and  $U_4$ . And is arranged as shown in the matrix of equation 3.25.

$$U = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ b(-\Omega_2^2 + \Omega_4^2) \\ b(\Omega_1^2 + \Omega_3^2) \\ d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{bmatrix} \quad (3.25)$$

$U_1$  is the resulting upwards force of the four rotors responsible for the altitude of the quadrotor and its rate of change ( $z, \dot{z}$ ).  $U_2$  is the difference in thrust between rotors 2 and 4 responsible for the roll rotation and its rate of change ( $\phi, \dot{\phi}$ ).  $U_3$  represents the difference in thrust between rotors 1 and 3, generating the pitch rotation and its rate of change ( $\theta, \dot{\theta}$ ).  $U_4$  is the difference in torque between the two clockwise turning rotors and the two counterclockwise turning rotors generating the yaw rotation and its rate of change ( $\psi, \dot{\psi}$ ). The rotors (1 through 4) velocities are represented by  $\Omega_1, \Omega_2, \Omega_3$  and  $\Omega_4$ .

Two subsystems describe the nonlinear dynamics: rotational dynamics and translational dynamics. The equations 3.26, 3.27 and 3.28 represent the the angular accelerations of the system.

$$\ddot{\phi} = \dot{\theta}\dot{\psi}\left(\frac{I_{yy} - I_{zz}}{I_{xx}}\right) + \dot{\theta}\left(\frac{J}{I_{xx}}\right)\Omega_r + \frac{L_a}{I_{xx}}U_2 \quad (3.26)$$

$$\ddot{\theta} = \dot{\phi}\dot{\psi}\left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right) - \dot{\phi}\left(\frac{J}{I_{yy}}\right)\Omega_r + \frac{L_a}{I_{yy}}U_3 \quad (3.27)$$

$$\ddot{\psi} = \dot{\theta}\dot{\phi}\left(\frac{I_{xx} - I_{yy}}{I_{zz}}\right) + \frac{1}{I_{zz}}U_4 \quad (3.28)$$

The equations 3.29, 3.30 and 3.31 represent the linear accelerations of the system.

$$\ddot{z} = g - (\cos(\phi)\cos(\theta)) + \frac{1}{m}U_1 \quad (3.29)$$

$$\ddot{x} = (\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi))\frac{1}{m}U_1 \quad (3.30)$$

$$\ddot{y} = (\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi))\frac{1}{m}U_1 \quad (3.31)$$

Table 3.1 presents the physical parameters of the quadrotor system and its units, that are represented in the equations 3.26, 3.27, 3.28, 3.29, 3.30 and 3.31.

Table 3.1: Physical parameters of the quadrotor system.

Variable	Physical parameters of the quadrotor system
$I_{xx}$	Moment of Inertia (X axis) - ( $kg * m^2$ )
$I_{yy}$	Moment of Inertia (Y axis) - ( $kg * m^2$ )
$I_{zz}$	Moment of Inertia (Z axis) - ( $kg * m^2$ )
$L_a$	Distance between rotor and center of mass - ( $m$ )
$d$	Drag constant
$b$	Lift constant
$g$	Gravity - ( $m/s^2$ )
$m$	Mass of the Quadrotor - ( $kg$ )
$\Omega_r$	Overall residual angular velocity of the four motors (rad/s) ( $-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4$ )
$J$	Moment of inertia of the rotor about its axis of rotation - ( $kg * m^2$ )

Using the equations 3.26, 3.27 and 3.28 of the rotational angular acceleration. And those of translation, equations 3.29, 3.30 and 3.31, the complete mathematical model of the quadrotor can be written in a state space representation in equation 3.32:

$$\dot{X} = f(X, U) = \begin{cases} \dot{x}_1 = \dot{\phi} = & x_2 \\ \dot{x}_2 = \ddot{\phi} = & x_4 x_6 \left( \frac{I_{yy} - I_{zz}}{I_{xx}} \right) + x_4 \left( \frac{J}{I_{xx}} \right) \Omega_r + \frac{L_a}{I_{xx}} U_2 \\ \dot{x}_3 = \dot{\theta} = & x_4 \\ \dot{x}_4 = \ddot{\theta} = & x_2 x_6 \left( \frac{I_{zz} - I_{xx}}{I_{yy}} \right) - x_2 \left( \frac{J}{I_{yy}} \right) \Omega_r + \frac{L_a}{I_{yy}} U_3 \\ \dot{x}_5 = \dot{\psi} = & x_6 \\ \dot{x}_6 = \ddot{\psi} = & x_2 x_4 \left( \frac{I_{xx} - I_{yy}}{I_{zz}} \right) + \frac{1}{I_{zz}} U_4 \\ \dot{x}_7 = \dot{z} = & x_8 \\ \dot{x}_8 = \ddot{z} = & g - \frac{U_1}{m} (\cos(x_1) \cos(x_3)) \\ \dot{x}_9 = \dot{x} = & x_{10} \\ \dot{x}_{10} = \ddot{x} = & -\frac{U_1}{m} ((\sin(x_1) \sin(x_5)) + \cos(x_1) \sin(x_3) \cos(x_5)) \\ \dot{x}_{11} = \dot{y} = & x_{12} \\ \dot{x}_{12} = \ddot{y} = & \frac{U_1}{m} ((\sin(x_1) \cos(x_5)) + \cos(x_1) \sin(x_3) \sin(x_5)) \end{cases} \quad (3.32)$$

Euler angles and quaternions were considered in this thesis because in the EKF algorithm, quaternions are used to simplify the calculations, and Euler angles are used in

the PID controller for better control of the quadrotor. Quaternions vary in values from 0 to 1, while the Euler angle goes from 0 to 360 degrees.

### 3.7.1 Linearization

In order to transform the nonlinear movement equations representation presented in equation 3.32 into the space state model shown in 3.33, it is necessary to linearize the system.

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \quad (3.33)$$

Breaking down the equation 3.33, we can assert that  $x$  is a vector containing all the systems states where,  $x \in \mathbb{R}^n$ ,  $y$  is called the output vector where,  $y \in \mathbb{R}^q$  and  $u$  is the input vector where,  $u \in \mathbb{R}^p$ . Regarding the matrices,  $A$  is the state matrix where,  $A \in \mathbb{R}^{n \times n}$ ,  $B$  is the input matrix where,  $B \in \mathbb{R}^{n \times p}$ ,  $C$  is the output matrix where,  $C \in \mathbb{R}^{q \times n}$  and  $D$  is the feedforward matrix where,  $D \in \mathbb{R}^{q \times p}$ .

The linearization of a nonlinear set of dynamics  $\dot{x} = f(x, u)$  (see. eq 3.32) requires the determination of an equilibrium point. Which is a point where the system starts, and it remains there for all future time, e.g., a stable operating point. In the case of the quadrotor, it is the stationary hover point above the ground. This way, the state system vector assumes the form expressed in equation 3.34

$$\text{Hover Condition (Equilibrium Point): } \begin{cases} \theta = \phi = \psi = \dot{\theta} = \dot{\phi} = \dot{\psi} = \ddot{\theta} = \ddot{\phi} = \ddot{\psi} = 0 \\ \dot{x} = \dot{y} = \dot{z} = \ddot{x} = \ddot{y} = \ddot{z} = 0 \end{cases}, \quad U = \begin{bmatrix} mg \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.34)$$

Setting the equilibrium points  $x_e$  and  $u_e$ , we can rewrite  $x$  and  $u$  around the operating points:

$$\begin{cases} x = x_e + \delta x \\ u = u_e + \delta u \end{cases} \quad (3.35)$$

Where  $\delta$  denotes minor variations about the operation points. We can develop the linearized equations by using Taylor Series Expansion on  $f(\cdot, \cdot)$ , about  $x_e$  and  $u_e$ :

$$\frac{d}{dt}(x_e + \delta x) = f(x_e + \delta x, u_e + \delta u) \approx f(x_e, u_e) + \left. \frac{\partial f}{\partial x} \right|_0 \delta x + \left. \frac{\partial f}{\partial u} \right|_0 \delta u \quad (3.36)$$

Combining all  $n$  state equations we have:

$$\begin{aligned} \frac{d}{dt} \delta x &= \begin{bmatrix} \left. \frac{\partial f_1}{\partial x} \right|_0 \\ \left. \frac{\partial f_2}{\partial x} \right|_0 \\ \vdots \\ \left. \frac{\partial f_n}{\partial x} \right|_0 \end{bmatrix} \delta x + \begin{bmatrix} \left. \frac{\partial f_1}{\partial u} \right|_0 \\ \left. \frac{\partial f_2}{\partial u} \right|_0 \\ \vdots \\ \left. \frac{\partial f_n}{\partial u} \right|_0 \end{bmatrix} \delta u \\ &= A \delta x + B \delta u \end{aligned} \quad (3.37)$$

Where,

$$A = \begin{bmatrix} \left. \frac{\partial f_1}{\partial x_1} \right|_{x=x^0} & \left. \frac{\partial f_1}{\partial x_2} \right|_{x=x^0} & \cdots & \left. \frac{\partial f_1}{\partial x_{12}} \right|_{x=x^0} \\ \left. \frac{\partial f_2}{\partial x_1} \right|_{x=x^0} & \left. \frac{\partial f_2}{\partial x_2} \right|_{x=x^0} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \left. \frac{\partial f_{12}}{\partial x_1} \right|_{x=x^0} & \cdots & \cdots & \left. \frac{\partial f_{12}}{\partial x_{12}} \right|_{x=x^0} \end{bmatrix}, \quad B = \begin{bmatrix} \left. \frac{\partial f_1}{\partial u_1} \right|_{u=u^0} & \left. \frac{\partial f_1}{\partial u_2} \right|_{u=u^0} & \cdots & \left. \frac{\partial f_1}{\partial u_{12}} \right|_{u=u^0} \\ \left. \frac{\partial f_2}{\partial u_1} \right|_{u=u^0} & \left. \frac{\partial f_2}{\partial u_2} \right|_{u=u^0} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \left. \frac{\partial f_{12}}{\partial u_1} \right|_{u=u^0} & \cdots & \cdots & \left. \frac{\partial f_{12}}{\partial u_4} \right|_{u=u^0} \end{bmatrix} \quad (3.38)$$

Similarly, we can apply this expansion to the nonlinear measurement equations  $y = g(x, u)$  and  $y = y_e + \delta y$ :

$$\begin{aligned} \frac{d}{dt} \delta y &= \begin{bmatrix} \left. \frac{\partial g_1}{\partial x} \right|_0 \\ \left. \frac{\partial g_2}{\partial x} \right|_0 \\ \vdots \\ \left. \frac{\partial g_n}{\partial x} \right|_0 \end{bmatrix} \delta x + \begin{bmatrix} \left. \frac{\partial g_1}{\partial u} \right|_0 \\ \left. \frac{\partial g_2}{\partial u} \right|_0 \\ \vdots \\ \left. \frac{\partial g_n}{\partial u} \right|_0 \end{bmatrix} \delta u \\ &= C \delta x + D \delta u \end{aligned} \quad (3.39)$$

Finally, we can find the matrices  $A$ ,  $B$ ,  $C$  and  $D$  for the system in equation 3.32, for the equilibrium points in 3.34:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \end{bmatrix} \quad (3.40)$$



$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.41)$$

### 3.8 Most Commonly Used Controllers in Quadrotors

Today's principal control methodologies in Quadrotors designs can be separated into three distinct categories: linear controllers, nonlinear controllers, and learning-based controllers. Figure 3.7 shows the main control schemes found in a preliminary research for this project. Within these categories, this work shows the most commonly used control techniques for quadrotors in the literature.

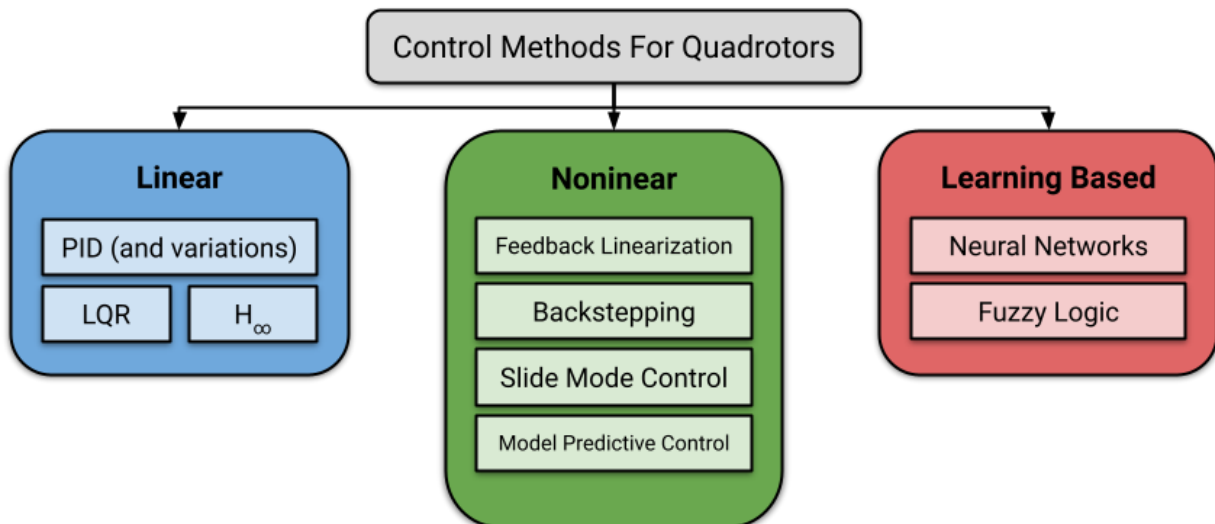


Figure 3.7: Diagram showing with the most commonly used controllers by quadrotors in the literature.

To arrive at this list of quadrotor control methods, we used the following studies: Nguyen et al. [NQN<sup>+</sup>20], Mo and Farid [MF19], Nascimento and Saska [NS19], Kim et al. [KGW19], Shraim et al. [SAY18] and Li et al. [LSJ15]. Having compiled the list of surveys, we selected the control methods more likely to be used in the quadrotor context. The itemization below briefly explains all of this list, giving summarized descriptions and concepts of all the controllers. The later section explains why the final three controllers were selected for the thesis study case, giving a more in-depth explanation of these selected controllers.

### 3.8.1 Linear Controllers Used by Quadrotors

As shown in Figure 3.7, the most common linear control methods for the control of quadrotors are:

- **Proportional-Integral-Derivative (PID) control:** Proportional-Integral-Derivative Controllers (PID) are one of the simplest classical controllers seen in many textbook applications. A PID controller is a control loop feedback mechanism that directly adjusts control values with a closed-form formula based on derivative, integral, and proportional gains [Xia16]. The classical PID linear controller has the advantage that parameter gains are easy to adjust, are simple to design, and have good robustness.
- **Linear Quadratic Regulator (LQR) control:** The LQR control algorithm is an optimal control that operates a dynamic system at minimum cost [KGD14]. A quadratic function defines a linear differential equation representing the system's cost. This cost function is minimized to provide the best control signal. The function is formulated like:  $J = \int_{t_0}^{\infty} (x^T Q x + u^T R u) dt$  and LQR control is determined by the Q and R metrics. Selection of appropriate Q and R values obtains the best feedback gain values for the plant system [DP+17].
- **H-Infinity ( $H_{\infty}$ ) control:** Robust  $H_{\infty}$  control is an essential branch of control theory. A robust  $H_{\infty}$  control problem for systems with parameter uncertainty can be stated as follows: given a dynamic system with exogenous input and measured output, where the goal is to design a control law such that the  $L_2$  gain<sup>2</sup> of the mapping from the exogenous input to the regulated output is minimized or no larger than some prescribed level for all admissible uncertainties [Cha14].

### 3.8.2 Nonlinear Controllers Used by Quadrotors

As shown in Figure 3.7, the most common nonlinear control methods for the control of quadrotors are:

- **Feedback Linearization Control:** The central idea of Feedback Linearization is to algebraically transform a nonlinear system dynamics into a (fully or partly) linear one so that linear control techniques can be applied. This differs entirely from conventional linearization in that feedback linearization is achieved by exact state transformations and feedback rather than by linear approximations of the dynamics. In its simplest form,

---

<sup>2</sup> $L_2$  gain of an input-output system quantifies the maximal gain in "energy transmission" from input to output, where "energy" is of a signal  $g$  is understood as the integral of  $|g(t)|^2$  over a time interval [Meg06].

feedback linearization amounts to canceling the nonlinearities in a nonlinear system so that the closed-loop dynamics are in a linear form [SL<sup>+</sup>91].

- **Backstepping Control:** The common idea of the Backstepping procedure is to apply a passivation design to a small part of the system and then to reapply it step-by-step by augmenting the sub-system at each step. Backstepping employs an analytic expression for the time-derivative of the control law designed at the preceding step. Backstepping starts with the system equation (integrator), which is the farthest from the control input and reaches the control input at the last step [SJK12].
- **Slide Mode Control:** Sliding mode control is a particular type of Variable Structure System (VSS) characterized by some feedback control laws and a decision rule. The decision rule, termed the switching function, has as inputs some measure of the current system behavior and produces the particular feedback controller that should be used at that instant in time. In sliding mode control, Variable Structure Control Systems (VSCS) are designed to drive and then constrain the system state to lie within a neighborhood of the switching function [Spu14].
- **Model Predictive Control (MPC):** In Model Predictive Control (MPC), the control action is obtained by solving a finite horizon open-loop optimal control problem at each sampling instant. Each optimization yields a sequence of optimal control moves. However, only the first move is applied to the process: At the next time step, the computation is repeated over a shifted time horizon by taking the most recently available state information as the new initial condition of the optimal control problem. For this reason, MPC is also called receding or rolling horizon control [AB09].

### 3.8.3 Learning-Based Controllers Used by Quadrotors

As shown in Figure 3.7, the most common learning-based control methods for the control of quadrotors are:

- **Neural Networks:** The feedback and feedforward controllers and the prefilter can all be implemented as multilayered neural networks. The learning process gradually tunes the neural network's weights so that the error signal between the desired and actual plant responses is minimized. Since the error signal is the input to the feedback controller, the network's training leads to a gradual switching from feedback to feedforward action as the error signal becomes small [PSY88].
- **Fuzzy Logic:** The primary thrust of this control paradigm is to utilize the human control operator's knowledge and experience to intuitively construct controllers so that the

resulting controllers can emulate human control behavior to a certain extent [Yin00]. The fuzzy controller uses the control error and the rate of change in the control error as its inputs. A typical fuzzy logic control architecture has a standard additive mode; hence, it comprises two primary components: a decision-making logic module, also referred to as an inference engine, and a defuzzifier. The inference engine processes the inputs using a knowledge base. The outputs of the inference engine are converted into crisp values, defuzzified, by a defuzzifier [Zak03].

### 3.9 Chosen Control Methods and Quadrotor Implementation

Based on the list compiled in Section 3.8 we chose three control methods as study case for the quadrotor. The criteria for the choice of these controllers is the increasing complexity of the algorithms. One of the goals of this thesis is to evaluate the trade-off between computational cost and control performance in the proposed software framework with the selected controllers. Simpler algorithms such as the PID require a low demand in computing power. However, they may not deliver a satisfactory performance as a more demanding algorithm such as the MPC.

#### 3.9.1 Proportional Integral Derivative Control - PID

Proportional-Integral-Derivative Controllers (PID) are one of the simplest classical controllers seen in many textbook applications. A PID controller is a control loop feedback mechanism that directly adjust control values with a closed-form formula based on derivative, integral, and proportional gains [Xia16]. The classical PID linear controller has the advantage that parameter gains are easy to adjust, is simple to design and has good robustness. However some of the major challenges with the quadrotor include the non-linearity associated with the mathematical model and the imprecise nature of the model due to unmodeled or inaccurate mathematical modeling of some of the dynamics [ZJ14].

$$u_c(t) = k_p e(t) + K_i \int e(\tau) d\tau + k_d \frac{de}{dt} \quad (3.42)$$

In the PID equation (3.42),  $k_p$  is the proportional gain,  $k_i$  is the integral gain,  $k_d$  is the derivative gain, and the controller operates on the measured reference error time signal,  $e(t) = (sig_{reference} - sig_{measured})$ . The proportional component of the controller only depends on the difference between the goal set for the system's response and the actual system's response. The proportional gain  $K_p$  determines the response rate of the system's error signal, which means when  $K_p$  is raised, the system respond faster to the error signal trying

to get to  $e(t) = 0$ . But a large increase of  $K_p$  usually results in oscillation in the system's output. If  $K_p$  keeps increasing past the oscillatory state, the system becomes unstable.

The integral term is responsible for the sum of the error signal ( $I = \sum e(t)$ ) as time passes. Even a small error value makes the integral component to rise, not rising only when the error is equal to zero. That behavior forces the steady-state error to converge to zero. The derivative term monitors the change rate of the error signal ( $D = (e(t) - e(t-1)))$  causing a dampening on the effects of the proportional gain, reducing overshoot and oscillations. Figure 3.8 shows a block diagram of a PID controller in a feedback loop.

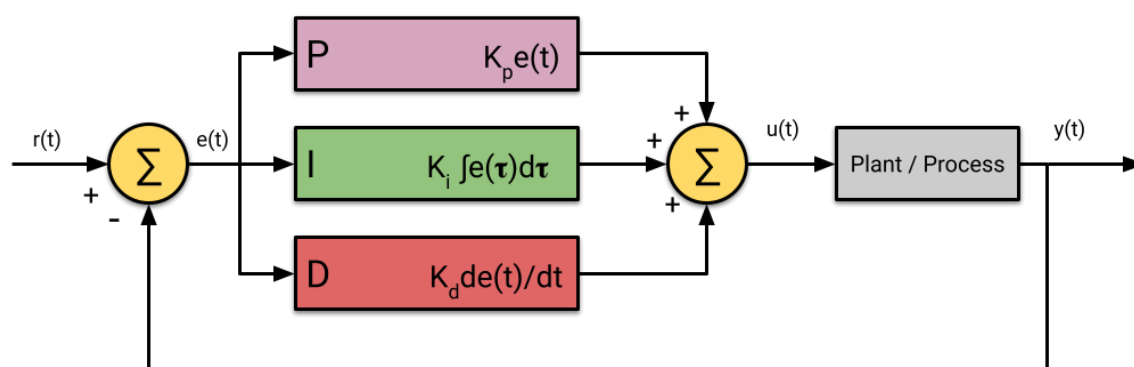


Figure 3.8: Block diagram of a PID controller in a feedback loop.

### Integral windup

The presence of nonlinear effects limits the PID performance and also causes the well-known phenomenon of integrator windup. All physical systems are subject to actuator saturation since all actuators have limitations. Actuator limits create a nonlinear effect that can be expressed by a saturation term, and this nonlinearity is one of the significant reasons for integral windup [MGEY06].

If an actuator that realizes the control action has an effective range limit, then the integrator may saturate and future correction is ignored until the saturation is offset. This cause slow-frequency oscillations and may lead to instability. A usual measure taken to counteract this effect is *anti-windup* [ACL05]. There are several anti-windup techniques in the literature, but for this project was chosen the Back-Calculation method to mitigate the effects of the phenomenon of integrator windup.

In Back-Calculation, when the controller output goes beyond the saturated value of the system, the additional feedback measures the difference between saturate control signals,  $U_{actuator}$ , and unsaturated control signal,  $U_{pid}$ . Then, the difference is fed back to the integrator to reset the integral term. This recalculation process occurs continuously until the value of the integral term gives a controller signal at the saturation limits [JTRH14].

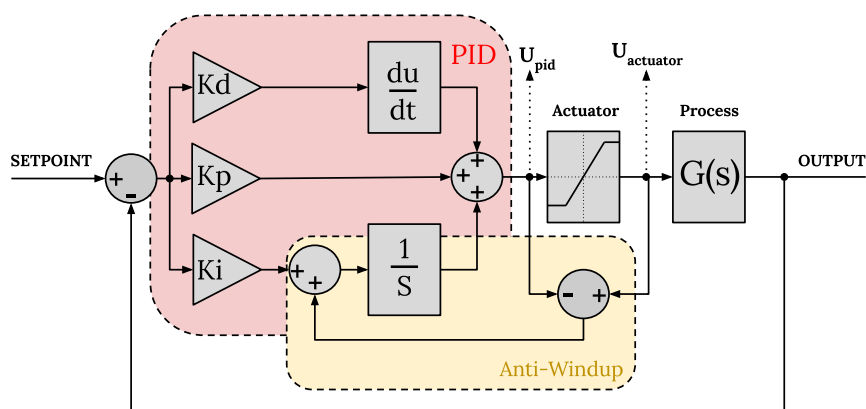


Figure 3.9: PID with Back-Calculation Anti-Windup Block Diagram.

### PID Controller Automatic Tuning

One of the features proposed in this thesis is the ability that a decentralized processing system could create a controller that would be able to control another control process. In this case, the author takes the PID process responsible for the attitude and height control and change its gains in runtime. This sub-controller is based on a fuzzy controller algorithm. The idea and algorithm behind this sub-controller is presented in this section.

### Fuzzy Logic Theory

The theory of fuzzy sets and Fuzzy Logic were developed in 1965 by engineer Lotfi A. Zadeh in [Zad65] and UC Berkley published his findings. This theory provides a mathematical basis for treating imprecise or vague information. Fuzzy systems are capable of imparting the knowledge of human specialists in problems regarding classification, modeling, or control systems [Lei09]. Fuzzy Logic (FL) is a multi-valued logic that allows intermediate values to be defined between conventional evaluations like true/false, yes/no, high/low, and more. Notions like rather tall or very fast can be formulated mathematically and processed by computers to apply a more human-like way of thinking in the programming of computers [Hel01].

According to [KK12], a fuzzy system has four main parts. First a fuzzification interface that modifies and converts inputs into suitable linguistic values so that they can be compared to the rules in the rule base. A rule base holds the knowledge in the form of rules on how best to control the system. An inference mechanism that evaluates which control rules are relevant at the current time and then decides what the input to the plant should be. And finally, a defuzzification interface converts the conclusions reached by the inference mechanism into crisp ones. Figure 3.10 shows a diagram example of a fuzzy system, followed by a brief explanation of each element.

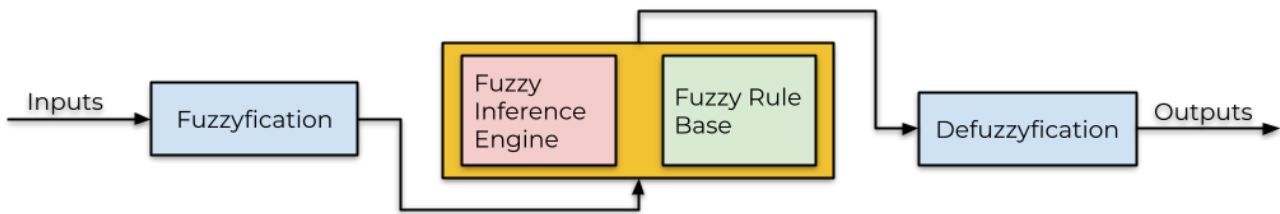


Figure 3.10: Fuzzy system diagram. Adapted from [Ian12].

## Fuzzification

Fuzzification measures the values of inputs variables; it performs scale mappings that transfer the range of values of inputs variables into corresponding universes of discourse. It serves the function of fuzzification that converts input data into suitable linguistic values, which may be viewed as the label of fuzzy sets.

## Fuzzy Inference Engine

The fuzzy inference engine is the kernel of a fuzzy logic system; it can simulate human decision-making based on fuzzy concepts. And is capable of inferring fuzzy actions employing implications and the rules of inference in fuzzy logic. A simple representation of this concept is shown in Figure 3.11.

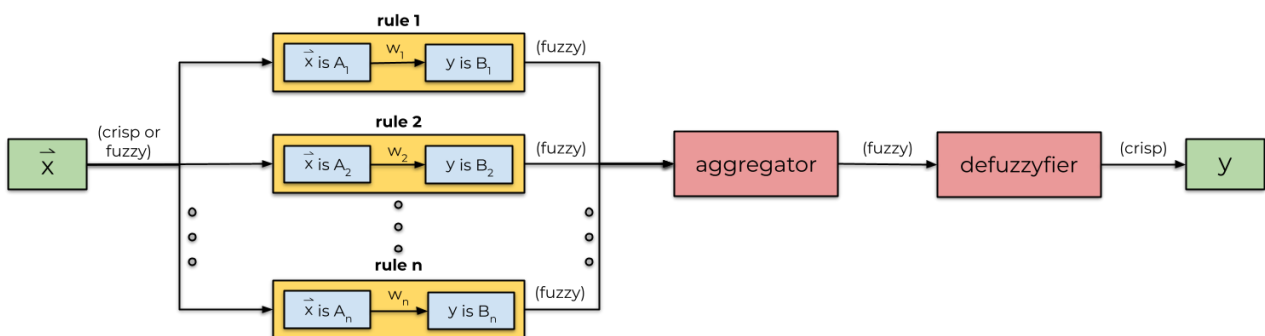


Figure 3.11: Fuzzy system inference engine. Adapted from [dRdS21].

## Fuzzy Rule Base

The fuzzy rule base comprises knowledge of the application domain. It consists of a "database" and a "linguistic (fuzzy) control rule base". It provides necessary definitions used to define linguistic rules and fuzzy data manipulation. Then characterizes the goals and the policies of the domain experts using a set of linguistic rules. (See Table 3.2)

Table 3.2: Example of a fuzzy system rule table.

Rules					
$R_1$	if $input_1$ is $LV_x$	$\cap$	if $input_2$ is $LV_g$	then	output is $LV_a$
$R_2$	if $input_1$ is $LV_y$	$\cap$	if $input_2$ is $LV_h$	then	output is $LV_b$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$R_n$	if $input_1$ is $LV_z$	$\cap$	if $input_2$ is $LV_i$	then	output is $LV_c$

## Defuzzification

Defuzzification performs scale mapping: which converts the range of values of output variables into corresponding universes of discourse. Then performs, defuzzification itself: which yields a non-fuzzy control action from an inferred fuzzy control action.

### Center of Gravity Defuzzification

In this thesis, from many possibilities, we chose the center of gravity (COG) method for defuzzification. The main idea of the COG method is to find the point  $\bar{x}$  where a vertical line would slice the fuzzy set aggregate into two equal masses, as presented in Figure 3.12.

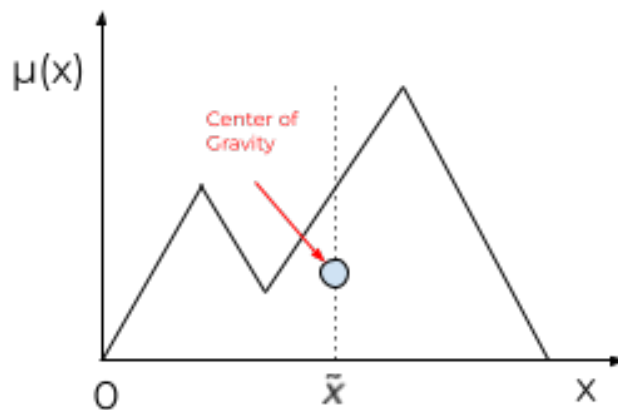


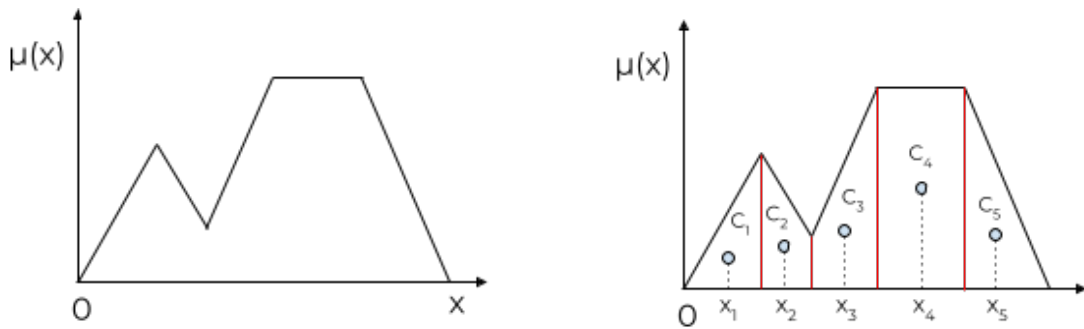
Figure 3.12: Representation of a defuzzification using COG method.

The area of the membership function distribution used to describe the combined control action is divided into several sub-areas. The areas and centers of gravity of each region are calculated. Then the sum of all these regions is used to determine the defuzzified value for a discrete fuzzy set. (See fig. 3.13)

According to [RP01], this method is prevalent. It is often used as a standard defuzzification method in experimental and industrial controllers. The COG generates a value that is the center of gravity of a fuzzy set. It minimizes the membership graded weighted mean of the square of the distance and can be calculated as:

$$\bar{x} = \frac{\sum_{i=1}^n A_i x_i}{\sum_{i=1}^n A_i} \quad (3.43)$$



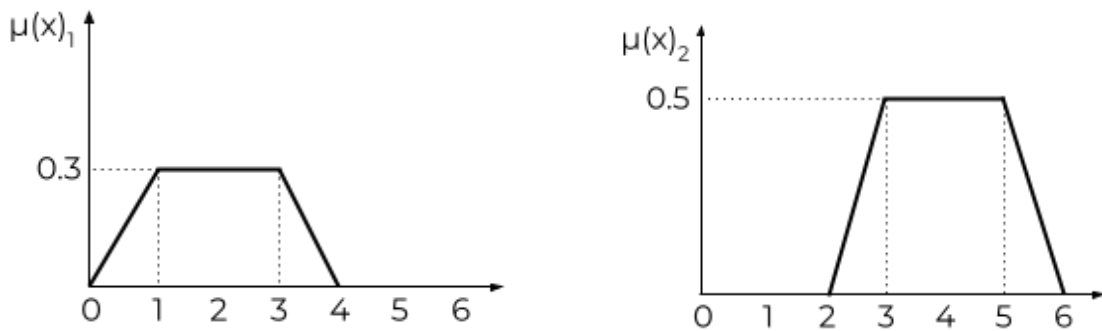


(a) Aggregated fuzzy set output without division.

(b) Aggregated fuzzy set output with division.

Figure 3.13: Center of gravity method - fuzzy set aggregation.

where  $A_i = \int \mu(x)dx$  and  $n$  is the number of geometrical components.



(a) Example of a output fuzzy set (1).

(b) Example of a output fuzzy set (2).

Figure 3.14: Example of two output fuzzy sets.

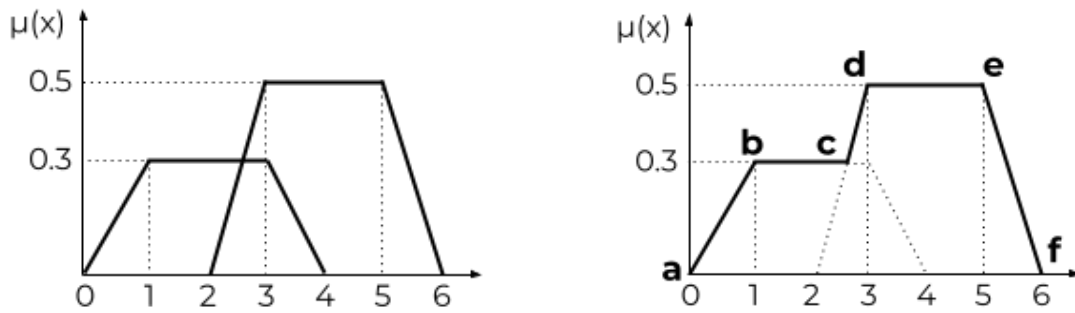
Figure 3.14 shows an example of two output fuzzy sets. To calculate the output value, first we need to aggregate the separate fuzzy sets, by putting them in the same axis, as seen in Figure 3.15.

With the aggregation of the fuzzy sets, the next step is to determine the function that represents the curve  $\mu(x)$ . The resulting function of curve  $a, b, c, d, e$  seen in Figure 3.15b, can be expressed as:

$$\mu(x) = \begin{cases} 0.3x, & 0 \leq x < 1 \\ 0.3, & 1 \leq x < 2.5 \\ 0.4x + 0.18, & 2.5 \leq x < 3 \\ 0.5, & 3 \leq x < 5 \\ -0.2x + 1.5, & 5 \leq x \leq 6 \end{cases} \quad (3.44)$$

And the crisp value of this fuzzy set is given by  $x = \frac{\int \mu_c(x).xdx}{\int \mu_c(x)dx}$ .

According to [JS20], fuzzy sets can be defined as a set with a vague (ambiguous) boundary as compared to a crisp boundary of classical sets. Then, a membership function



(a) Example of the overlap of fuzzy sets to calculate the output.

(b) Example of the aggregation of the fuzzy sets.

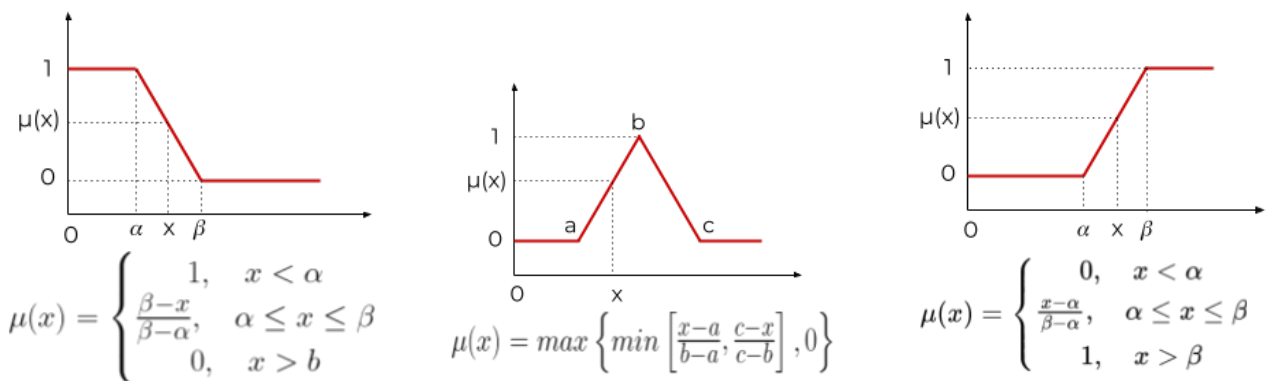
Figure 3.15: Aggregation of the fuzzy sets process.

(MF) is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1. The input space is sometimes referred to as the universe of discourse. Or, the membership function is a graphical representation of the magnitude of participation of each input [AAS15]. Therefore, considered the building blocks of fuzzy set theory.

In this work we chose the two most common forms of member functions:

- Triangular member functions;
- Trapezoidal member functions;

Figure 3.16 shows the graphical representation of these functions and the equations that define them.



(a) Schematic of an open-left membership function.

(b) Schematic of a triangular membership function.

(c) Schematic of an open-right membership function.

Figure 3.16: Schematics of three membership functions.

Grouping several membership functions we can create a set of fuzzy rules. Figure 3.17 shows how each function can be labeled as a fuzzy value. As shown in the same figure and what is used in this thesis, were created seven fuzzy values:

1. NL: Negative Large;
2. NM: Negative Medium;
3. NS: Negative Small;
4. Z: Zero;
5. PS: Positive Small;
6. PM: Positive Medium;
7. PL: Positive Large.

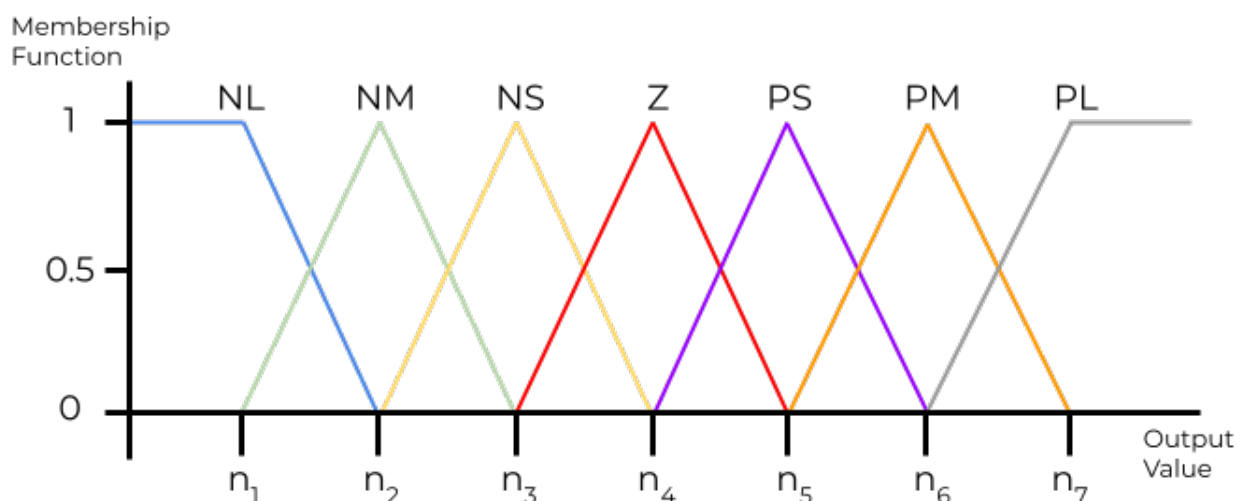


Figure 3.17: Example of a fuzzy system membership functions.

The  $x$  axis represents the range of possible outputs values, where  $x = n_1, n_2, \dots, n_i$ .

### PID Controller with Fuzzy System Gains Tuning

The success of the PID controller depends on the appropriate PID gains. Tuning the PID gains is very important to optimize performance. In practice, the PID gains are usually tuned by experienced human experts based on some “rule of thumb” [HGH<sup>+</sup>19]. In this work we implement a self-tuning PID controller using fuzzy theory.

The term self-tuning indicates the controller characteristics of tuning its control parameters on-line automatically so as to have the most suitable values of those gains obtained, which results in optimization of the process output. The design of controlling rules for a self-tuning fuzzy PID controller is based on theoretical and experimental analysis. The gains  $k_p$ ,  $k_i$  and  $k_d$  can be tuned by adjusting other controlling parameters and +coefficients

on-line, which increases the precision of control resulting in better performance than the classical PID controller [DBD16].

As in [AMY13], the self-tuning fuzzy PID controller implemented here, which takes error "e" and rate of change-in-error "ed" as the input to the controller makes use of the fuzzy controller rules to modify PID gains on-line. The self-tuning the PID controller refers to finding the fuzzy relationship between the three gains of PID, and and "e" and "ed", and according to the principle of fuzzy control modifying the three gains in order to meet different requirements for control gains when "e" and "ed" are different and making the control object produce a good dynamic and static performance. As seen in Figure 3.18.

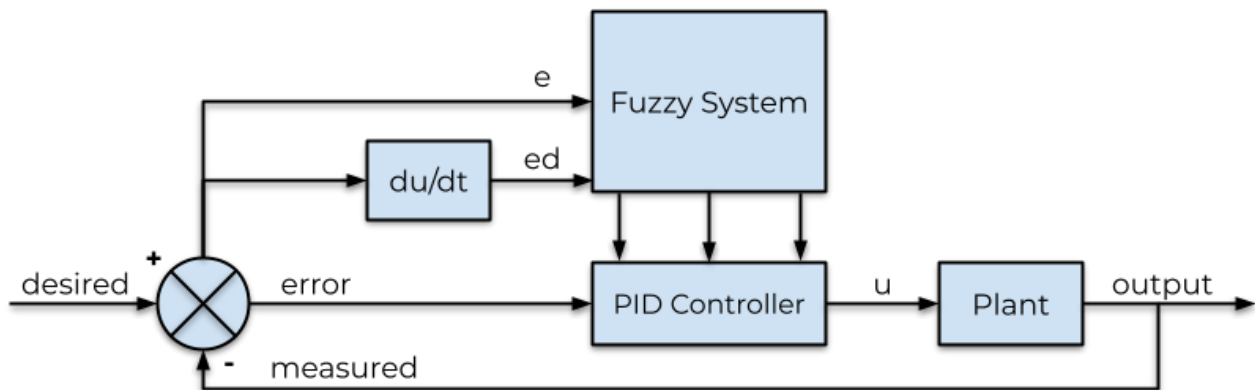


Figure 3.18: PID controller structure with Fuzzy System for gains tuning. Adapted from [AMY13].

The self-tuning PID controller seen in Figure 3.18, uses the theory shown in Figure 3.17 and in Table 3.2, is possible to create the Tables 3.3 and 3.4 that contains all the rules for the tuning of the PID gains.

Table 3.3: Fuzzy rules for  $K_p$  and  $K_i$ .

		Error						
		NL	NM	NS	Z	PS	PM	PL
Error Rate	NL	M	S	VS	VVS	VS	S	M
	NM	L	M	S	VS	S	M	L
	NS	VL	L	M	S	M	L	VL
	Z	VVL	VL	L	M	L	VL	VVL
	PS	VL	L	M	S	M	L	VL
	PM	L	M	S	VS	S	M	L
	PL	M	S	VS	VVS	VS	S	M

The rules presented at the Tables 3.3 and 3.4 can be read as follows: For example, IF the error is NL AND the error rate is PS THEN  $K_p$  is VL and  $K_i$  is VL and  $K_d$  is VS. The output of the fuzzy system logic is fuzzy. These outputs can't be input into the controller, then we can use equation 3.43 and the values  $n_i$  of Figure 3.17 to calculate the gains. Therefore, we can describe the tables as a set of rules:

Table 3.4: Fuzzy rules for  $K_d$ .

		Error						
		NL	NM	NS	Z	PS	PM	PL
Error Rate	NL	M	L	VL	VVL	VL	L	M
	NM	S	M	L	VL	L	M	S
	NS	VS	S	M	L	M	S	VS
	Z	VVS	VS	S	M	S	VS	VVS
	PS	VS	S	M	L	M	S	VS
	PM	S	M	L	VL	L	M	S
	PL	M	L	VL	VVL	VL	L	M

R1: IF 'error' IS NL AND 'error<sub>derivative</sub>' IS NL THEN 'k<sub>p</sub>' IS M

R2: IF 'error' IS NL AND 'error<sub>derivative</sub>' IS NM THEN 'k<sub>p</sub>' IS S

R3: IF 'error' IS NL AND 'error<sub>derivative</sub>' IS NS THEN 'k<sub>p</sub>' IS VS

R4: IF 'error' IS NL AND 'error<sub>derivative</sub>' IS Z THEN 'k<sub>p</sub>' IS VVS

⋮

Rn: IF 'error' IS Partition X AND 'error<sub>derivative</sub>' IS Partition Y THEN 'k<sub>p</sub>' IS Partition Z

With all parts defined, is possible to build the PID-Fuzzy algorithm. Figure 3.19 shows a diagram that explains the process of the estimation of the gains  $K_p$ ,  $K_i$  and  $K_d$  for attitude and height control.

### 3.9.2 Linear Quadratic Regulator - LQR

LQR is an optimal control technique that provides the best possible performance for some given performance measure. The LQR design problem is to design a state feedback controller  $K$  such that the objective function  $J$  is minimized (see eq. 3.46). In this technique, a feedback gain matrix is designed, which minimizes the objective function to achieve some compromise between the use of control effort, the magnitude, and the speed of response, guaranteeing a stable system [AVN16]. Figure 3.20 shows a block diagram representation of an LQR control.

The goal of the LQR is to drive all the states to zero in the fastest amount of time, given a set of constraints described in the weighting matrices  $Q$  and  $R$  [GR15]. To fully understand how a linear quadratic regulator works, first, we need to describe a continuous-time linear system, like the one in equation 3.45:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \quad (3.45)$$

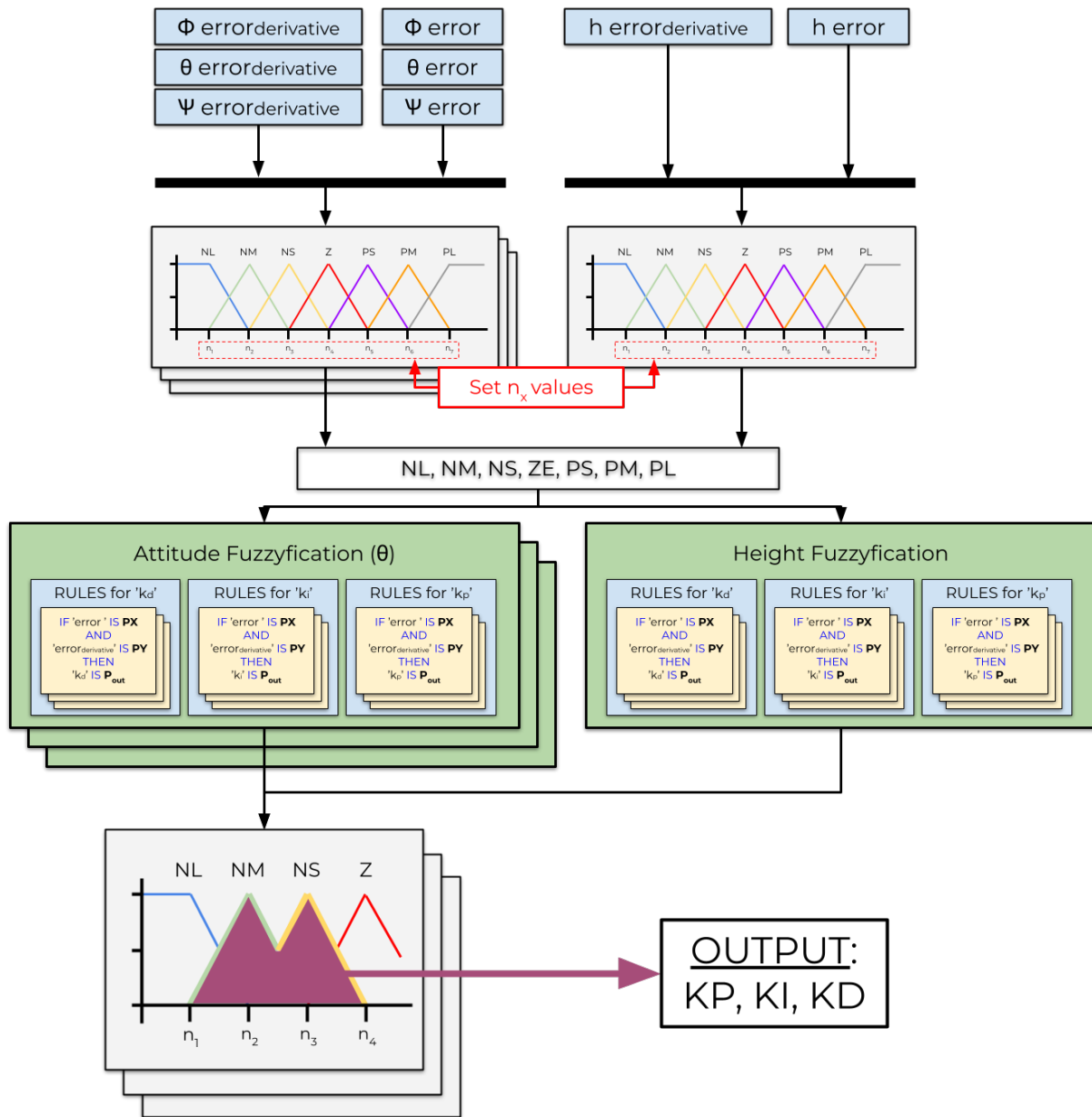


Figure 3.19: Diagram of the fuzzy algorithm for PID gains determination.

With a cost function defined as

$$J = \int_0^{\infty} (x^T Qx + u^T Ru) dt \tag{3.46}$$

Q and R are matrices representing the weights assigned to the state parameters and the input parameters. By varying the values of the two matrices, the total value of the cost function can be adjusted according to the desired output [SGP20]. Q is required to be a positive definite or positive semi-definite symmetry matrix; R is required to be a positive definite symmetry matrix. For better performance, matrix Q has to be changed, whereas

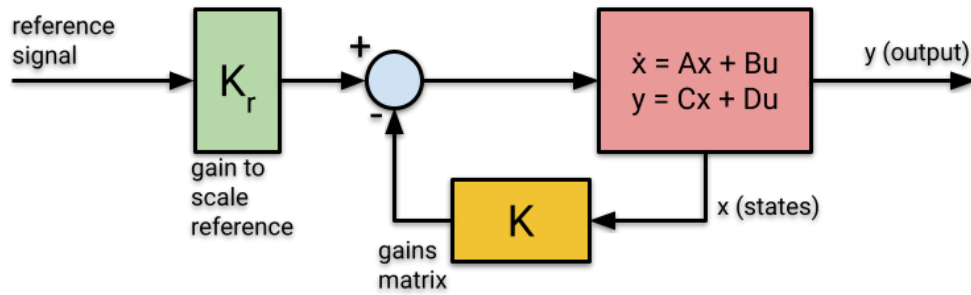


Figure 3.20: Block diagram of an LQR control system.

the R matrix values need to be adjusted for minimizing effort. The feedback control law that minimizes the value of the cost function is given by equation 3.47.

$$u = -Kx \quad (3.47)$$

Where matrix K is found by equation 3.48:

$$K = R^{-1}B^T P \quad (3.48)$$

And P is found by solving the continuous time Algebraic Riccati Equation (ARE) in equation 3.49:

$$A^T P + PA + Q - PBR^{-1}B^T P = 0 \quad (3.49)$$

As stated before, the LQR only brings the states to zero; it does not track the references. For that, we need an LQR controller with integral action.

### LQR with Integral Action

In order to obtain zero steady-state error, an integral action is included in the LQR Control. The basic approach in integral feedback is to create a state within the controller that computes the integral of the error signal, which is then used as a feedback term [AVN16]. It is done by augmenting the description of the system with a new state  $\dot{q}$ :

$$\dot{q} = ref - y = ref - Cx \quad (3.50)$$

The Figure 3.21 shows a block diagram an LQR control system with integral action.

The augmented system is described in equations 3.51 and 3.52.

$$\frac{d}{dt} \begin{bmatrix} x \\ \dot{q} \end{bmatrix} = \begin{bmatrix} Ax + Bu \\ y - ref \end{bmatrix} = \begin{bmatrix} Ax + Bu \\ Cx - ref \end{bmatrix} \quad (3.51)$$

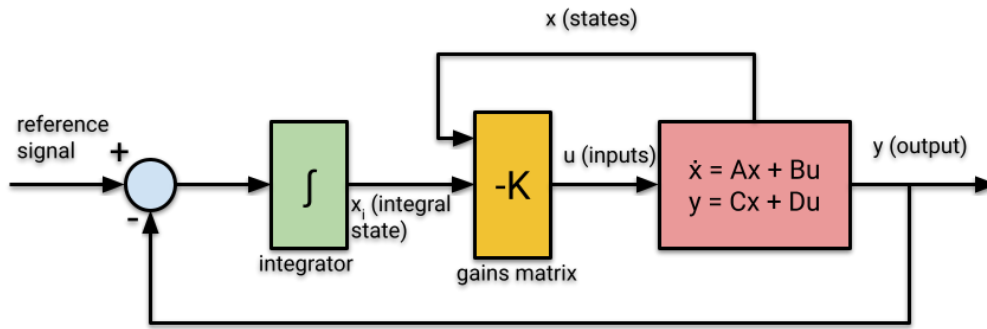


Figure 3.21: Block diagram of an LQR control system with integral action.

$$\tilde{A} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} B \\ 0 \end{bmatrix}, \quad \tilde{C} = \begin{bmatrix} C & 0 \end{bmatrix} \quad (3.52)$$

The final compensator is given by equation 3.53.

$$u = -K[x; x_i] \quad (3.53)$$

### 3.9.3 Model Predictive Control - MPC

Model Predictive Control (MPC) has become an established control technology owing to its capability of solving problems with physical constraints [LWM08]. The MPC uses a system model to predict the future states of the system and generates a control vector that minimizes a specific cost function over the prediction horizon in the presence of disturbances and constraints [AJS14]. The name "Model Predictive Control" arises from how the control law is computed. At the *present time*  $k$ , the behavior of the process over a horizon  $p$  is considered. Using a model, the process response to changes in the manipulated variable is predicted. The moves of the manipulated variables are selected such that the predicted response has certain desirable characteristics. Only the first computed change in the manipulated variable is implemented. At the time  $(k + 1)$  the computation is repeated with the horizon moved by a one-time interval [GPM89].

For the modeling of the MPC controller, we can look at the work of Wang [Wan09] for an MPC design approach. Moreover, first, we can assume that the soon to be controlled discrete plant can be represented as an adaptation of equation 3.45 :

$$\begin{aligned} x_m(k + 1) &= A_m x_m(k) + B_m u(k), \\ y(k) &= C_m x_m(k) \end{aligned} \quad (3.54)$$



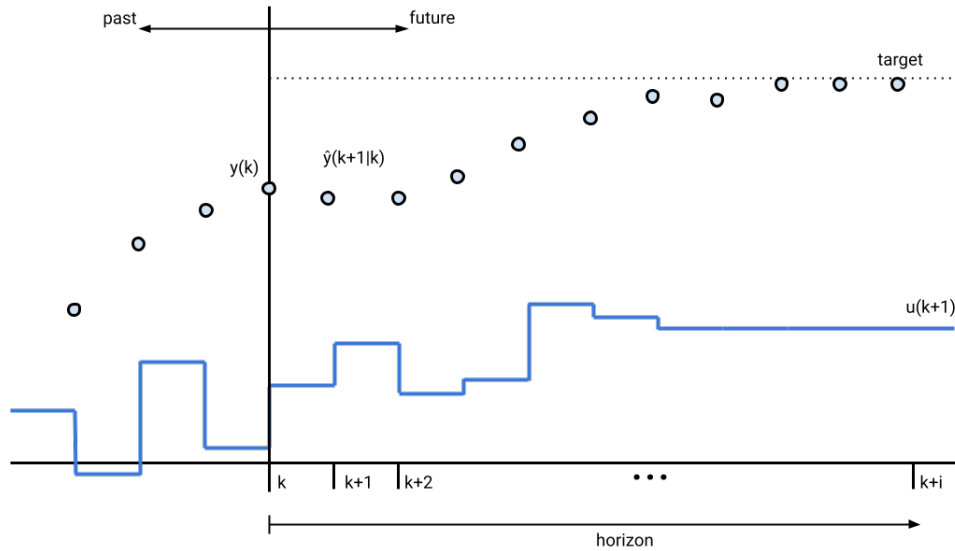


Figure 3.22: The "moving horizon" approach of Model Predictive Control. Adapted from [GPM89].

where  $u$  is the manipulated variable or input variable,  $y$  is the process output, and  $x_m$  is the state variable vector.

Notably, a general formulation of a state-space model has a direct term from the input signal  $u(k)$  to the output  $y(k)$  as shown in equation 3.55.

$$y(k) = C_m x_m(k) + D_m u(k) \quad (3.55)$$

Due to the principle of receding horizon control, it is assumed that the input  $u(k)$  cannot affect the output  $y(k)$  at the same time. Thus,  $D_m = 0$  in the plant model of equation 3.54.

Placing a difference operation in both sides of  $x_m(k+1) = A_m x_m(k) + B_m u(k)$ , produces:

$$x_m(k+1) - x_m(k) = A_m(x_m(k) - x_m(k-1)) + B_m(u(k) - u(k-1)) \quad (3.56)$$

Making the the difference of the state variable:

$$\Delta x_m(k+1) = x_m(k+1) - x_m(k) \quad (3.57)$$

and

$$\Delta x_m(k) = x_m(k) - x_m(k-1) \quad (3.58)$$

The difference of the control variable is:

$$\Delta u(k) = u(k) - u(k - 1) \quad (3.59)$$

Equations 3.57, 3.58 and 3.59 represent the increments of the variables  $x_m(k)$  and  $u(k)$ . Then, the difference of the state-space equation is:

$$\Delta x_m(k + 1) = A_m \Delta x_m(k) - B_m \Delta u(k) \quad (3.60)$$

In equation 3.60 the input of the state-space model is given by  $\Delta u(k)$ , and to connect  $\Delta x_m(k)$  to the output  $y(k)$  a new state variable vector is chosen:

$$x(k) = [\Delta x_m(k)^T \ y(k)^T]^T \quad (3.61)$$

and the output gives:

$$\begin{aligned} y(k + 1) - y(k) &= C_m(x_m(k + 1) - x_m(k)) = C_m \Delta x_m(k + 1) \\ &= C_m A_m \Delta x_m(k) + C_m B_m \Delta u(k) \end{aligned} \quad (3.62)$$

Merging equations 3.60 and 3.62 it is possible to recreate the space-state model as:

$$\begin{aligned} \overbrace{\begin{bmatrix} \Delta x_m(k + 1) \\ y(k + 1) \end{bmatrix}}^{x(k+1)} &= \overbrace{\begin{bmatrix} A_m & o_m^T \\ C_m A_m & 1 \end{bmatrix}}^A \overbrace{\begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix}}^{x(k)} + \overbrace{\begin{bmatrix} B_m \\ C_m B_m \end{bmatrix}}^B \Delta u(k) \\ y(k) &= \overbrace{\begin{bmatrix} o_m & 1 \end{bmatrix}}^C \begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix} \end{aligned} \quad (3.63)$$

where  $o_m = \overbrace{\begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}}^{n_1}$ ,  $n_1$  is the assumed dimension of the state variable vector, and the triplet  $(A, B, C)$  is called the augmented model.

After having determined the mathematical model, we need to calculate the predicted plant output with the future control signal as the adjustable variables. This prediction is described within an optimization window. But first, we need to assume that at the sampling instant  $k_i$ ,  $k_i > 0$ , the state variable vector  $x(k_i)$  is available through measurement, the state  $x(k_i)$  provides the current plant information.

The future control trajectory of a single-input and single-output system is given by:

$$\Delta u(k_i), \Delta u(k_i + 1), \dots, \Delta u(k_i + N_c - 1) \quad (3.64)$$

where  $N_c$  is called the control horizon dictating the number of parameters used to capture the future control trajectory.

The future state variables are described as:

$$x(k_i + 1|k_i), x(k_i + 2|k_i), \dots, x(k_i + m|k_i), \dots, x(k_i + N_p|k_i) \quad (3.65)$$

where  $N_p$  are the number of samples, or the prediction horizon,  $x(k_i + m|k_i)$  is the predicted state variable at  $k_i + m$  with given current plant information  $x(k_i)$ .

Based on the state-space model  $(A, B, C)$ , the future state variables are calculated:

$$\begin{aligned} x(k_i + 1|k_i) &= Ax(k_i) + B\Delta u(k_i) \\ x(k_i + 2|k_i) &= Ax(k_i + 1|k_i) + B\Delta u(k_i + 1) \\ x(k_i + 3|k_i) &= A^2x(k_i) + AB\Delta u(k_i) + B\Delta u(k_i + 1) \\ &\vdots \\ x(k_i + N_p|k_i) &= A^{N_p}x(k_i) + A^{N_p-1}B\Delta u(k_i) + A^{N_p-2}B\Delta u(k_i + 1) + \dots + A^{N_p-N_c}B\Delta u(k_i + N_c - 1) \end{aligned} \quad (3.66)$$

Then, the predicted output variables are:

$$\begin{aligned} y(k_i + 1|k_i) &= CAx(k_i) + CB\Delta u(k_i) \\ y(k_i + 2|k_i) &= CA^2x(k_i) + CAB\Delta u(k_i) + CB\Delta u(k_i + 1) \\ y(k_i + 3|k_i) &= CA^3x(k_i) + CA^2B\Delta u(k_i) + CAB\Delta u(k_i + 1) + CB\Delta u(k_i + 2) \\ &\vdots \\ y(k_i + N_p|k_i) &= CA^{N_p}x(k_i) + CA^{N_p-1}B\Delta u(k_i) + CA^{N_p-2}B\Delta u(k_i + 1) + \dots + CA^{N_p-N_c}B\Delta u(k_i + N_c - 1) \end{aligned} \quad (3.67)$$

All predicted variables are formulated in terms of current state variable information  $x(k_i)$  and the future control movement  $\Delta u(k_i + j)$ , where  $j = 0, 1, \dots, N_c - 1$ . Moreover, we can define the following vectors:

$$\begin{aligned} Y &= \left[ y(k_i + 1|k_i) \quad y(k_i + 2|k_i) \quad y(k_i + 3|k_i) \quad \dots \quad y(k_i + N_p|k_i) \right]^T \\ \Delta U &= \left[ \Delta u(k_i) \quad \Delta u(k_i + 1) \quad \Delta u(k_i + 2) \quad \dots \quad \Delta u(k_i + N_c - 1) \right]^T \end{aligned} \quad (3.68)$$

where the dimension of  $Y$  is  $N_p$  and the dimension of  $\Delta U$  is  $N_c$ . Putting together equations 3.66 and 3.67 in a compact matrix form we get:

$$Y = Fx(k_i) + \Phi\Delta U \quad (3.69)$$

where

$$F = \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{N_p} \end{bmatrix}; \quad \Phi = \begin{bmatrix} CB & 0 & 0 & \dots & 0 \\ CAB & CB & 0 & \dots & 0 \\ CA^2B & CAB & CB & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{N_p-1}B & CA^{N_p-2}B & CA^{N_p-3}B & \dots & CA^{N_p-N_c}B \end{bmatrix} \quad (3.70)$$

The objective of the predictive control system is to bring the predicted output as close as possible to the set-point signal, where we assume that the set-point signal remains constant in the optimization window [Wan09]. We need to find an optimal control parameter vector  $\Delta U$  such that an error function between the set-point and the predicted output is minimized.

Equation 3.71 shows the data vector that contains the set-point information:

$$R_s^T = \overbrace{\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}}^{N_p} r(k_i) \quad (3.71)$$

and the cost function  $J$  that reflects the control objective is:

$$J = (R_s - Y)^T (R_s - Y) + \Delta U^T \bar{R} \Delta U \quad (3.72)$$

where the first term is linked to the objective of minimizing the errors between the predicted output and the set-point signal while the second term reflects the consideration given to the size of  $\Delta U$  when the objective function  $J$  is made to be as small as possible.  $\bar{R}$  is a diagonal matrix used as a tuning parameter for the desired closed-loop performance.

Using equation 3.69 to find the optimal  $\Delta U$  that minimizes  $J$ :

$$J = (R_s - Fx(k_i))^T (R_s - Fx(k_i)) - 2\Delta U^T \Phi^T (R_s - Fx(k_i)) + \Delta U^T (\Phi^T \Phi + \bar{R}) \Delta U \quad (3.73)$$

The first derivative of the cost function  $J$ :

$$\frac{\partial J}{\partial \Delta U} = -2\Phi^T (R_s - Fx(k_i)) + 2(\Phi^T \Phi + \bar{R}) \Delta U \quad (3.74)$$

the necessary condition of the minimum  $J$  is obtained as

$$\frac{\partial J}{\partial \Delta U} = 0 \quad (3.75)$$

then, the optimal solution for the control signal is

$$\Delta U = (\Phi^T \Phi + \bar{R})^{-1} \Phi^T (R_s - Fx(k_i)) \quad (3.76)$$

In equation 3.76, the matrix  $(\Phi^T \Phi + \bar{R})^{-1}$  is the Hessian matrix, and  $R_s$  is a data vector that contains the set-point information expressed as

$$R_s = \overbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix}^T}_{N_p} r(k_i) = \bar{R}_s r(k_i) \quad (3.77)$$

where

$$\bar{R}_s = \overbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix}^T}_{N_c} \quad (3.78)$$

Finally, we can link the set-point signal and the state variable with the optimal solution of the control signal with the following equation:

$$\Delta U = (\Phi^T \Phi + \bar{R})^{-1} (\Phi^T R_s - \Phi^T Fx(k_i)) \quad (3.79)$$

in equation 3.79,  $(\Phi^T \Phi + \bar{R})^{-1} \Phi^T R_s$  corresponds to the set-point change, while  $-(\Phi^T \Phi + \bar{R})^{-1} \Phi^T F$  corresponds to the state feedback control within the framework of predictive control. The receding horizon control principle makes that only the first element of  $\Delta U$  at time  $k_i$  is used as the incremental control, then

$$\begin{aligned} \Delta u(k_i) &= \overbrace{\begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}}^{N_c} (\Phi^T \Phi + \bar{R})^{-1} (\Phi^T \bar{R}_s r(k_i) - \Phi^T Fx(k_i)) \\ &= K_y r(k_i) - K_{mpc} x(k_i) \end{aligned} \quad (3.80)$$

where  $K_y$  is the first element of

$$(\Phi^T \Phi + \bar{R})^{-1} \Phi^T \bar{R}_s \quad (3.81)$$

and  $K_{mpc}$  is the first row of

$$(\Phi^T \Phi + \bar{R})^{-1} \Phi^T F \quad (3.82)$$

Equation 3.80 shows a linear time-invariant state feedback control, where the state feedback control gain vector is  $K_{mpc}$ . Combining with the following augmented model:

$$x(k+1) = Ax(k) + B\Delta u(k) \quad (3.83)$$

the closed-loop system is obtained by substituting the equation 3.80 into the augmented system equation; changing index  $k_i$  to  $k$ , leading to the closed-loop equation:

$$\begin{aligned} x(k+1) &= Ax(k) - BK_{mpc}x(k) + BK_y r(k) \\ x(k+1) &= (A - BK_{mpc})x(k) + BK_y r(k) \end{aligned} \quad (3.84)$$

Up to this point, the formulation of the MPC was related to single-input, single-output (SISO) systems, but the Quadrotor platform is described as a multiple-input, multiple-output (MIMO) system. Therefore we need to transform the previous theory to accommodate a MIMO system. First, assume that the plant has  $m$  inputs,  $q$  outputs, and  $n_1$  states. According to Wang [Wan09], the general formulation of the predictive control problem takes the plant noise and disturbance into consideration.

$$\begin{aligned} x_m(k+1) &= A_m x_m(k) + B_m u(k) + B_d \omega(k) \\ y(k) &= C_m x_m(k) \end{aligned} \quad (3.85)$$

So, the input disturbance is related to a zero-mean, white noise sequence  $\epsilon(k)$ :

$$\omega(k) = \omega(k-1) = \epsilon(k) \quad (3.86)$$

then

$$x_m(k) = A_m x_m(k-1) + B_m u(k-1) + B_d \omega(k-1) \quad (3.87)$$

considering that,  $\Delta x_m(k) = x_m(k) - x_m(k-1)$  and  $\Delta u(k) = u(k) - u(k-1)$ , then

$$\Delta x_m(k) = A_m \Delta x_m(k) + B_m \Delta u(k) + B_d \epsilon(k) \quad (3.88)$$

to relate the output  $y(k)$  to the state variable  $\Delta x_m(k)$ , then

$$\Delta y(k+1) = C_m \Delta x_m(k+1) = A_m C_m \Delta x_m(k) + C_m B_m \Delta u(k) + C_m B_d \epsilon(k) \quad (3.89)$$

where  $\Delta y(k) = y(k) - y(k-1)$ .

With a new state variable vector  $x(k) = \begin{bmatrix} \Delta x_m(k)^T & y(k)^T \end{bmatrix}^T$ , then

$$\begin{aligned} \begin{bmatrix} \Delta x_m(k+1) \\ y(k+1) \end{bmatrix} &= \begin{bmatrix} A_m & o_m^T \\ C_m A_m & I_{q \times q} \end{bmatrix} \begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix} + \begin{bmatrix} B_m \\ C_m B_m \end{bmatrix} \Delta u(k) + \begin{bmatrix} B_d \\ C_m B_d \end{bmatrix} \epsilon(k) \\ y(k) &= \begin{bmatrix} o_m & I_{q \times q} \end{bmatrix} \begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix} \end{aligned} \quad (3.90)$$

where  $I_{q \times q}$  is an identity matrix with  $q \times q$  dimension,  $o_m$  is a  $q \times n_1$ , then we can rewrite equation 3.90 as:

$$\begin{aligned} x(k+1) &= Ax(k) + B\Delta u(k) + B_e \epsilon \\ y(k) &= Cx(k) \end{aligned} \quad (3.91)$$

In a multiple-input, multiple-output environment, we can define the vectors  $Y$  and  $\Delta U$  as

$$\begin{aligned} \Delta U &= \left[ \Delta u(k_i)^T \quad \Delta u(k_i+1)^T \quad \Delta u(k_i+N_c-1)^T \right]^T \\ Y &= \left[ y(k_i+1|k_i)^T \quad y(k_i+2|k_i)^T \quad y(k_i+3|k_i)^T \quad \cdots \quad y(k_i+N_p|k_i)^T \right]^T \end{aligned} \quad (3.92)$$

Based on the model  $(A, B, C)$  the future state variables are calculated sequentially:

$$\begin{aligned} x(k_i+1|k_i) &= Ax(k_i) + B\Delta u(k_i) + B_d \epsilon(k_i) \\ x(k_i+2|k_i) &= Ax(k_i+1|k_i) + B\Delta u(k_i+1) + B_d \epsilon(k_i+1|k_i) \\ x(k_i+3|k_i) &= A^2x(k_i) + AB\Delta u(k_i) + B\Delta u(k_i+1) \\ &\quad + AB_e \epsilon(k_i) + B_d \epsilon(k_i+1|k_i) \\ &\quad \vdots \\ x(k_i+N_p|k_i) &= A^{N_p}x(k_i) + A^{N_p-1}B\Delta u(k_i) + A^{N_p-2}B\Delta u(k_i+1) \\ &\quad + A^{N_p-N_c}B\Delta u(k_i+N_c-1) + A^{N_p-1}B_d \epsilon(k_i) \\ &\quad + A^{N_p-2}B_d \epsilon(k_i+1|k_i) + \dots + B_d \epsilon(k_i+N_p-1|k_i) \end{aligned} \quad (3.93)$$

That way we can put the set of equations in 3.93 in the format:

$$Y = Fx(k_i) + \Phi \Delta U \quad (3.94)$$

where  $F$  and  $\Phi$  are represented by matrices shown in 3.70. The incremental optimal control within one optimization window is given by

$$\Delta U = (\Phi^T \Phi + \bar{R})^{-1} (\Phi^T \bar{R}_s - \Phi^T Fx(k_i)) \quad (3.95)$$

where matrix  $\Phi^T \Phi$  have size  $mN_c \times mN_c$ ,  $\Phi^T F$  have size  $mN_c \times n$  and  $\Phi^T \bar{R}_s$  is equal to the last  $q$  columns of  $\Phi^T F$ .

With the receding horizon control principle, the first  $m$  elements in  $\Delta U$  are taken to form the incremental optimal control:

$$\begin{aligned}\Delta u(k_i) &= \overbrace{\begin{bmatrix} I_m & o_m & \cdots & o_m \end{bmatrix}}^{N_c} (\Phi^T \Phi + \bar{R})^{-1} (\Phi^T \bar{R}_s r(k_i) - \Phi^T F x(k_i)) \\ &= K_y r(k_i) - K_{mpc} x(k_i)\end{aligned}\quad (3.96)$$

where  $I_m$  is the identity matrix with size  $m \times m$  and the  $o_m$  is a zero matrix with size  $m \times m$ .

### 3.10 Overall View of The Quadrotor Study Case

This section summarizes the topics addressed in this chapter, as illustrated in Figure 3.23.

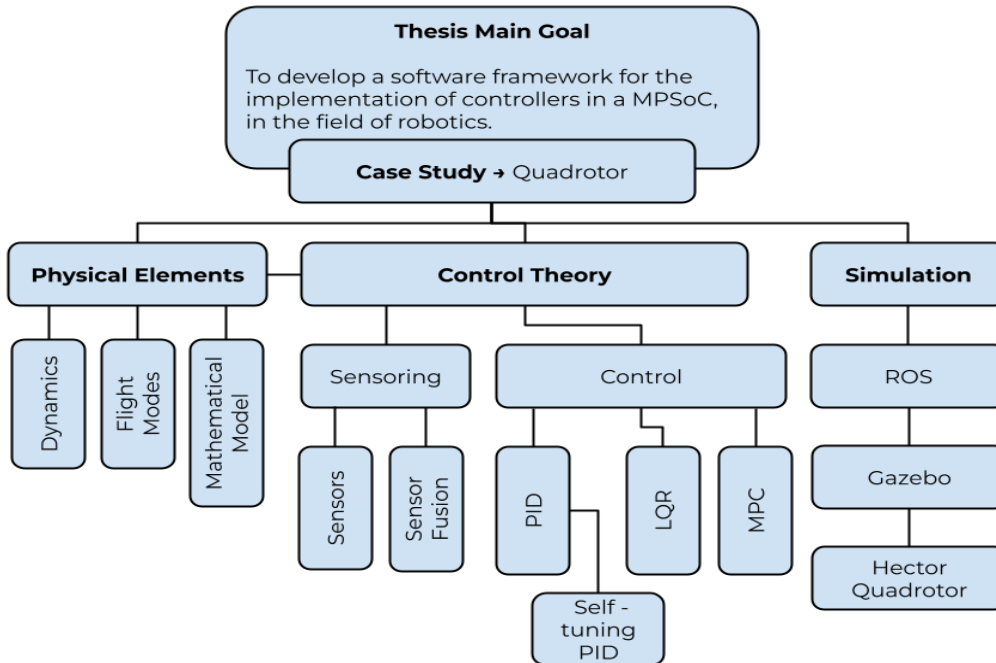


Figure 3.23: Chapter 3 research topics summary diagram.

To create the control system to implement in the thesis, the study case of the quadrotor breaks into three parts: the physical aspects of the quadrotor, like dynamic characteristics, flight modes, and mathematical modeling. The second part is the theory related to the quadrotor's control: to be possible to understand how the quadrotor can be controlled and which of them were chosen for this thesis. Lastly, the process to be controlled is chosen in the form of a quadrotor simulation.

To better understand this chapter and to see how the previous sections are linked. The author refers to the block diagram representation of a closed-loop control system in Figure 2.2. Based on this figure and the research made in this chapter, we can represent how the control system of the quadrotor was built. Figure 3.24 shows a closed-loop sys-



tem diagram showing how each research topic of this chapter relates to the overall control system.

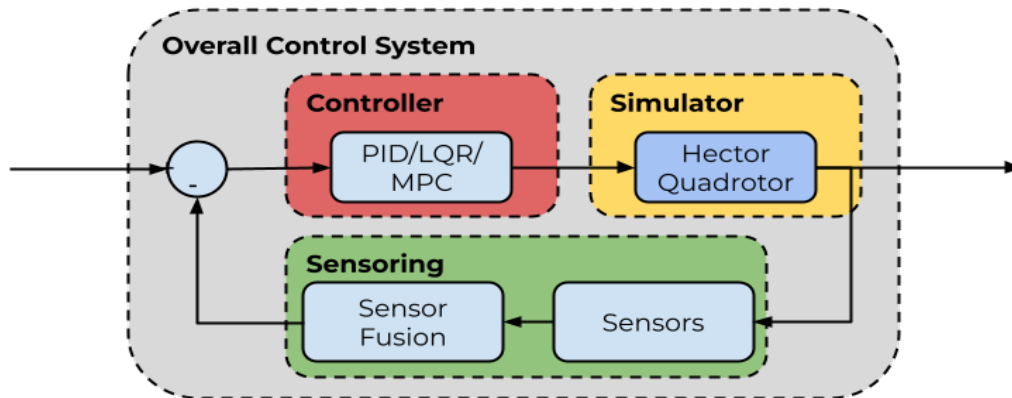


Figure 3.24: Closed-loop system diagram with Chapter 3 research topics.

## 4. HARDWARE PLATFORM AND SUPPORTING RESOURCES

This chapter presents the hardware and software basis used in this thesis. The author defines concepts of Multiprocessed System-on-chip in Section 4.1, the RISC-V processor architecture in Section 4.2, and the ORCA platform in Section 4.4. This Chapter it also explains the Hellfire operational system running in the RISC-V architecture in Section 4.3. The URSA simulator where the ORCA platform is simulated is presented in Section 4.5. Lastly, this chapter brings the software basis for the energy estimation in the processor in Section 4.6.1. All of the sections of this chapter are not original works from this thesis.

### 4.1 Multiprocessed System-on-chip

Limitations of power and performance create the impossibility to continue to increase the processing power of monolithic processor architectures. One solution for this problem is the Multiprocessed System-on-chip (MPSoC) architecture.

An MPSoC is a VLSI system that incorporates most or all the components necessary for an application that uses multiple programmable processors as system components. MPSoCs are widely used in networking, communications, signal processing, and multimedia, among other applications [WJM08].

An MPSoC consists of a set of processing elements (PEs) interconnected by some network. These networks are defined by an interconnect topology and several specific policies and structures, such as communication packet structure, routing algorithms etc. [YBDM03].

This work adopts as standard a set of characteristics found in several MPSoC architectures [MSC<sup>+</sup>14], which are:

- each PE contains at least one processor with a private memory (scratch-pad memory);
- the employed communication model is message passing;
- there is no shared memory in the system;
- applications are modeled as task graphs;
- a multi-tasking operating system (OS) runs at each PE;
- a mapping function assigns tasks to PEs, being possible to have more than one task per PE.

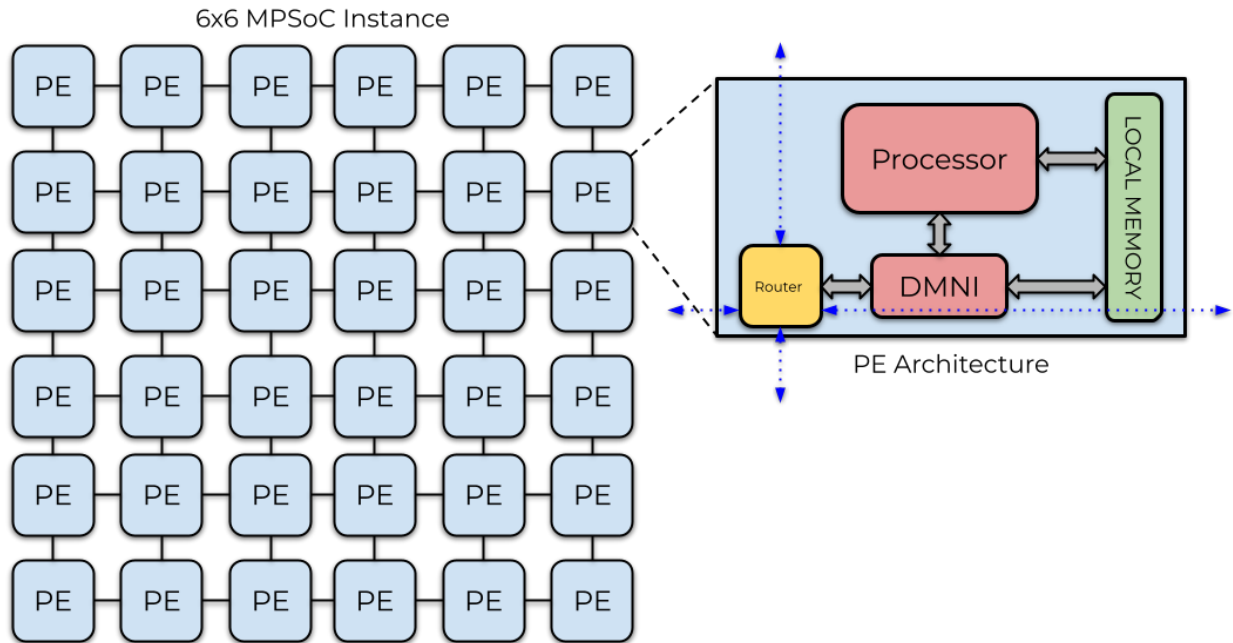


Figure 4.1: MPSoC Diagram of a  $6 \times 6$  MPSoC instance and the internal structure of a processing element, typically composed of a processor, a local memory, a DMA/network interface module (DMNI) and a network router. Adapted from [PUC19].

Figure 4.1 shows a diagram with the representation of a  $6 \times 6$  MPSoC instance with the information of the elements that usually form a PE.

Jerraya et al. [JW05] state that MPSoCs became an attractive class of computing architectures because they provide support for massively parallel, real-time embedded applications with low-energy consumption and small-sized chips. Moreover, these architectures can handle multiple processes simultaneously, even without data parallelism, as opposed to graphics processing units (GPUs).

The architecture used in this thesis is based on an MPSoC, which its PE includes a RISC-V processor. This processor architecture is detailed in Section 4.2.

## 4.2 RISC-V Processor Architecture

Architectures like x86 and ARM are widely available and supported. However, they are also complex, and the licensing model is difficult for experimental and academic use. Then, to create an alternative *ISA*, Krste Asanovi, Andrew Waterman, and Yunsup Lee developed the RISC-V architecture at UC Berkeley [Kan16].

In the past few years, a large number of both proprietary and open-source RISC-V implementations emerged. Furthermore, RISC-V ecosystems have been developed to pro-

vide software compilers, System-on-Chip (SoC) peripherals, and other components, simplifying the generation of FPGA- or ASIC-based RISC-V processors systems [DAK<sup>+</sup>21].

According to [WLPA11] RISC-V is designed to support computer architecture research and education. Furthermore, their goals with this architecture were:

- Provide a realistic but open ISA that captures important details of commercial general-purpose ISA designs, and that is suitable for direct hardware implementation.
- Provide a small but complete base ISA that avoids “over-architecting” for a particular microarchitecture style or implementation technology but allows efficient implementation in any of them.
- Support for both 32-bit and 64-bit address space variants for applications, operating system kernels, and hardware implementations.
- Support highly-parallel multicore or manycore implementations, including heterogeneous multiprocessors.
- Support an efficient dense instruction encoding with variable-length instructions, improving performance and reducing energy and code size.
- Support the revised 2008 IEEE 754 floating-point standard.
- Be fully virtualizable.
- Be simple to subset for educational purposes and reduce the complexity of bringing up new implementations.
- Support experimentation with user-level ISA extensions and specialized variants.
- Support independent experimentation with new supervisor-level ISA designs.

According to [Kan16], RISC-V is a three-operand load-store architecture that heavily emphasizes cleanliness and simplicity. As Table 4.1 shows that it effectively has three base instruction sets and six extensions. The base ISA plus MADF extensions form a general-purpose ISA (sometimes known as the G extensions) that can handle scalar integer or floating-point code readily.

### 4.3 HellfireOS

The operating system used in the MPSoC in this project is the HellfireOS. Created in the GSE laboratory of the Pontifical Catholic University of Rio Grande do Sul (PUCRS),

Table 4.1: RISC-V base instruction sets and extensions. Adapted from: [Kan16].

Base ISA	Instructions	Description
<b>RV32I</b>	47	32-bit address space and integer instructions
<b>RV32E</b>	47	Subset of RV32I, restricted to 16 registers
<b>RV64I</b>	59	64-bit address space and integer instructions, along with several 32-bit integer instruction
<b>RV128I</b>	71	128-bit address space and integer instructions, along with several 64- and 32-bit instructions
Extension	Instructions	Description
<b>M</b>	8	Integer multiply and divide
<b>A</b>	11	Atomic memory operations, load-reserve/store conditional
<b>F</b>	26	Single-precision (32 bit) floating point
<b>D</b>	26	Double-precision (64 bit) floating point; requires F extension
<b>Q</b>	26	Quad-precision (128 bit) floating point; requires F and D extensions
<b>C</b>	46	Compressed integer instructions; reduces size to 16 bits

this OS is a real-time operating system based on models for both the application and architecture features concerning modern MPSoCs [SFAM<sup>+</sup>12].

To allow high-level platform customization, HellfireOS was implemented modularly, where each module corresponds to some specific functionality [AH12]. Such as the maximum number of tasks on a processor, and the task stack size, among others. The idea is to allow the designer to optimize the final kernel image size according to software application and system constraints. Thus, allowing the designer to make the system more suitable to run on low memory constrained architectures, like typical critical embedded systems [ASFM<sup>+</sup>10].

The HellfireOS is organized in layers, and all hardware-specific functions are defined in the first layer, known as HAL (Hardware Abstraction Layer). The uKernel lies just above it, and the communication, migration, memory management, mutual exclusion drivers, and the API are placed over the uKernel layer. The user applications belong to the top layer [MSFLH14]. Figure 4.2 shows this organization.

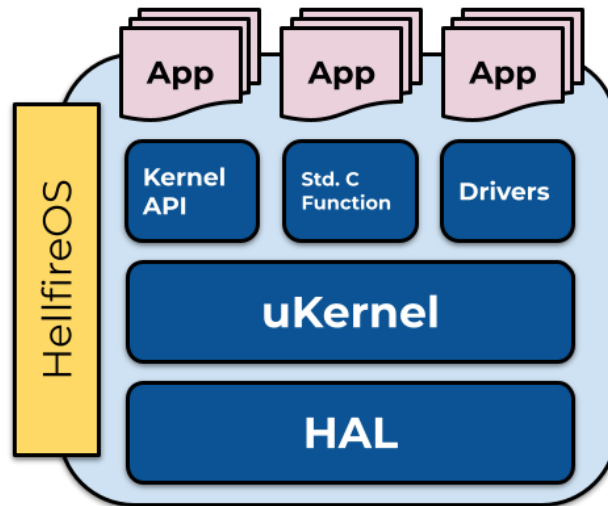


Figure 4.2: HellfireOS Structure. Adapted from [AH12].

From a single processor point of view, the designer can develop the application C code and run it over the Hellfire Operating System [MLSF<sup>+</sup>12].

#### 4.4 ORCA Platform

In [D<sup>+</sup>20], the author proposes ORCA (self-adaptive multiprocessOR system-on-Chip platform), a development platform to aid in designing self-adaptive systems. The platform provides abstractions to deal with self-adaptation complexity, including a configurable hardware architecture, operating system, and software libraries.

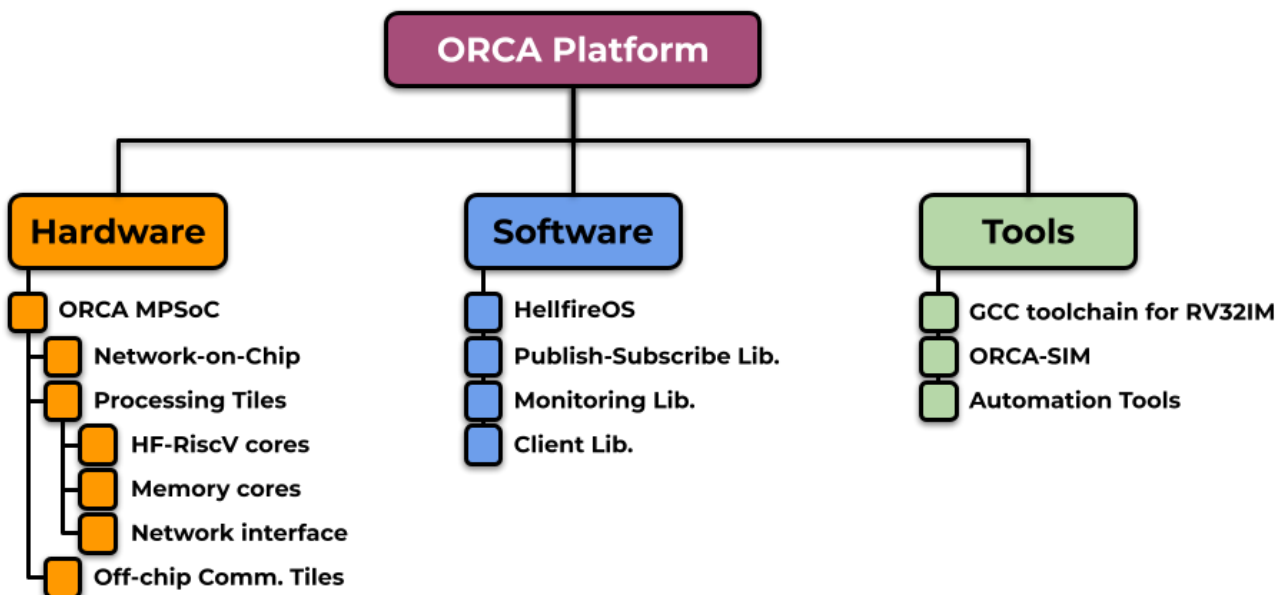


Figure 4.3: The organization of ORCA platform, depicting software, hardware, and tools. Adapted from [D<sup>+</sup>20].

According to [VDP+20], the ORCA platform allows for the design-time configuration of multiple MPSoC parameters, including the number of nodes, router buffer width, and processor memory size. The configuration adopted in this paper has a message size of 64 16-bit flits. Communication is carried out by an instance of the Hermes NoC [MCM+04], with a buffer depth of 16 flits, XY routing algorithm, and wormhole packet switching.

In [DJSFA19], the authors configure ORCA to have two node types:

- **Processing nodes:** comprising processor core, three scratchpad memory modules, network interface and router;
- **Networking nodes:** comprising a network bridge, two scratchpad memory modules, network interface, and router

Moreover, this configuration is used in this work. Figure 4.4 shows the arrangement of a  $2 \times 2$  ORCA MPSoC, similar to what was used in some cases of this thesis.

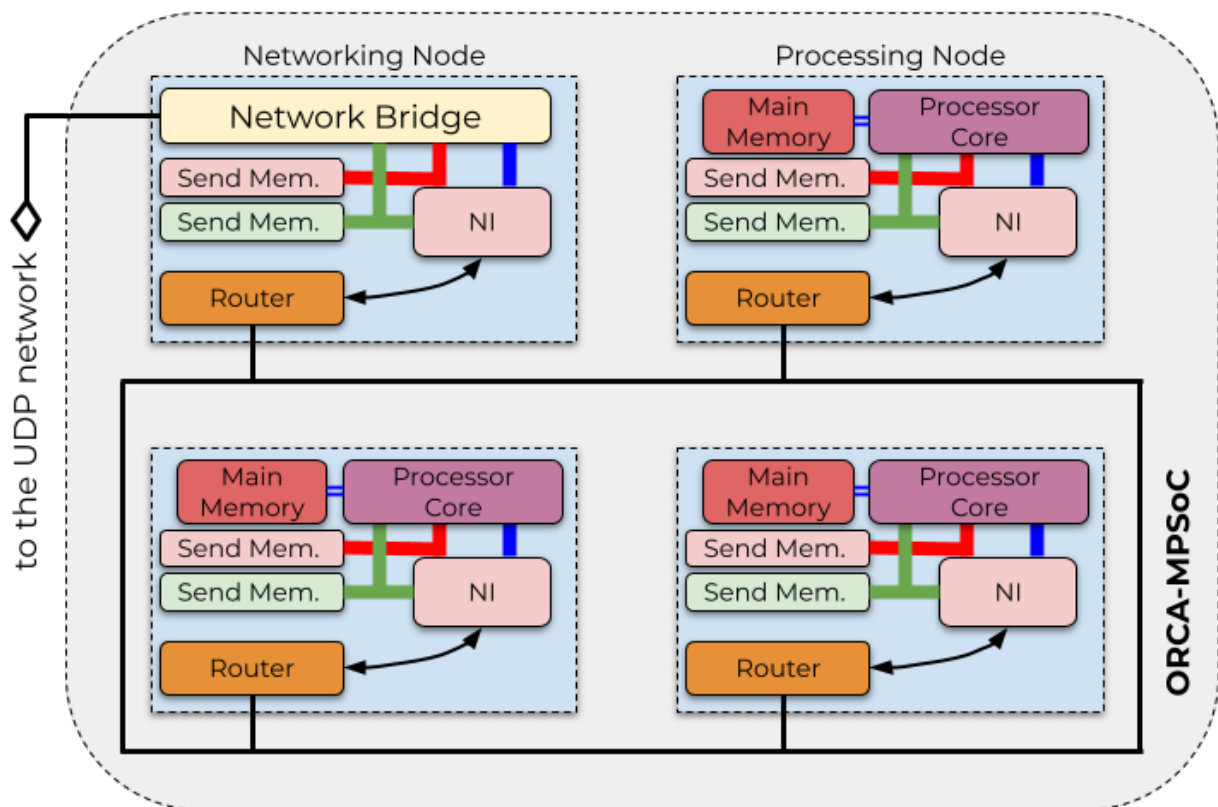


Figure 4.4: The MPSoC comprising four different nodes. The top-left most node is a networking node, while the others are processing nodes. Nodes are interconnected through a network-on-chip, namely Hermes. Adapted from [DJSFA19].

## 4.5 URSA Simulator

As the ORCA platform was proposed in [D<sup>+</sup>20], the URSA (Micro ( $\mu$ ) Rapid-Simulation API) is also first presented in the same work. The URSA is defined as an application programming interface (API) for the modeling and simulation of computing platforms. In the context of this thesis, the URSA is used to model and simulate the ORCA MPSoC.

URSA is a C++ API for system-level modeling and simulation. It provides a set of language-related assets that can be used to create system-level, cycle-accurate hardware simulators, like SystemC. The URSA hardware models are modeled as a set of finite state machines (FSM), and its underlying simulation is based on discrete-event simulation. A clock cycle in URSA corresponds to the activation of the transition function of the FSMs of the simulated system [JMdma<sup>+</sup>21].

The processor core model implemented in [D<sup>+</sup>20] and used in this thesis utilizes either the RV32I or RV32IM (see Table 4.1) instruction sets of the Risc-V user mode standard, with 32 user-level registers in the architecture (from x04 to x31), and four core instruction formats (R, I, S, and U-type).

## 4.6 Processing Element Energy Estimation

This section presents the theory behind the processing element's energy estimation used in this thesis, based on the work of [M<sup>+</sup>18]. Estimating power and energy relies on a calibration process to define the energy/power values. The characterization flow employs the synthesizable VHDL description of the reference platform. Here, only the processor, router, and memory are considered; the NI power consumption is neglected because it is a small module compared to the others.

### 4.6.1 Processor Characterization

The characterization of the processor's energy and power relies upon the classes of instructions utilized by said processor, obtaining measures of energy per instruction and power per instruction for all classes and presenting these values in parcels of leakage and dynamic power. Table 4.2 shows which instruction classes were created and how many cycles per instruction are used by the class to execute a program based on RTL simulations.



Table 4.2: RTL simulation results obtained from instruction set classes. Adapted from [M<sup>+</sup>18].

Class	Cycles per instruction
arithmetic	1
logical	1
shift	1
move	1
nop	1
branches	1
jumps	1
load-store	2

The method to estimate power and energy is general because it is based on a calibration process to define the energy/power values. Therefore, [M<sup>+</sup>18] provides accurate and reliable measurement of dynamic and static power based on the switching activity annotation from a netlist simulation. The results of this can be found in Table 4.3.

Table 4.3: Power characterization results and energy estimation for each instruction class of the processor (65nm), at 1.1V, 25°C (T=4ns). Adapted from [M<sup>+</sup>18].

Class	Avg. Power (mW)		Energy per inst. (pJ)	
	Leakage	Dynamic	Leakage	Dynamic
arithmetic	0.452	5.894	1.808	23.58
logical		5.176		20.70
shift		4.940		19.76
move		4.768		19.07
nop		3.331		13.32
branches		5.723		31.70
jumps		4.175		18.56
load-store		5.507		3.616

The energy for a given class of instructions is given by Equation 4.1.

$$E_{class} = P_{class} \times CPI \times T \quad (4.1)$$

where  $P_{class}$  is the average power for a given instruction class, CPI is the number of cycles per instruction, and T is the clock period. Equation 4.1 works for both static and dynamic power.

From equation 4.1, the total energy consumption and the power dissipation can be estimated for the processor by equations 4.2 and 4.3.

$$E_{processor} = \sum_{i=0}^{n_{class}} n_{instruction_i} \times E_{class_i} \quad (4.2)$$

$$P_{processor} = \sum_{i=0}^{n_{class}} n_{instruction_i} \times P_{class_i} \quad (4.3)$$

#### 4.6.2 Router Characterization

The characterization process of the router is similar to the processor characterization; Table 4.4 presents the power characterization of the router for the two traffic rates. The second column presents the dynamic average power consumption for one buffer. The third column, combinational logic, corresponds to the remaining parts of the 5-port router ( $P_{leak_{router}}(n_{ports})$ ). The last table column presents the router leakage power [M<sup>+</sup>18].

Table 4.4: Router Average Power, at 1.1V, 25°C (T=4ns). Adapted from [M<sup>+</sup>18].

Traffic Rate	One buffer	Combinational Logic	$P_{leak_{router}}(n_{ports} = 5)$
0% - idle	364.64 $\mu$ W	575.64 $\mu$ W	223.08 $\mu$ W
100% - active	755.56 $\mu$ W	2655.25 $\mu$ W	

Equations 4.4 and 4.5 show the dynamic active energy to receive one flit and the spent dynamic energy in idle mode, respectively.

$$E_{active} = [(n_{ports} - 1) \times P_{buffer}^{idle} + P_{buffer}^{active} + P_{comb}^{active}] \times T \quad (4.4)$$

$$E_{idle} = [(n_{ports}) \times P_{buffer}^{idle} + P_{comb}^{idle}] \times T \quad (4.5)$$

where  $n_{ports}$  is the number of ports of the router,  $P_{component}^{idle}$  is the average idle power of a given component,  $P_{component}^{active}$  is the average active power, and T is the period used to characterize the router.

#### 4.6.3 Memory Characterization

For the memory energy characterization, [M<sup>+</sup>18] uses a memory generator tool called CACTI-P [LGA<sup>+</sup>11]. Table 4.5 presents the characterization data produced by said tool. Where  $P_{leak_{memory}}$  is the leakage power,  $E_{load}$  is the dynamic read energy per access, and  $E_{store}$  is the dynamic write energy per access.

Therefore, we can define the computation of power and energy of the PE ( $E_{PE}$ ) into the epoch, with  $E_{proc}$  the dynamic processor energy,  $E_{mem}$  the dynamic read/write energy per access,  $E_{router}$  the dynamic router energy, and  $E_{leak}$  the energy from leakage power of the whole PE (see eq. 4.6).

Table 4.5: CACTI-P Report for a Scratchpad Memory, at 1.1V, 25°C (T=4ns). Adapted from [M<sup>+</sup>18].

Access time	$P_{leak_{memory}}$	$E_{load}$	$E_{store}$
3.98ns	0.66mW	67pJ	38pJ

$$\left\{ \begin{array}{l} E_{proc} = \sum_{i=0}^{n_{classes}} n_{instructions_i} \times E_{dyn_{class_i}} \\ E_{mem} = n_{instructions_{load}} \times E_{load} + n_{instructions_{store}} \times E_{store} \\ E_{router} = E_{idle_{router}} \times (cycles_{total} - cycles_{active}) + E_{active_{router}} \times cycles_{active} \\ E_{leak} = [P_{leak_{proc}} + P_{leak_{mem}} + P_{leak_{router}}(n_{ports})] \times cycles_{total} \times T \\ E_{PE} = E_{proc} + E_{mem} + E_{router} + E_{leak} \end{array} \right. \quad (4.6)$$

#### 4.6.4 ORCA Monitoring

To apply the energy estimation methods presented in Sections 4.6.1, 4.6.2 and 4.6.3, the author utilize the hardware counters implemented in the ORCA MPSoC. These counters are exposed to the application level through the system API, which accesses reserved memory space to read from these counters via memory-mapped I/O. Counters appear to the rest of the system as a memory region, accessible through software. Reads to that memory region return the current value stored in the counter, and software can write to these counters to modify their value to zero, corresponding to a reset to the counter [D<sup>+</sup>20]. Table 4.6 shows the counters used in the experiments of this thesis.

Table 4.6: Counters available for energy estimation in ORCA. Adapted from [D<sup>+</sup>20].

Counter Name	Description
M0_COUNTER_STORE	Returns the number of writes on main memory since the last reset
M0_COUNTER_LOAD	Returns the number of reading on main memory since the last reset
M1_COUNTER_STORE	Returns the number of writes on receiving memory since the last reset
M1_COUNTER_LOAD	Returns the number of reading on receiving memory since the last reset
M2_COUNTER_STORE	Returns the number of writes on sending memory since the last reset
M2_COUNTER_LOAD	Returns the number of reading on sending memory since the last reset

Continued on next page

Table 4.6 – Continued from previous page

Counter Name	Description
CPU_COUNTER_ARITH	Returns the number of arithmetic instructions that the CPU of the tile, executed since the last reset
CPU_COUNTER_LOGICAL	Returns the number of logical instructions that the CPU of the tile, executed since the last reset
CPU_COUNTER_SHIFT	Returns the number of shift instructions that the CPU of the tile, executed since the last reset
CPU_COUNTER_BRANCHES	Returns the number of branch instructions that the CPU of the tile, executed since the last reset
CPU_COUNTER_JUMPS	Returns the number of jump instructions that the CPU of the tile, executed since the last reset
CPU_COUNTER_LOADSTORE	Returns the number of load and store instructions that the CPU of the tile, executed since the last reset
CPU_COUNTER_CYCLES_TOTAL	Returns the number of cycles in that the CPU was not stalled since the last reset
CPU_COUNTER_CYCLES_STALL	Returns the number of cycles in that the CPU was stalled since the last reset
ROUTER_COUNTER_ACTIVE	Returns the number of cycles that the router had being active since the last reset. Routers are active when at least one port is transmitting or receiveing data.

## 5. FRAMEWORK IMPLEMENTATION AND APPLICATIONS

This Chapter, defines the architecture used in this thesis. We explain how the MPSoC platform interacts with the quadrotor simulation, detailing the operating system used to simulate the quadrotor and how it is used to communicate with the simulation of the MPSoC. We explain how the experimental setup was built and how the control algorithms were developed and implemented on the system.

### 5.1 Target Architecture

The system's architecture proposed in this thesis can be seen in Figure 5.1 and been a simulation-only project, the entirety of the system run in a Linux OS. From there, we have the system divided into two main parts: the ROS system and the URSA simulator. The ROS system is responsible for simulating the quadrotor and communicating with the ORCA MPSoC. The URSA creates a simulation of an MPSoC running the control system of the quadrotor within its processing cores.

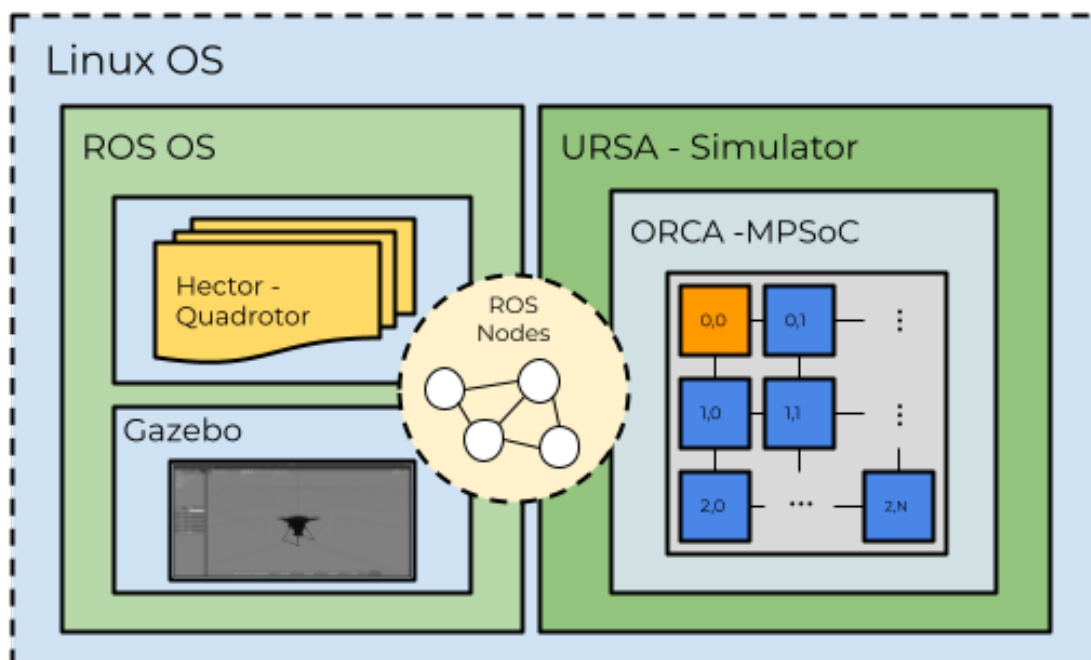


Figure 5.1: Overall representation of the proposed system.

### 5.1.1 Experimental Setup

In order to implement this system, we decide to break apart the architecture into two different machines. This choice was made to maximize the processing time due to the heavy computation demand by Hector quadrotor and URSA simulations. Figure 5.2 shows an overview of this implementation.



Figure 5.2: Experimental Setup.

Therefore, to better explain this setup, we need to understand running multiple machines in the ROS environment. In a system that relies upon multiple PCs, all of them must be configured to run ROS. Then the following rules must be applied:

- All machines (master and clients) must be in the same network;
- In all systems with multiple machines only a single PC running as a server (master), but can exist several clients;
- All machines recognize the master through a setting called `ROS_MASTER_URI`, which tells nodes where they can locate the server;
- All machines can run multiple nodes.

Then, in the implementation of the thesis was used two computers. The first machine (master) runs the URSA simulation and the ROS communication nodes. Moreover, the other (client) runs the hector quadrotor simulation. The specifications of each machine is described in Table 5.1.

Table 5.1: Computational setup specifications.

PC	OS	Processor	Memory	ROS Distro (Version)
ROS Master	Ubuntu 18.04.6 LTS	Intel® Core™ i7-10700F CPU @ 2.90GHz × 16	16GB	Melodic (1.14.13)
ROS Client	Ubuntu 18.04.5 LTS	Intel® Core™ i7-5500F CPU @ 2.40GHz × 4	6GB	Melodic (1.14.10)

## 5.2 ROS - Ursa Interface

In order to understand how the Hector simulation can communicate with ORCA, we need to expand the concepts briefly presented in Section 4.4. In Figure 4.4, we have seen the structure of nodes in the ORCA MPSoC, with network and processing nodes. According to [DJSFA19], networking nodes include hardware from the processing node. However, they have a network bridge attached to the NI instead of a processor core and main memory. The network bridge translates raw UDP packets (containing UDP/IP headers and beyond) to the protocol of the NoC, adding the required headers. Translation occurs when transmitting data from the NoC to the UDP network and vice-versa.

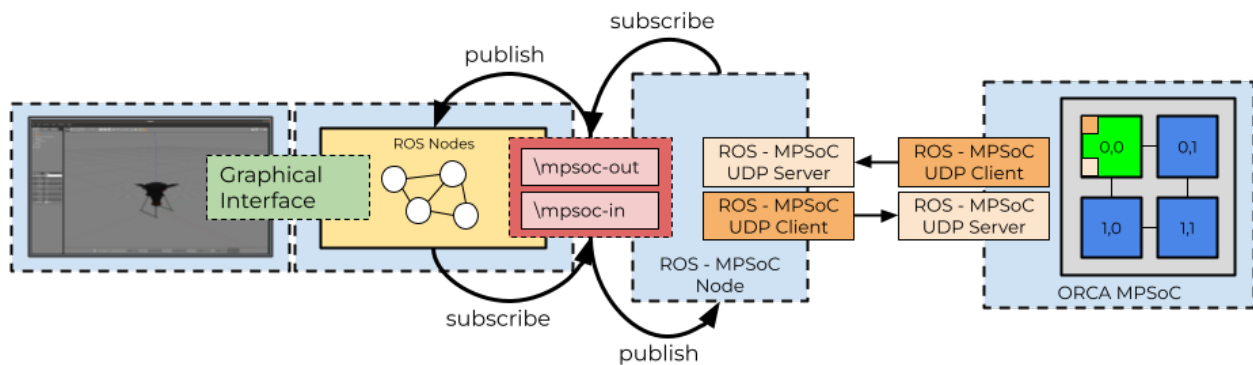


Figure 5.3: Data-flow of the MPSoC integration into a ROS system.

As shown in Figure 5.3, a ROS node was developed to create a communication line between ORCA and ROS. This node wraps the communication with the MPSoC and exposes the MPSoC as a resource to the whole system via topics. This node executes two simultaneous threads. The first treats the messages from the ROS systems to the MPSoC, and the second treats UDP packets from the MPSoC.

## 5.3 Implementation of the Algorithms

This section describes how the algorithms of Section 5.5 are implemented in the ORCA platform. Based on the Figure 4.3, the folder HellfireOS shows its components. Folder "*libc*" houses the algorithms, each one described as a .C file.

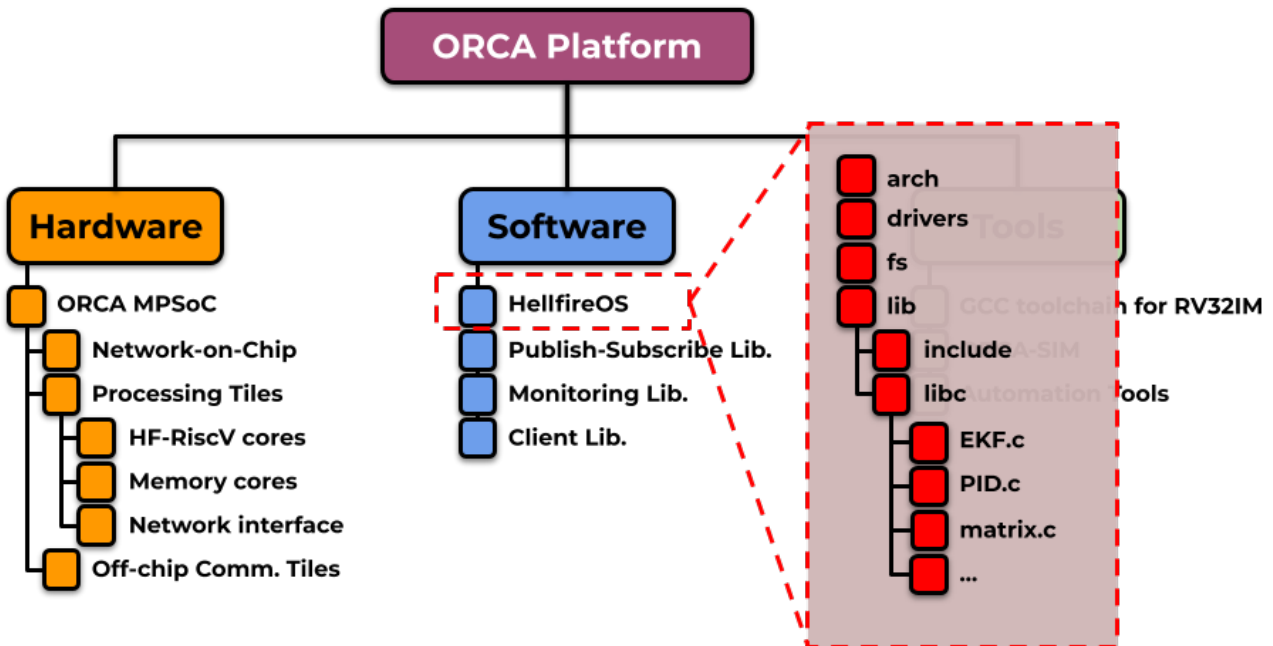


Figure 5.4: Algorithm implementation within the ORCA platform.

In order to build and assign the algorithms to each processing node of the MP-SoC, first, we need to create a "*applications*" folder in the software partition of the system. There, we create all applications needed for the project, also as a .C file with its header and makefile.

Figure 5.5 shows how the EKF application was created for this project.

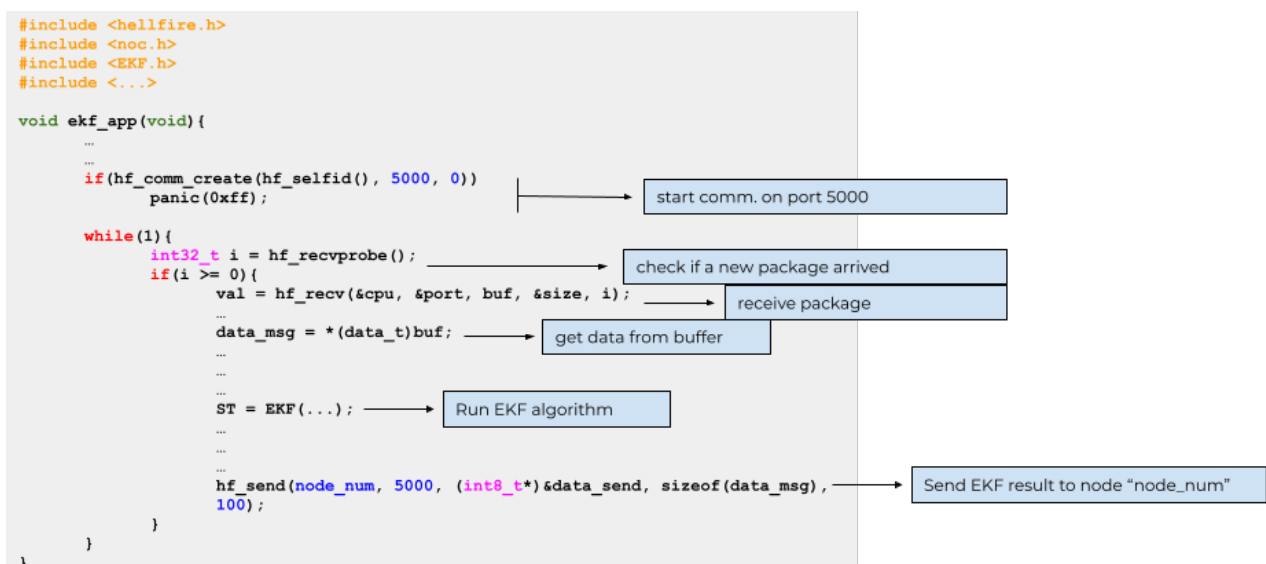


Figure 5.5: Application implementation of the EKF algorithm.

The next step is configuring the ORCA settings; this is done by creating a "Configuration.mk" file in the ROOT partition of the project. This file is populated by all the values that ORCA needs to run every simulation. Values like the number of cores, how they are



placed in the grid, which application runs in what core, and information related to topics like the URSA engine, netsocket, buffer, memory, and more.

## 5.4 C and C++ Language Libraries

This section presents a few C and C++ libraries developed that serve as the backbone of this project. They are a small matrix operations library and a fixed-point arithmetic library—the items below detail how each was implemented.

- **matrix.h** → *Small Custom Matrix Operations Library*: For the sake of memory usage was developed a small matrix operation library in C, containing the exact amount of mathematical operations needed to make the system work, abiding by the maximum and minimum of matrices sizes that the *EKF* uses in this context. Each matrix is described as a *struct* containing two integers for column and row sizes and a vector for the matrices values. Next, is enumerated the functions present in this library.

- SET VALUES: This function is responsible for aggregating vital information about the matrix. As stated before, each matrix is described as a *struct* containing two integers for column and row sizes and a vector for the matrices values. As seen in equations 5.1 and 5.2, the values in the vector are used to populate the matrix. The first values are disposed in the matrix from left to right, then row by row. This function is used throughout the EKF task.

\* inputs:

$$\left\{ \begin{array}{l} n_{rows} \\ n_{columns} \\ \mathit{vector} = [v_0, v_1, v_2, \dots, v_{k-1}, v_k] \end{array} \right. \quad (5.1)$$

\* output:

$$A = \begin{bmatrix} v_0 & v_1 & v_2 & \dots \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \dots & v_{k-1} & v_k & \end{bmatrix} \quad (5.2)$$

- TRANSPOSED MATRIX: The transposed function receives a Matrix *struct* *A* and returns a new *struct* *A* with the values of rows and columns swapped. This function is used in lines 2 and 3 of Algorithm 3.2 and in equation 3.13.

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \quad A^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \quad (5.3)$$

- **SUM OF MATRICES:** Straightforward sum operation, where the function receives two matrices,  $A$  and  $B$ , then returns the *struct* of a new matrix  $C$  with the sums of each of the individual elements of the two matrices. Examples of use of this functions can be found in equations 3.11 and 3.14.
- **SUBTRACTION OF MATRICES:** Another straightforward operation, where the function receives two matrices,  $A$  and  $B$ , then returns the *struct* of a new matrix  $C$  with the subtraction of each of the individual elements of the two matrices. This function is used in lines 4 and 5 of Algorithm 3.2
- **MULTIPLICATION OF MATRICES:** In this operation, the function receives two matrices,  $A$  and  $B$ , then returns the *struct* of a new matrix  $C$  with the of each corresponding multiplication of row by column of each of the elements of the two matrices. This function is used throughout the code.
- **MATRIX MULTIPLICATION BY A SCALAR:** In this operation, the function receives matrix  $A$  and a scalar  $x$ , then returns the *struct* of a new matrix  $B$  that contains the elements of the matrix  $A$  multiplied by  $x$ . This function is used throughout the code.

$$A = \begin{bmatrix} a_{(0,0)} & a_{(0,1)} & a_{(0,2)} \\ a_{(1,0)} & a_{(1,1)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,1)} & a_{(2,2)} \end{bmatrix}, \quad x \forall x \in \mathbb{R}, \quad B = \begin{bmatrix} xb_{(0,0)} & xb_{(0,1)} & xb_{(0,2)} \\ xb_{(1,0)} & xb_{(1,1)} & xb_{(1,2)} \\ xb_{(2,0)} & xb_{(2,1)} & xb_{(2,2)} \end{bmatrix}, \quad Ax = B \quad (5.4)$$

- **MATRIX DIVISION BY A SCALAR:** This operation is similar to the prior function, it receives matrix  $A$ , and a scalar  $x$  then returns the *struct* of a new matrix  $B$  that contains the elements of the matrix  $A$  divided by  $x$ .

$$A = \begin{bmatrix} a_{(0,0)} & a_{(0,1)} & a_{(0,2)} \\ a_{(1,0)} & a_{(1,1)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,1)} & a_{(2,2)} \end{bmatrix}, \quad x \forall x \in \mathbb{R}, \quad B = \begin{bmatrix} \frac{a_{(0,0)}}{x} & \frac{a_{(0,1)}}{x} & \frac{a_{(0,2)}}{x} \\ \frac{a_{(1,0)}}{x} & \frac{a_{(1,1)}}{x} & \frac{a_{(1,2)}}{x} \\ \frac{a_{(2,0)}}{x} & \frac{a_{(2,1)}}{x} & \frac{a_{(2,2)}}{x} \end{bmatrix}, \quad A/x = B \quad (5.5)$$

- **MATRIX SUM BY A SCALAR:** This operation is similar to the prior functions, it receives matrix  $A$ , and a scalar  $x$  then returns the *struct* of a new matrix  $B$  that contains the elements of the matrix  $A$  summed by  $x$ .

$$A = \begin{bmatrix} a_{(0,0)} & a_{(0,1)} & a_{(0,2)} \\ a_{(1,0)} & a_{(1,1)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,1)} & a_{(2,2)} \end{bmatrix}, \quad x \forall x \in \mathbb{R}, \quad B = \begin{bmatrix} a_{(0,0)} + x & a_{(0,1)} + x & a_{(0,2)} + x \\ a_{(1,0)} + x & a_{(1,1)} + x & a_{(1,2)} + x \\ a_{(2,0)} + x & a_{(2,1)} + x & a_{(2,2)} + x \end{bmatrix}, \quad A+x = B \quad (5.6)$$

- **CREATE AN IDENTITY MATRIX:** Due to the frequent use of identity matrices in control theory, was created a function that receives a scalar  $x$  representing the

matrix order and creates a matrix in the form of equation 5.7.

$$I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix} \quad (5.7)$$

- CREATE A MATRIX OF ZEROS: For convenience, a function that receives two scalars (representing rows and columns) was created that determines the size of a populated by zeros matrix  $A$ .

$$A = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad (5.8)$$

- CREATE A MATRIX OF ONES: This function has the same principle as the prior function; the difference is that matrix  $A$  is populated by ones.

$$A = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ 1 & 1 & \ddots & 1 \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (5.9)$$

- CREATE A  $3 \times 3$  SKEW-SYMMETRIC MATRIX: In Section 3.6.3 is presented the quaternion representation theory. With it, it is shown the skew-symmetric matrix, to create a more straightforward user interface, was created a function that receives the three values  $x$ ,  $y$ , and  $z$  and builds a  $3 \times 3$  skew-symmetric matrix, as presented in equation 5.10.

$$A = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}, \quad \forall xyz \in \mathbb{R} \quad (5.10)$$

- BLOCK DIAGONAL CONCATENATION: This topic comprises three functions designed to create a matrix with the input of other matrices. In this case, the concatenation of different matrices along the diagonal of the resultant matrix, as seen in equation 5.11. As said previously, three functions were created utilizing this idea, varying in the number of input matrices, two, three, or four. The rest of the resultant matrix is filled with zeros. In equations, 5.12, 5.13 and 5.14 is shown an

example of how this method is called, for an input of two matrices and its result.

$$M = \begin{bmatrix} A & 0 & 0 & \dots & 0 \\ 0 & B & 0 & \dots & 0 \\ 0 & 0 & C & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \quad (5.11)$$

Example:

$$M = \text{blkdiag2}(A, B) \quad (5.12)$$

where,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \text{and} \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \quad (5.13)$$

then,

$$M = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & 0_{2 \times 2} \\ 0_{2 \times 2} & \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \end{bmatrix}, \quad \text{where } 0_{2 \times 2} \text{ is } \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (5.14)$$

- **CUSTOM MATRIX OF MATRICES:** This function also aims the creation of a matrix using the input of other matrices. Contrary to the prior task, which was a restriction on the placement of matrices, this one lets the user position each of the input matrices wherever he wants, as shown in equation 5.15. The user must inform the system of the number of input matrices, a configuration vector containing the information on matrix placement, and all the matrices that build the new one.

$$M = \begin{bmatrix} A & B & \dots \\ C & D & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad (5.15)$$

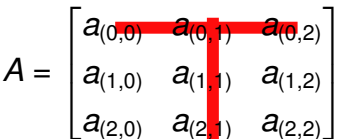
The function is called by the line: "customMat(*n\_mats*, *vet\_conf*, A,B,C,D..)", where, *vet\_conf* = [confX, confY, row, column], confY and confX indicate how the matrix is built. In case of confX = 2 and confY = 2, the matrices A, B, C and D are placed in a 2x2 grid. The parameters row and column indicates the total size of the matrix.

- **DELETE ROW AND COLUMN OF A MATRIX:** The A.14 algorithm shows the method for obtaining a matrix, eliminating the row and column of a particular element which is in the form of a square or rectangle. This algorithm is later used in the determinant function (alg.A.15) to calculate co-factors.

Take matrix  $A$ :

$$A = \begin{bmatrix} a_{(0,0)} & a_{(0,1)} & a_{(0,2)} \\ a_{(1,0)} & a_{(1,1)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,1)} & a_{(2,2)} \end{bmatrix} \quad (5.16)$$

For the element  $a_{(0,1)}$  the first row and second column are not considered as seen in Figure 5.4.

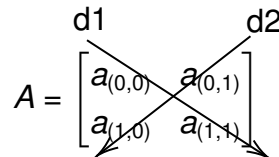
$$A = \begin{bmatrix} a_{(0,0)} & a_{(0,1)} & a_{(0,2)} \\ a_{(1,0)} & a_{(1,1)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,1)} & a_{(2,2)} \end{bmatrix}$$


Therefore, the resulting matrix is:

$$A_{a_{(0,1)}} = \begin{bmatrix} a_{(1,0)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,2)} \end{bmatrix} \quad (5.17)$$

– DETERMINANT OF A MATRIX: To calculate the determinant of a square matrix, the Algorithm A.15 is divided into three cases:

- \* Matrix  $(1 \times 1)$ : The determinant is equal the only element in the matrix;
- \* Matrix  $(2 \times 2)$ : The determinant is equal to the product of diagonal 1 ( $d1$ ) minus the product of diagonal 2 ( $d2$ ):

$$A = \begin{bmatrix} a_{(0,0)} & a_{(0,1)} \\ a_{(1,0)} & a_{(1,1)} \end{bmatrix}$$


- \* Matrix  $(n \times n) \in n > 2$ : For each element of first row or first column get cofactor (alg. A.14) of those elements and then multiply the element with the determinant of the corresponding cofactor, and finally add them with alternate signs.

Take the matrix  $A$  with order 3:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad (5.18)$$

Then, the determinant is:

$$\det(A) = a(ei - fh) - b(di - gf) + c(dh - eg) \quad (5.19)$$

To illustrate this in terms of cofactor matrices:

$$\begin{bmatrix} a & & \\ & \times & \\ & & \begin{vmatrix} e & f \\ h & i \end{vmatrix} \end{bmatrix} - \begin{bmatrix} & b & \\ & \times & \\ \begin{vmatrix} d & f \\ g & i \end{vmatrix} & & \end{bmatrix} + \begin{bmatrix} & & c \\ & & \times \\ \begin{vmatrix} d & e \\ g & h \end{vmatrix} & & \end{bmatrix} \quad (5.20)$$

- INVERSE OF A MATRIX: To calculate the inverse of a matrix, the Algorithm A.16 computes the following equation:

$$A^{-1} = \frac{1}{\det(A)} \text{adjoint}(A) \quad (5.21)$$

- COPY: The Algorithm A.17 takes an input matrix  $A$  and creates an output matrix  $B$  with the same size and values.
- RANK OF A MATRIX: The Algorithm A.18 calculates the dimension of the subspace spanned by the rows of a matrix. The dimension of the column space is equal to the rank.
- CHECK SYMMETRY OF A MATRIX: To check the symmetry of a matrix, the following preposition must be obeyed:

$$A \text{ is symmetric} \Leftrightarrow A = A^T \quad (5.22)$$

The output of this function is either *True* or *False*.

- LU DECOMPOSITION: This function (Algorithm A.20) is responsible for decomposing a matrix into lower and upper triangular matrices:

$$A = LU \quad (5.23)$$

$$\begin{bmatrix} a_{(0,0)} & a_{(0,1)} & a_{(0,2)} \\ a_{(1,0)} & a_{(1,1)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,1)} & a_{(2,2)} \end{bmatrix} = \begin{bmatrix} l_{(0,0)} & 0 & 0 \\ l_{(1,0)} & l_{(1,1)} & 0 \\ l_{(2,0)} & l_{(2,1)} & l_{(2,2)} \end{bmatrix} \begin{bmatrix} u_{(0,0)} & u_{(0,1)} & u_{(0,2)} \\ 0 & u_{(1,1)} & u_{(1,2)} \\ 0 & 0 & u_{(2,2)} \end{bmatrix} \quad (5.24)$$

- GET A MATRIX DIAGONAL: This function retrieves the values of the diagonal of a square matrix and returns a vector with the values mentioned above:

$$\begin{bmatrix} a_{(0,0)} & a_{(0,1)} & a_{(0,2)} \\ a_{(1,0)} & a_{(1,1)} & a_{(1,2)} \\ a_{(2,0)} & a_{(2,1)} & a_{(2,2)} \end{bmatrix} \rightarrow \begin{bmatrix} a_{(0,0)} \\ a_{(1,1)} \\ a_{(2,2)} \end{bmatrix} \quad (5.25)$$

- GET A SECTION OF A MATRIX: This function is responsible for creating a matrix  $B_{(i,j)}$  based on a desired section of an original matrix  $A_{(i,j)}$ . The Algorithm A.13 receives four parameters:  $R_i$ , the first position of the row to be copied,  $R_f$ , the last

position of the row to be copied,  $C_i$ , the first position of the column to be copied,  $C_f$ , and the last position of the column to be copied.

$$A(i, j) = \begin{bmatrix} a(o, o) & a(o, 1) & a(o, 2) & a(o, 3) & a(o, 4) \\ a(1, o) & a(1, 1) & a(1, 2) & a(1, 3) & a(1, 4) \\ a(2, o) & a(2, 1) & a(2, 2) & a(2, 3) & a(2, 4) \\ a(3, o) & a(3, 1) & a(3, 2) & a(3, 3) & a(3, 4) \\ a(4, o) & a(4, 1) & a(4, 2) & a(4, 3) & a(4, 4) \end{bmatrix}$$

$B(i, j)$

$C_i$                        $C_f$

$R_i$                        $R_f$

- QR DECOMPOSITION: The Algorithm A.23 receives a matrix  $A$  and outputs the decomposition of the matrix into matrix  $Q$ , where  $Q$  is an orthogonal matrix, and  $R$  is an upper triangular matrix. Used in Algorithm A.27

$$A = QR \quad (5.26)$$

- MATRIX TO THE POWER OF  $N$ : The Algorithm A.24 receives matrix  $A$  and scalar  $N$  and return matrix  $B$  that is equal to  $A$  to the power of  $A$ .

$$B = A^N \quad (5.27)$$

- TRACE OF A MATRIX: Auxiliary function. The Algorithm A.25 receives a matrix  $A$  and outputs the sum of the values from its diagonal. It is used in Algorithm A.26.

$$tr(A) = \sum_{k=0}^K A_{(kk)} \quad (5.28)$$

- EIGENVALUES OF A MATRIX ( $2 \times 2$ ): Auxiliary function. The Algorithm A.26 receives a matrix  $A$  with size  $2 \times 2$  and outputs its eigenvalues. It is used in Algorithm A.27.
- EIGENVALUES OF A MATRIX: The Algorithm A.27 receives a matrix  $A$  with sizes higher than  $2 \times 2$  and outputs its eigenvalues.
- ZEROS ALL THE ELEMENTS BELOW THE MATRIX DIAGONAL: Auxiliary function. It receives a matrix  $A$  and returns a matrix  $B$  with the values below the diagonal zeroed.

$$A = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}, \quad B = \begin{bmatrix} a & b & c & d \\ 0 & f & g & h \\ 0 & 0 & k & l \\ 0 & 0 & 0 & p \end{bmatrix} \quad (5.29)$$

- ZEROS ALL THE ELEMENTS ABOVE THE MATRIX DIAGONAL: Auxiliary function. It receives a matrix  $A$  and returns a matrix  $B$  with the values above the diagonal zeroed.

$$A = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}, \quad B = \begin{bmatrix} a & 0 & 0 & 0 \\ e & f & 0 & 0 \\ i & j & k & 0 \\ m & n & o & p \end{bmatrix} \quad (5.30)$$

- 2-NORM OF A MATRIX: The Algorithm A.30 computes the norm of all the matrix  $A$  values:

$$\text{norm}(A_{(i,j)}) = \sqrt{\sum \|a_{(i,j)}^2\|} \quad (5.31)$$

- DISCRETE-TIME ALGEBRAIC RICCATI EQUATION SOLVER: The Algorithm A.31 is responsible to find a symmetric solution, that converge quadratically to a stable state of the equation  $X = A^T X A - (B^T X A)^T (R + B^T X B)^{-1} B^T X A + Q$  (adapted from [CFLW04]).
- **fixed\_point.h** → *Fixed-point arithmetic library*: To utilize an alternative for floating-point arithmetic that put unnecessary stress on the computation, a fixed-point arithmetic library in C was developed to represent fractional (non-integer) numbers by storing a fixed number of digits of their fractional part. Each fixed-point variable is an signed integer of 32 bits (`int32_t`). Table 5.2 shows a summary of the functions used in this library.

Table 5.2: Fixed-point functions used in this Thesis.

Function	Description
<code>fix_add(A, B)</code>	sum of two numbers
<code>fix_sub(A, B)</code>	subtraction of two numbers
<code>fix_mul(A, B)</code>	multiplication of two numbers
<code>fix_div(A, B)</code>	division of two numbers
<code>fixtoa(A, char, dec)</code>	convert fixed-point number to char
<code>fix_sqrt(A)</code>	square root of a number
<code>fix_exp(A)</code>	exponential function
<code>fix_ln(A)</code>	natural logarithm of a number
<code>fix_log(A, base)</code>	logarithm of a number in a certain base
<code>fix_pow(A, exp)</code>	exponential of a number
<code>fix_rad(deg)</code>	convert degree to radians
<code>fix_sin(rad)</code>	sine of a angle (in radians)

Continued on next page



Table 5.2 – Continued from previous page

Function	Description
<code>fix_cos(rad)</code>	cosine of a angle (in radians)
<code>fix_tan(rad)</code>	tangent of a angle (in radians)
<code>fix_atan(rad)</code>	inverse tangent of a angle (in radians)
<code>fix_atan2(rad1, rad2)</code>	2-argument arctangent (in radians)
<code>fix_asin(rad)</code>	inverse sine of a angle (in radians)
<code>fix_acos(rad)</code>	inverse cosine of a angle (in radians)
<code>fix_sinh(rad)</code>	hyperbolic sine of a angle (in radians)
<code>fix_cosh(rad)</code>	hyperbolic cosine of a angle (in radians)
<code>fix_tanh(rad)</code>	hyperbolic tangent of a angle (in radians)
<code>fix_print(A)</code>	print a fixed-point number
<code>fix_sign(A)</code>	function to show signal from a fixed-point number: Return 1 if $A \geq 0$ , and return -1 if $A < 0$
<code>fix_norm(vector)</code>	euclidian norm of a vector
<code>fix_cpsign(A, B)</code>	copy sign of a number (A) into an another number (B)

## 5.5 Description of the Algorithms

This Section approaches the algorithms implemented based on concepts that were already referenced in chapter 3 and other ideas that are presented next. Every algorithm here described used the software libraries described in Section 5.4, demonstrating the wide range of control applications that can be embedded in this system.

### 5.5.1 Attitude Measurement of the Quadrotor based on Accelerometer and Magnetometer

In the EKF algorithm presented in 5.5.2, the "correction" portion of the algorithm is calculated with the quaternion measurement from the accelerometer and magnetometer. Equation 5.32 show the calculations used to estimate *roll*, *pitch* and *yaw*, based in the readings of the accelerometer and magnetometer,

$$\left\{ \begin{array}{l} \text{roll} = \tan^{-1}\left(\frac{ay}{\sqrt{ax^2+az^2}}\right) \\ \text{pitch} = \tan^{-1}\left(\frac{-ax}{\sqrt{ay^2+az^2}}\right) \\ \text{yaw} = \tan^{-1}\left(\frac{-my \cdot \cos(\text{roll}) + mz \cdot \cos(\text{roll})}{mx \cdot \cos(\text{pitch}) + my \cdot \sin(\text{pitch}) \cdot \sin(\text{roll}) + mz \cdot \sin(\text{pitch}) \cdot \cos(\text{roll})}\right) \end{array} \right. \quad (5.32)$$

to transform the estimation *roll*, *pitch* and *yaw* to quaternion, equation 5.33 is used.

$$\begin{cases} qw = \cos\left(\frac{pitch}{2}\right) \cdot \cos\left(\frac{roll}{2}\right) \cdot \cos\left(\frac{yaw}{2}\right) + \sin\left(\frac{pitch}{2}\right) \cdot \sin\left(\frac{roll}{2}\right) \cdot \sin\left(\frac{yaw}{2}\right) \\ qx = \sin\left(\frac{pitch}{2}\right) \cdot \cos\left(\frac{roll}{2}\right) \cdot \cos\left(\frac{yaw}{2}\right) + \cos\left(\frac{pitch}{2}\right) \cdot \sin\left(\frac{roll}{2}\right) \cdot \sin\left(\frac{yaw}{2}\right) \\ qy = \cos\left(\frac{pitch}{2}\right) \cdot \sin\left(\frac{roll}{2}\right) \cdot \cos\left(\frac{yaw}{2}\right) + \sin\left(\frac{pitch}{2}\right) \cdot \cos\left(\frac{roll}{2}\right) \cdot \sin\left(\frac{yaw}{2}\right) \\ qz = \cos\left(\frac{pitch}{2}\right) \cdot \cos\left(\frac{roll}{2}\right) \cdot \sin\left(\frac{yaw}{2}\right) + \sin\left(\frac{pitch}{2}\right) \cdot \sin\left(\frac{roll}{2}\right) \cdot \cos\left(\frac{yaw}{2}\right) \end{cases} \quad (5.33)$$

Where the inputs are the vectors:  $a = [a_x, a_y, a_z]$ , representing the accelerometer readings in the  $x, y$  and  $z$  axis. Furthermore,  $m = [m_x, m_y, m_z]$  represents the magnetometer readings in the  $x, y$ , and  $z$  axis. In lines 1 and 2, the accelerometer readings are used to approximate the angles of *roll* and *pitch*. Because the quadrotor only operate in the stable conditions, where  $roll \approx pitch \approx yaw \approx 0$ . The calculations accounting *gimbal lock* are not considered. In line 3, the yaw angle is calculated using the magnetometer readings with the *roll* and *pitch* findings. Finally, the Euler angles are converted to quaternions on lines 5, 6, 7, and 8.

### 5.5.2 Extended Kalman Filter for Attitude Estimation

To understand the principles of the attitude estimation by the extended Kalman filter first is necessary to understand how the quadrotor's attitude is represented. As showed in Figure 5.6 attitude is define as the orientation of the quadrotor in the body frame ( $B$ ) in relation to the navigation frame ( $N$ ), usually represented by Euler angles *roll* ( $\phi$ , rotation around the X-axis), *pitch* ( $\theta$ , rotation around the Y-axis) and *yaw* ( $\psi$ , rotation around the Z-axis).

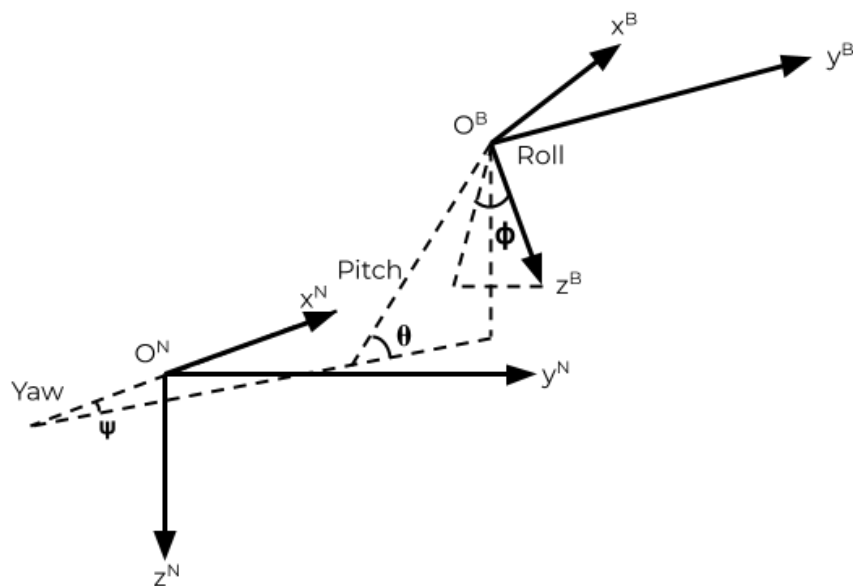


Figure 5.6: Euler angles representation. Adapted from [JCH<sup>+</sup>17].

A simple way to represent this rotation is by quaternions, that are represented by

$$q = \begin{bmatrix} q_0 \\ q_v \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \quad (5.34)$$

And to represent the angular rate  $\omega$  we need to get the quaternion's derivative

$$\dot{q} = \Omega(\omega)q \quad (5.35)$$

where

$$\Omega(\omega) = \frac{1}{2} \begin{bmatrix} 0 & -\omega^T \\ \omega & [\omega_s] \end{bmatrix} \quad (5.36)$$

and

$$[\omega_s] = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix}. \quad (5.37)$$

That way,

$$\Omega(\omega) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \quad (5.38)$$

In this algorithm, the angular rate  $\omega$  can be measured by a gyroscope sensor in the body frame, and represented by

$$\omega = [\omega_x \quad \omega_y \quad \omega_z]^T \quad (5.39)$$

The discrete-time form of the system process model can be described as

$$q_{k+1} = \exp(\Omega_k \Delta_t) q_k + w_k, \quad k = 0, 1, 2, \dots \quad (5.40)$$

where  $\Delta_t$  represents the system sample interval and  $w_k$  is the process noise. To linearize the  $\exp(\Omega_k \Delta_t) q_k$  portion of the equation 5.40, we use its first-order and second-order items of Taylor series expansion, that gives:

$$q_{k+1} = \underbrace{\left( I_{4 \times 4} + \frac{1}{2} \Omega_k \Delta t \right)}_F q_k \quad (5.41)$$

The observation vector of the model is given by:

$$Z_k = [q_0 \quad q_1 \quad q_2 \quad q_3]^T \quad (5.42)$$

And it is obtained by the Algorithm 5.1. Finally the applied algorithm of the EKF in this project is represented as:

Algorithm 5.1: Attitude Estimation EKF.

```

1 algorithm ATTITUDE_EKF ( $x, ACC, GYR, MAG, Q, R, P$ ) :
2   Set measured_quaternion using eq.5.32 and 5.33;
3    $w = [GYR]$ ;
4    $wt = [GYR]^T$ ;
5    $sw = SKEW\_MAT([GYR]) \rightarrow$  Algorithm A.11;
6   Set omega using eq. 5.36;
7    $F = EYE(4) + \omega$ ;
8    $X_k = F \cdot X_{k-1}$ 
9    $P_k = F \cdot P_{k-1} \cdot F^T + Q$ 
10   $K_k = P_{k-1} (P_{k-1} + R)^{-1}$ 
11   $X_k = X_k + K_k [Z_k - X_k]$ 
12   $P_k = (EYE(4) - K_k) P_k$ 
13 return ( $X_k, P_k$ )

```

### 5.5.3 Kalman Filter for Linear Position Estimation

In order to create an XY position estimation of the quadrotor, a Kalman Filter algorithm is implemented, aggregating the readings of the accelerometer and GPS.

Therefore, first, we determine the linear equations that describe the model associated with the motion of a body:

$$\begin{cases} Pos = pos_i + vel \Delta t + \frac{1}{2} acc \Delta t^2 \\ Vel = vel_i + acc \Delta t \end{cases} \quad (5.43)$$

Where  $Pos$  is the one-dimensional position of the body,  $Pos_i$  is the initial position,  $Vel$  is the velocity of the body,  $\Delta t$  is the variation of time, and  $acc$  is the acceleration. Assuming that the desired states  $X$  are position and velocity, we can describe the equations on system 5.43 as an space-state format:

$$\begin{bmatrix} Pos \\ Vel \\ x \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} PREVIOUS \\ STATES \\ previous\ states \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \\ B \end{bmatrix} [acc]_u \quad (5.44)$$

The system represented in equation 5.45 is known as a state equation, where matrix  $A$  is the state matrix and matrix  $B$  is the output matrix.

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ \begin{bmatrix} \dot{Pos} \\ \dot{Vel} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Pos \\ Vel \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} [acc] \end{cases} \quad (5.45)$$

The system represented in equation 5.46 is known as an output equation, where matrix  $C$  is the output matrix and matrix  $D$  is the feedthrough matrix.

$$\begin{cases} y(t) = Cx(t) + Du(t) \\ Pos = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} Pos \\ Vel \end{bmatrix} + [0]u(t) \end{cases} \quad (5.46)$$

To build the Kalman Filter, it is necessary to derive the state extrapolation equation (see eq. 5.47). This equation can predict the next system state, based on the knowledge of the current state.

$$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + Gu_{n,n} + \omega_n \quad (5.47)$$

where,  $\hat{x}_{n+1,n}$  is a predicted system state vector at time step  $n+1$ ,  $\hat{x}_{n,n}$  is an estimated system state vector at time step  $n$ ,  $u_n$  is a measurable input to the system,  $\omega_n$  is a process noise or disturbance,  $F$  is a state transition matrix and  $G$  is a control matrix.

In matrix form, we can define  $F$  and  $G$  as the equations in 5.48:

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \quad (5.48)$$

As the system controlled is the XY position of the quadrotor, we need to rewrite the equation 5.43 to comprise the entirety of the system.

$$x(t) = \begin{bmatrix} Pos_x \\ Pos_y \\ Vel_x \\ Vel_y \end{bmatrix}, \quad \begin{cases} Pos_x = Pos_{x_i} + vel_x \Delta t + \frac{1}{2} acc_x \Delta t^2 \\ Pos_y = Pos_{y_i} + vel_y \Delta t + \frac{1}{2} acc_y \Delta t^2 \\ Vel_x = vel_{x_i} + acc_x \Delta t \\ Vel_y = vel_{y_i} + acc_y \Delta t \end{cases} \quad (5.49)$$

Therefore, the matrix  $F$  is represented as:

$$\dot{x}(t) = \begin{bmatrix} \dot{Pos}_x \\ \dot{Pos}_y \\ \dot{Vel}_x \\ \dot{Vel}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Pos_{x_i} \\ Pos_{y_i} \\ vel_{x_i} \\ vel_{y_i} \end{bmatrix} \Rightarrow F \quad (5.50)$$

$A$   $x(t)$

And the matrix  $G$  is represented as:

$$\ddot{x}(t) = \begin{bmatrix} \ddot{Pos}_x \\ \ddot{Pos}_y \\ \ddot{Vel}_x \\ \ddot{Vel}_y \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} \Rightarrow G \quad (5.51)$$

$B$   $u(t)$

Based on the Algorithm 3.1 the applied algorithm of the KF in this project is represented as:

Algorithm 5.2: XY Position Estimation KF.

```

1 algorithm XY_POSITION_KF ( $x, ACC, GPS, Q, R, P$ ) :
2   Set  $F$  and  $G$  with eq. 5.48;
3    $X_k = F.X_{k-1} + G.u_{k-1}$ 
4    $P_k = F.P_{k-1}.F^T + Q$ 
5    $K_k = P_{k-1}.H^T.(H.P_{k-1}.H^T + R)^{-1}$ 
6    $X_k = X_k + K_k[Z_k - (H.X_k)]$ 
7    $P_k = (I - K_k.H)P_k$ 
8 return ( $X_k, P_k$ )

```

In the line 4 of the Algorithm 5.2 the Matrix  $H$  represents the observation matrix, and is shown in equation 5.52

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (5.52)$$

#### 5.5.4 Proportional, Integral and Derivative Control (PID) - Attitude and Height Control

As previously stated, the PID controller can be used to find the control signal ( $u_i$ ) for all of the states that guarantees the desired orientation (roll  $\phi_{sp}$ , pitch  $\theta_{sp}$ , yaw  $\psi_{sp}$ ) and height ( $z$ ) of the vehicle. The PID controller for the roll, pitch and yaw parameters are described in Equations 5.53, 5.54, 5.55 where  $K_p$  is the proportional gain,  $K_i$  is the integral gain,  $K_d$  is the derivative gain, and  $\phi_{sp}$  is the setpoint for the roll parameter. The same applies for pitch and yaw parameters. The height control also uses a modified PID controller [HHWT09], shown

in Equation 5.56, where  $T_{gravity}$  is the minimum thrust that overcomes the gravity force. The aforementioned control laws are used as components for the signal output for each of the four motors.

$$u_{\phi} = K_p(\phi_{sp} - \phi) + K_i \int_0^t (\phi_{sp} - \phi) dt + K_d \frac{d}{dt}(\phi_{sp} - \phi); \quad (5.53)$$

$$u_{\theta} = K_p(\theta_{sp} - \theta) + K_i \int_0^t (\theta_{sp} - \theta) dt + K_d \frac{d}{dt}(\theta_{sp} - \theta); \quad (5.54)$$

$$u_{\psi} = K_p(\psi_{sp} - \psi) + K_i \int_0^t (\psi_{sp} - \psi) dt + K_d \frac{d}{dt}(\psi_{sp} - \psi); \quad (5.55)$$

$$u_z = \frac{1}{\cos \phi \cos \theta} (K_{pz}(z_{sp} - z) + K_{iz} \int_0^t (z_{sp} - z) dt + K_{dz} \frac{d}{dt}(z_{sp} - z) + T_{gravity}); \quad (5.56)$$

Based on the modeling described in Section 3.2 and the equations 5.53, 5.54, 5.55 and 5.56, the control law for each individual rotor can be described in Equation 5.57.

$$\begin{aligned} u_1 &= -u_{\theta} + u_{\psi} + u_z & u_2 &= -u_{\phi} - u_{\psi} + u_z \\ u_3 &= u_{\theta} + u_{\psi} + u_z & u_4 &= u_{\phi} - u_{\psi} + u_z \end{aligned} \quad (5.57)$$

Algorithm 5.3: Generic PID used for the height and attitude parameters.

```

1 algorithm PID(input) :
2   error = setpoint - input
3   errorsum+ = error - antiwindup
4   derivative = error - lasterror
5   lasterror = error
6   u = Kd × derivative + Kp × error + Ki × errorsum
7   if (u < satlow) {
8     antiwindup = satlow
9   }
10  if (u > sathigh) {
11    antiwindup = sathigh
12  }
13  if (u ≤ satlow) && u ≥ sathigh) {
14    antiwindup = satlow
15  }
16 return (u1, u2, u3, u4)

```

### 5.5.5 Proportional, Integral and Derivative Control (PID) - XY Position Control

Quadrotors are an example of underactuated systems with six degrees of freedom that is larger than the number of independent control inputs. As a result, this underactuation limits the number of system configurations that can directly be controlled. The system cannot follow an unrestricted flight in full vector space due to the lack of adequate control actions in their configuration space. Hence, the dynamics model of the quadrotors is not fully linearizable [EN18]. That is, in a quadrotor system, there are six controllable states (roll, pitch, yaw, x position, y position, and height), and only four actuators, the four rotors. Therefore a single linear controller is not able to fully control a quadrotor.

To address this problem, we need to divide the system into two subsystems, containing an inner-loop representing the attitude and height dynamics and an outer-loop representing the XY position dynamics. Both of these loops are arranged into a cascade strategy. Furthermore, a PID controller is designed for each one, as shown in Figure 5.7. The outer-loop PID receives the reference signal for the overall position (XY) and the feedback of the states XY, then sends an attitude reference signal of the desired roll and pitch states to the inner-loop PID controller. The inner PID then sends the actuator signal to the quadrotor.

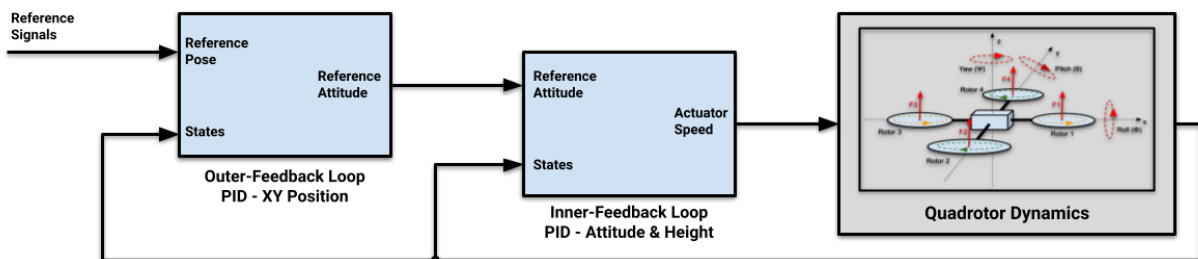


Figure 5.7: A block diagram shows the cascade control strategy.

### 5.5.6 Linear Quadratic Regulator (LQR) with Integral Action - Attitude and Height Control

To implement the theory behind the LQR with integral action, for the control of height and attitude of the quadrotor, first, we need to define our linear model:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \quad (5.58)$$

as a discrete linear model (see eq. 5.59).



$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases} \quad (5.59)$$

To achieve that goal, the use Integral Approximation Method is applied. According to [Gaj03], the integral approximation method for discretization of a continuous-time linear system is based on the assumption that the system input is constant during the given sampling period. Namely, the method approximates the input signal by its staircase form:

$$f(t) = f(kT), \quad kT \leq t < (k+1)T, \quad k = 0, 1, 2, \dots \quad (5.60)$$

$T$  represents the constant sampling interval. With the approximation of  $t = T$ , the space-states equation are given by:

$$\begin{aligned} x(T) &= e^{AT}x(0) + \int_0^T e^{A(T-\tau)}Bf(0)d\tau \\ &= e^{AT}x(0) + e^{AT} + \int_0^T e^{-A\tau}d\tau Bf(0) \\ &= \Phi(T)x(0) + \int_0^T \Phi(T-\tau)d\tau Bf(0) \end{aligned} \quad (5.61)$$

Therefore, we can conclude that,

$$A_d = e^{AT} = \Phi(T) \quad (5.62)$$

and,

$$\begin{aligned} B_d &= e^{AT} \int_0^T e^{-A\tau}d\tau B \\ &= \int_0^T e^{A(T-\tau)}d\tau B \\ &= \int_0^T e^{A\sigma}d\sigma B \end{aligned} \quad (5.63)$$

where  $\sigma = (k+1)T - \tau$ . The for the system's output can be derived in a similar manner:

$$y(kT) = Cx(kT) + Df(kT) \quad (5.64)$$

then, we can conclude that:

$$\begin{cases} C_d = C \\ D_d = D \end{cases} \quad (5.65)$$

To find  $B_d$ , integration is performed:

$$\begin{aligned} B_d &= e^{AT}(-e^{AT}A^{-1} + A^{-1})B \\ &= (A_d - I)A^{-1}B \end{aligned} \quad (5.66)$$

Finally we can represent the discretization of a continuous-time space-state system as:

$$\begin{cases} A_d = e^{AT} \\ B_d = (A_d - I)A^{-1}B \\ C_d = C \\ D_d = D \end{cases} \quad (5.67)$$

With the knowledge acquired in Sections 3.9.2 and 3.9.2, is possible to build the algorithm responsible for control the height and attitude of the quadrotor:

Algorithm 5.4: LQR gain with integral action used for the control the height and attitude parameters.

```

1 algorithm LQR_GAIN( $A_d, B_d$ ):
2   Set matrix  $Q$ ;
3   Set matrix  $R$ ;
4    $P = \text{RICCATI}(A_d, B_d, Q, R)$ ;
5    $K = R^{-1}B_d^T P$ ;
6 return ( $K$ )

```

The LQR gain algorithm runs offline before the quadrotor operation begins.

### 5.5.7 Kalman Filter for Full State Estimation - Height and Attitude Control

As said in Section 3.9.2, the goal of the LQR is to drive all the states to zero in the fastest amount of time, given a set of constraints described in the weighting matrices  $Q$  and  $R$ . Therefore, it is needed to estimate all of those states. Only the output states  $y$  (*height*, *roll*, *pitch* and *yaw*) are available to us at this point (see eq. 5.68).

$$\begin{bmatrix} z & \phi & \theta & \psi \end{bmatrix}^T \quad (5.68)$$

What is needed for the full state control of the system is the estimation of:

$$\begin{bmatrix} z & \dot{z} & \phi & \dot{\phi} & \theta & \dot{\theta} & \psi & \dot{\psi} \end{bmatrix}^T \quad (5.69)$$

To make this possible, we used a full state estimator, that receives the inputs and outputs of a system and estimates all of the states of said system. Figure 5.8 shows the representation of a generic estimator.

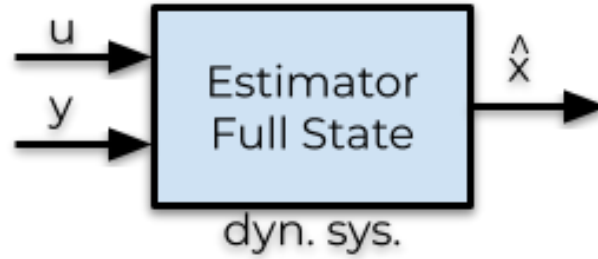


Figure 5.8: Block diagram representation of a generic estimator

In this work we chose a Kalman Filter based on a LQR to serve as the full state estimator. To build this estimator, first we need represent our space-state system considering disturbances and noise:

$$\begin{cases} \dot{x} = Ax + Bu + w_d \\ y = Cx + w_n \end{cases} \quad (5.70)$$

where,  $w_d$  is a gaussian white noise process of the disturbances acting in the states of system and  $w_n$  is a gaussian white noise process of the noise acting in the output of the system.

To start building the estimator, first we need to consider the following equation:

$$\dot{\hat{\varepsilon}} = (A - K_f C)\hat{\varepsilon} \quad (5.71)$$

where  $\hat{\varepsilon}$  is the prediction error of the estimator,  $K_k$  is the kalman filter gain and  $\varepsilon$  is the error ( $\varepsilon = x - \hat{x}$ ). As in the case of the LQR controller, the  $K_f$  gain that is necessary to minimize a cost function:

$$J = E((x - \hat{x})^t(x - \hat{x})) \quad (5.72)$$

where  $E((x - \hat{x})^t(x - \hat{x}))$  is the expected value of the error between the real values of the states and the prediction of the states by the estimator. Since is necessary to find a matrix gain to minimize a cost function, it is used the same linear algebra tools as used in the LQR controller calculation.

Therefore, we can find the kalman filter matrix gain by using a similar Algorithm of 5.5.

Algorithm 5.5: Kalman Filter Gain Algorithm for full-state estimator.

```

1 algorithm KALMAN_FILTER_GAIN( $A, C, V_d, V_n$ ) :
2    $P = \text{riccati}(A^T, C^T, V_d, V_n)$  ;
3    $K_f = V_n^{-1}CP$ ;

```

```
4 return (K_f)
```

where,  $V_d$  is the covariance matrix of the disturbance and  $V_n$  is the covariance matrix of the noise.

Then, to build the states estimator, we define the  $\hat{A}$ ,  $\hat{B}$ ,  $\hat{C}$  matrices as:

$$\begin{cases} \hat{A} = A - K_f C \\ \hat{B} = [B \quad K_f] \\ \hat{C} = C \end{cases} \quad (5.73)$$

Therefore the estimates states can be calculated with:

$$\hat{x} = \hat{A}y + \hat{B}u \quad (5.74)$$

Figure 5.9 shows the block diagram of the LQR controller setup with the kalman filter estimator.

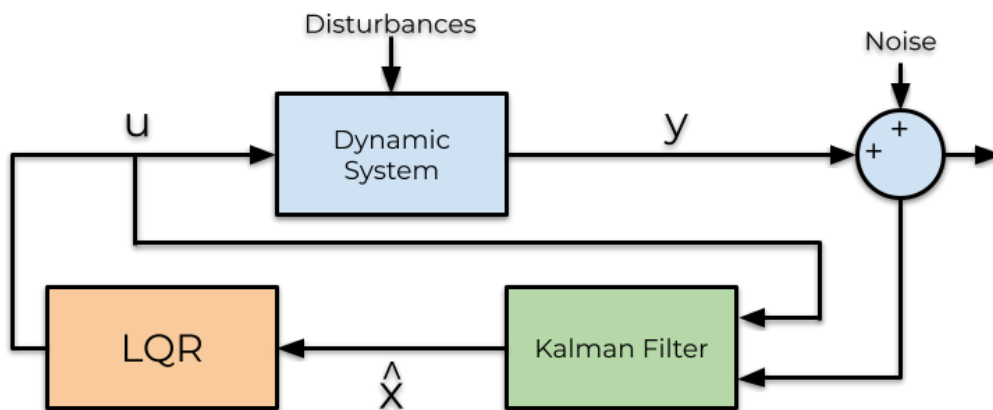


Figure 5.9: Block diagram of the LQR controller with a kalman filter estimator.

### 5.5.8 Model Predictive Control (MPC) - Height and Attitude Control

This section, shows the application of the theory explained in Section 3.9.3, how the algorithm was built and implemented on the MPSoC system. As the same for the LQR control, first, we need to define our linear model:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \quad (5.75)$$

and then, represent the system as a discrete linear model.

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases} \quad (5.76)$$

therefore, the first inputs of the MPC algorithm, are matrices  $A$ ,  $B$ ,  $C$  and  $D$ .

As [SEMDI16] states, the MPC calculations are based on current measurements and predictions of the future values of the outputs. The objective of the MPC control calculations is to determine a sequence of control moves so that the predicted response moves to the set point in an optimal manner. As shown in Figure 5.10 the current sampling instant is denoted by  $k$ , the MPC strategy calculates a set of  $M$  values of the input  $u$ :

$$u(k+i-1), \quad i = 1, 2, \dots, M \quad (5.77)$$

The set consists of the current input  $u(k)$  and  $M-1$  future inputs. The input is held constant after the  $M$  control moves. The inputs are calculated so that a set of  $P$  predicted outputs,

$$\hat{y}(k+i), \quad i = 1, 2, \dots, P \quad (5.78)$$

reaches the set point in an optimal manner. The control calculations are based on optimizing an objective function.

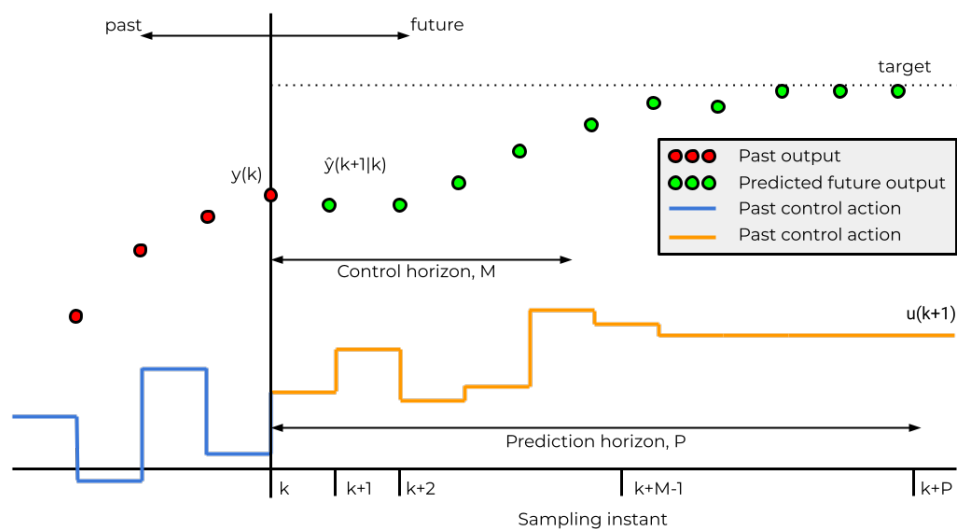


Figure 5.10: Basic concept for model predictive control.

The number of predictions  $P$  is referred to as the prediction horizon while the number of control moves  $M$  is called the control horizon. And these parameters are the basis to build the prediction matrices going forward.

Based on the equations on 3.63 and matrices  $A$ ,  $B$ ,  $C$  and  $D$ , the augmented system's matrices are build using Algorithm A.13.

Algorithm 5.6: Algorithm For Augmented System Build.

```

1 algorithm AUGMENTED_SYSTEM(A, B, C) :
2   Aa = CUSTOM_MAT(A, CA);
3   Ba = CUSTOM_MAT(B, CB);
4   Ca = CUSTOM_MAT(0, I);
5 return (Aa, Ba, Ca)

```

Next, it is necessary to build matrices  $F$  and  $\Phi$  from equation 3.70. To accomplish this task, it is used Algorithms A.13, A.24 and 5.6. Then, the auxiliary matrices that the MPC controller uses can be devised as:

Algorithm 5.7: Algorithm For Auxiliary MPC Build.

```

1 algorithm AUXILIARY_MATRICES( $\Phi, F, u, N_c, N_p$ ) :
2   BarR =  $\Gamma \times \text{ONES}((n_{\text{inputs}} N_c), (n_{\text{inputs}} N_c))$ ;
3   Phi_F =  $\Phi^T F$ ;
4   Phi_rs =  $\text{Phi\_F}(:, N_{\text{inputs}} - 2 : N_{\text{inputs}})$ ;
5 return (Aa, Ba, Ca)

```

With these matrices, we can implement the MPC controller based on the theory explained in Section 3.9.3.

Algorithm 5.8: Algorithm MPC Controller.

```

1 algorithm MPC_CONTROLLER( $x, \text{setpoint}, u, y$ ) :
2   xa = [ $\mathbf{x}_{\{k\}} - \mathbf{x}_{\{k-1\}}$ ;  $\mathbf{y}$ ];
3   Delta_U =  $(\Phi\Phi^T + \text{BarR})^{-1} \times (\text{Phi\_rs} \times \text{setpoint} \times \text{Phi\_F} \times \text{xa}_{k-1})$ ;
4    $u_k = u_{k-1} + \text{Delta\_U}$ ;
5 return (u)

```

As  $N_c$  is the controller horizon,  $N_p$  is the prediction horizon,  $\Gamma$  is the control effort and  $y$  is the system's output. And  $\Phi$  is calculated in equation 3.70.

### 5.5.9 PID Online Update - Fuzzy Control

To build the algorithm for calculate each of the PID gains used in the controller, it is used the functions described in appendix B. As described in Section 3.9.1 Algorithm 5.9 it is used obtain gains  $K_p$ ,  $K_i$ ,  $K_d$  for the attitude and height control. The algorithm receives the error and derivative error of roll, pitch, yaw and height, and based on a pre-defined list of rules and a defuzzification function, this algorithm is capable of return all the PID gains.

Algorithm 5.9: Algorithm for PID gain output.

```
1 algorithm gain_output(error, errorderivative):  
2   (PL, PM, ..., NL) = rules(NLerror, NLerrorderivative, ..., PLerror, PLerrorderivative);  
3   output = defuzzification(PL, PM, PS, ZE, NS, NM, NL);  
4   return (output)
```

## 6. EXPERIMENTS AND RESULTS

This Chapter presents a set of six main experiments and their results to validate the thesis proposal. These experiments concern the topics presented in Chapter 1.

Experiment 1, described in Section 6.1, consists of a *Proof-of-Concept* demonstration, embedding a distributed EKF-PID system to the ORCA MPSoC for the Quadrotor control. Experiment 2, described in Section 6.2, shows how an MPSoC can bring a decentralization feature into the control system architecture. Moreover, how this decentralization effectively improves the system's performance. In Experiment 3, Section 6.3, explores the system's capability to integrate more sophisticated control algorithms like LQR and MPC.

Experiment 4, depicted in Section 6.4, investigates how the system can provide fault-tolerant solutions for a runtime "freeze" of a processing core regarding the control performance and computational resources. Experiment 5, in Section 6.5, approaches the topic of energy management, which is explored by creating a system that controls the frequency of data injection into processing cores. Lastly, Experiment 6, in Section 6.6, tests the idea that the decentralization feature of the framework can be used to create an adaptive controller.

### 6.1 Experiment 1 - EKF with PID Implementation - Proof of Concept

This Section contains a PoC experiment to validate the system at its basic functionalities. These tests implements the ROS communication nodes in core zero, an EKF algorithm (sensor fusion) in core one, and a PID controller (quadrotor control) in core two. Figure 6.1 shows a representation of the software organization in the MPSoC. Section 6.1.1 evaluates the system's capability to estimate its attitude with the EKF algorithm. Section 6.1.2, brings the results of the PID controller output. Furthermore, Section 6.1.3 test four different application and hardware setups to determine the best configuration, considering energy consumption and response time for the task set.

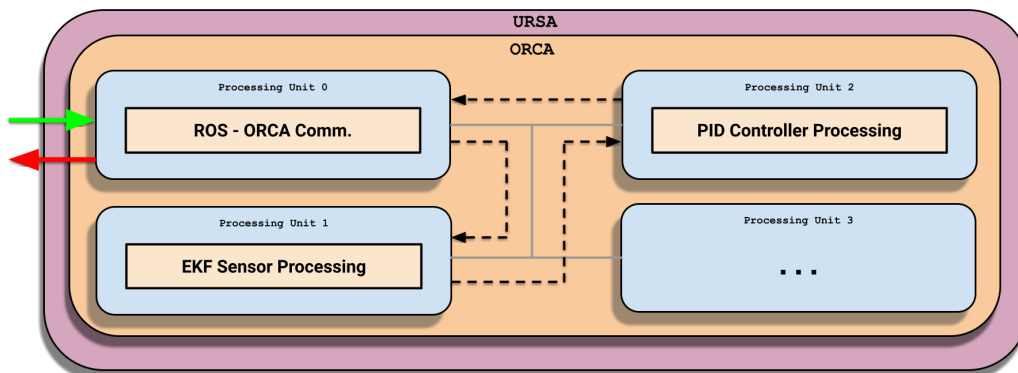


Figure 6.1: PID - EKF Implementation Diagram.



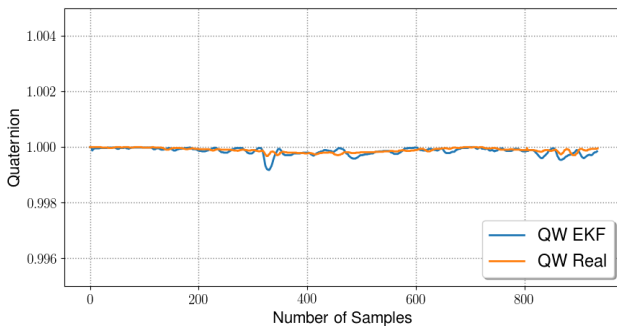
### 6.1.1 Experiment 1.A - EKF Performance

This test was designed to assess the system's capability to estimate, with accuracy, the quaternion that represents the quadrotor attitude. As previously explained, the Hector simulation publishes the IMU sensors data in topic `/raw_imu` and the magnetometer in topic `/magnetic`. Then this data is consumed by the ROS nodes running on core zero, therefore sent to core one to be processed by the EKF.

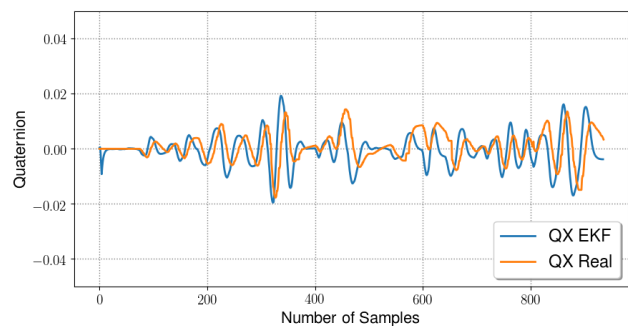
To determine the accuracy of the attitude estimation by the EKF, the system uses the information provided by ROS in the topic `/ground_truth_to_tf/pose`, that keeps the factual pose information of the simulation. That way, it is possible to make a definitive comparison.

In this experiment, the setpoint for the quadrotor stabilization is given by the quaternion  $q = \{1, 0, 0, 0\}$ , which represents the configuration: roll angle ( $\phi$ ) =  $0^\circ$ , pitch angle ( $\theta$ ) =  $0^\circ$  and yaw angle ( $\psi$ ) =  $0^\circ$ .

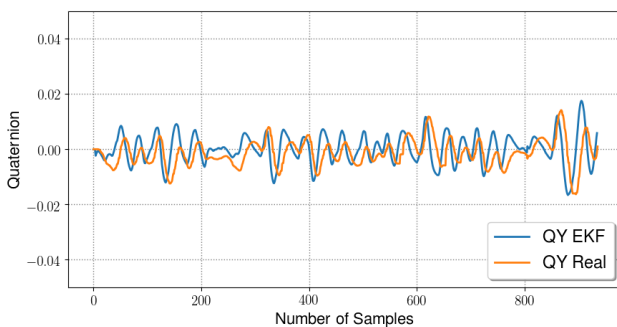
Figure 6.2 shows the results obtained by the experiment for all of the quaternion components.



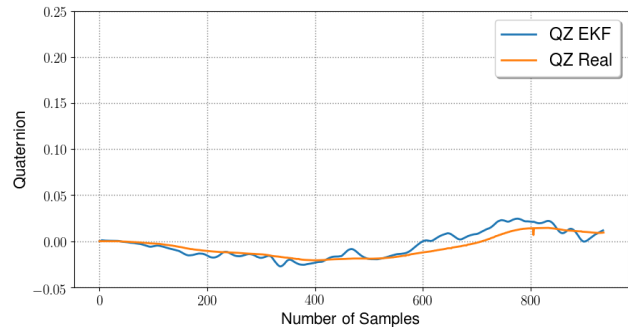
(a) Comparison of "QW" component between the EKF estimation and simulated real value.



(b) Comparison of "QX" component between the EKF estimation and simulated real value.



(c) Comparison of "QY" component between the EKF estimation and simulated real value.



(d) Comparison of "QZ" component between the EKF estimation and simulated real value.

Figure 6.2: Results of Experiment 1.A - Graphical comparison between the EKF estimation of the attitude quaternion and the real value presented by the simulation.

Based on Figure 6.2, it is possible to assess that the estimation of the EKF approximates the real simulated value of the quadrotor's attitude quaternion. Since this is a preliminary experiment, the numerical analysis of these results are only made in subsequent sections. The experiments of Sections 6.1.1 and 6.1.2 are only devised to validate the system's functionality. Then, the graphical analysis of the results shows that the quaternion estimation adequately approximates the real value.

### 6.1.2 Experiment 1.B - PID Performance

As the data from the EKF was collected in this experiment, it was also collected the data from the PID controller. Therefore, able to determine if this controller can keep the quadrotor hovering over the ground at a predetermined height. The controller set the height setpoint at 1 meters above the ground, and the attitude angles are the same as it is in the Section 6.1.1.

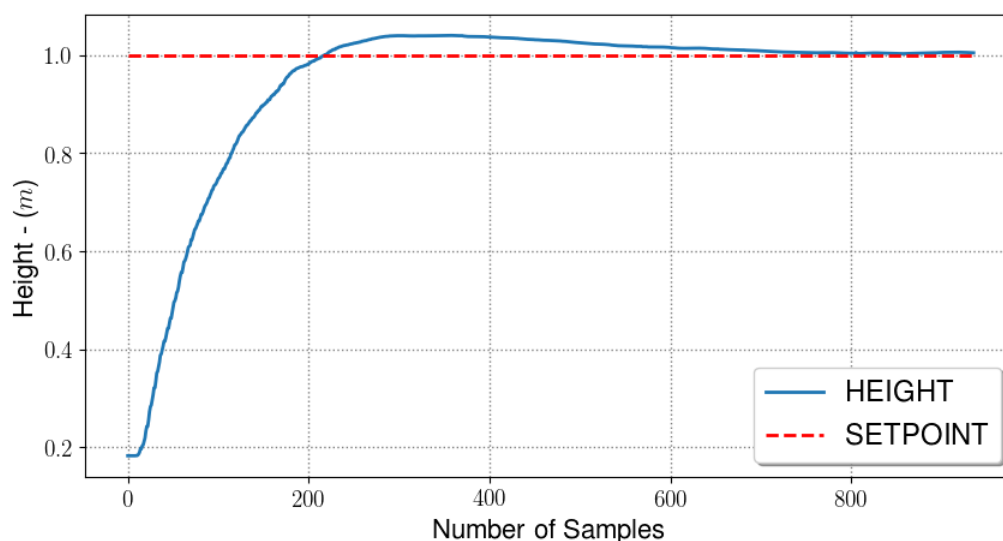


Figure 6.3: Results of Experiment 1.B - Height measured in meters representing the output of PID controller.

Figure 6.3 shows that the PID controller could stabilize the quadrotor at the desired height with minimal overshoot but at a relatively low rise speed. Therefore it is possible to conclude that as a preliminary analysis, the proposed system represented in Figure 6.1 showed satisfactory results for controlling the attitude and height of the quadrotor. As the work progresses in this thesis, it is possible to assess the optimum responses that this system is capable. And determine if it can reduce overshoot and stabilization time with better resource management.

### 6.1.3 Experiment 1.C - Floating Point $\times$ Fixed Point

To determine the best setup to conduct the experiments in Sections 6.2, 6.3, 6.4, 6.5 and 6.6, is devised four tests regarding the type of the manipulated data format and hardware component units. These tests compared four different application setups to determine the best configuration, considering the task set's energy consumption and response time. For both tasks, the system ran under environment variations using a software-emulated floating-point (SEFP), fixed-point, and a hardware multiplier unit (MU). There were four variations:

1. system with SEFP only;
2. SEFP + MU;
3. system with fixed-point only;
4. fixed-point + MU.

For each variation, is collected statistics for 100 iterations of each task. The fixed-point data is represented in 16.16 format (16-bit integral, 16-bit fractional parts). The response time results assume a clock period of 4 ns. The hardware counters of Section 4.6.4 are applied for each task to calculate the energy consumption.

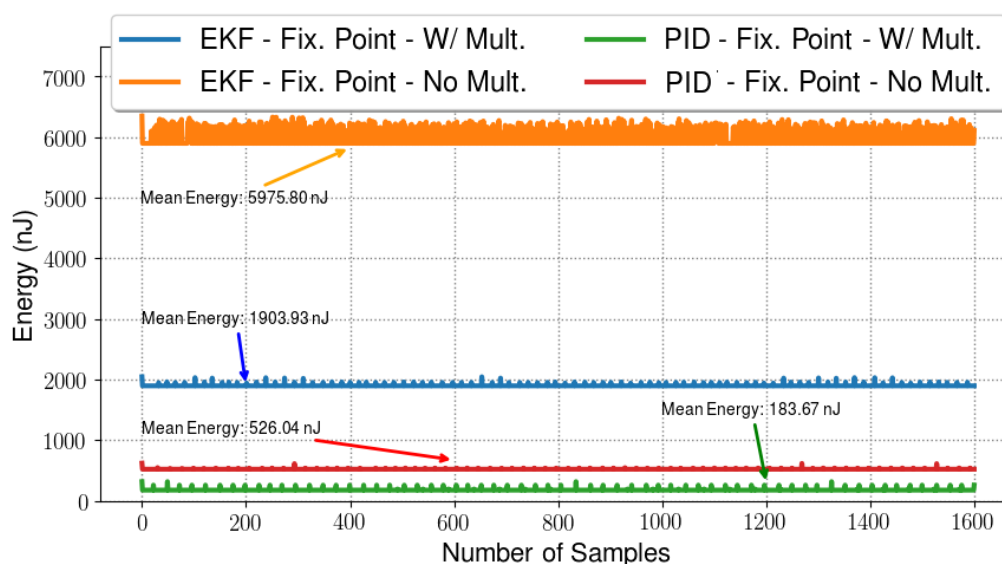


Figure 6.4: Energy estimation results by algorithm with calculations made with fixed-point format. With and without hardware multiplier unit.

Figure 6.4 shows the measurement of the energy consumed to process the EKF and PID algorithms using Fixed-point format and variations of hardware: with/without a multiplier unit.

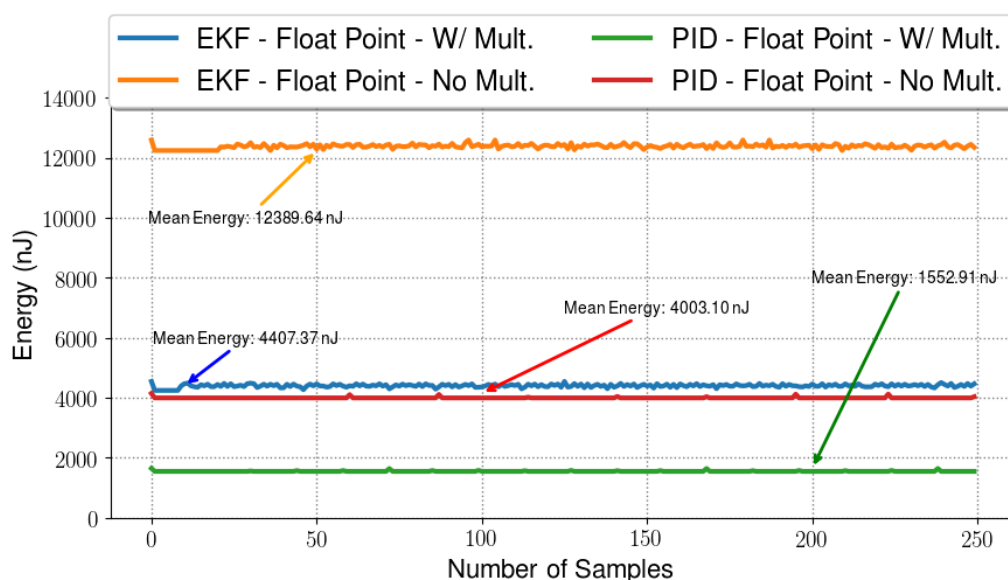


Figure 6.5: Energy estimation results by algorithm with calculations made with software-emulated floating-point format. With and without hardware multiplier unit.

Figure 6.5 shows the measurement of the energy consumed to process the EKF and PID algorithms using software-emulated floating-point format and variations of hardware: with/without a multiplier unit.

Analyzing both figures, it is possible to see that energy consumption drastically increases using software-emulated floating-point compared to fixed-point numbers. The other aspect inspected is the utilization of the multiplier unit, which shows that the utilization of the MU also decreases the energy consumed. Table 6.1 shows the numerical values that resume the findings of these tests. Besides the energy estimation, it is also measured the latency for each algorithm executed. That has been the time it takes to the core to process an iteration of an algorithm.

Table 6.1: Response time and energy evaluation for PID and EKF tasks.

	Task	Configuration			
		SEFP		fixed-point arith.	
		with MU	no MU	with MU	no MU
Avg. Response Time (ms)	PID	17 ~ 19	47 ~ 48	1 ~ 2	7 ~ 9
	EKF	43 ~ 48	123 ~ 126	18 ~ 20	77 ~ 81
Avg. Energy (nJ)	PID	1552.91	4003.10	183.67	526.04
	EKF	4407.37	12389.64	1903.93	4003.10

Regarding the utilization of a hardware multiplier unit, it reduces, on average, the processing time by 106% and the energy consumption by 87%. The difference between fixed and floating-point energy consumption is 125%, and the processing time is 100%. The defining factor for determining the quadrotor control performance was the latency of the EKF algorithm. When the configuration used was the combination of SEFP and no MU, the quadrotor was not controllable, and there was not enough time to stabilize the system.

By far, the better result is obtained by the Fixed-point and MU variation. Furthermore, that configuration is used in the subsequent experiments.

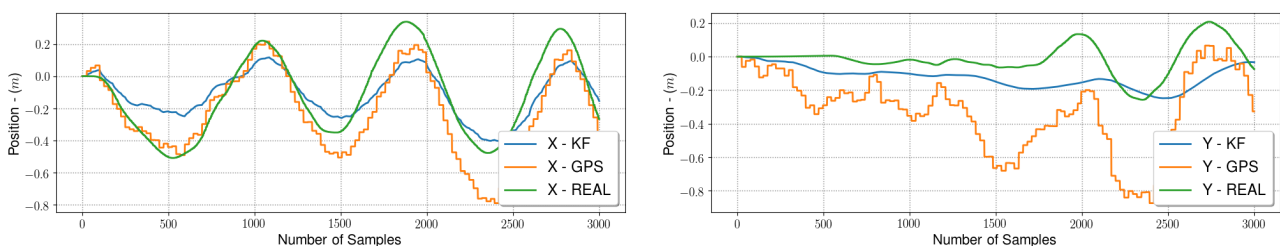
## 6.2 Experiment 2 - EKF/KF with PID Implementation - Centralization $\times$ Decentralization

This section explores the topic of the decentralization of the control architecture in the MPSoC frame. This experiment aims to compare a traditional implementation of a control system in only one core and a distributed one, with tasks assigned to several cores. It adds one additional task to the experiment of Section 6.1 to achieve this goal, the linear Kalman Filter for XY position estimation.

Section 6.2.1 performs a preliminary test, only evaluating the performance and accuracy of the estimation algorithm. Section 6.2.2 presents a baseline by implementing all estimation algorithms in a single thread (EKF + KF), running in a single core. Then, Section 6.2.3 distributes these tasks into two separate cores and rerun the simulation, allowing to assess the potential gains of this technique.

### 6.2.1 Experiment 2.A - KF Performance

As stated previously, this section shows the Kalman Filter's performance for the XY position estimator. For this test, in addition to implementing the KF algorithm, the PID controller counts with an additional controller: the XY position controller (see Section 5.5.5). This way creates a baseline for the evaluation of the KF.



(a) Comparison between the X position estimation by the KF algorithm, the GPS measurements and the simulated real position.

(b) Comparison between the Y position estimation by the KF algorithm, the GPS measurements and the simulated real position.

Figure 6.6: Results of Experiment 1.C - Graphical comparison between the XY position estimation by the KF algorithm, the GPS measurements and the simulated real position.

This experiment tests the accuracy of the KF for the XY position estimation. Here, the PID controller is set to position the quadrotor in the (0,0) position in the simulated environment. Also, the PID gains are set so that the quadrotor does not stabilize in the desired

position. Therefore, the quadrotor keeps flying around the setpoint position to test the KF estimator in motion.

Figure 6.6, shows the results of the KF position estimation, comparing the results with the measured position by the GPS and the actual data gathered by the ROS system. The graphical analysis of the graphs shows that the KF estimation significantly improves the GPS measurements. Although the Figure 6.6a shows a relatively small improvement from the GPS measurements to the KF estimation, the Figure 6.6b shows that the KF could correct the drift from the GPS measurements, using the accelerometer data.

Table 6.2: Table of mean errors of the XY position estimation made by the KF algorithm and the GPS measurements.

	X Position	Y Position
KF Mean Error - (m)	0.11877119164905567	0.11374884074983133
GPS Mean Error - (m)	0.13452016188308036	0.31011143907124045

To better understand these improvements, the Table 6.2 shows the numerical values gathered by this experiment. The data shows that the mean error of the X position estimation is improved by 12% from the GPS measurements. Moreover, the mean error of the Y position estimation is improved by 92%.

### 6.2.2 Experiment 2.B - EKF + KF (Centralized Processing)

In this experiment, the system runs both KF and EKF in a single thread in core 1, then feed the resulting data to the PID controller. This test creates a baseline that it is used to determine if there was any gain in the use of the decentralized system. An overview of this test is described in Figure 6.7.

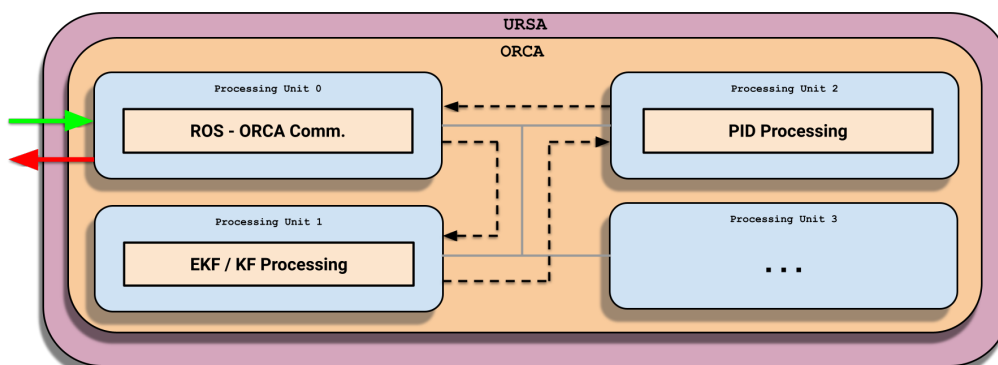
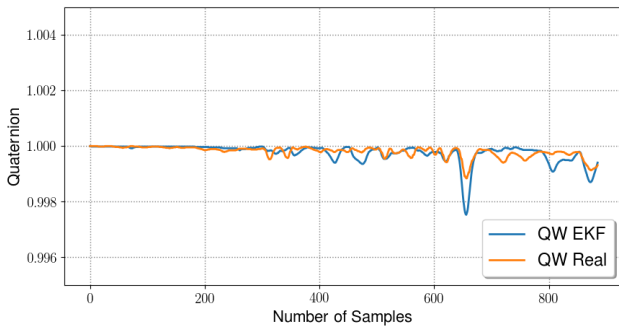


Figure 6.7: EKF + KF Centralized Processing Implementation Diagram.

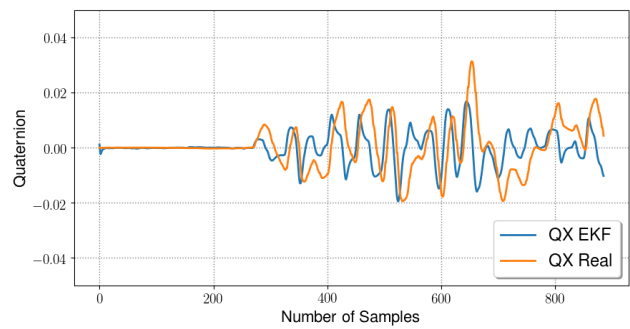
After running the simulations, the system can extract the necessary information to create the planned comparisons. In this case, the height output performance, the attitude output performance, the XY position performance, and the consumed energy.

## Experiment 2.B.A - EKF + KF + PID (Attitude Estimation Results)

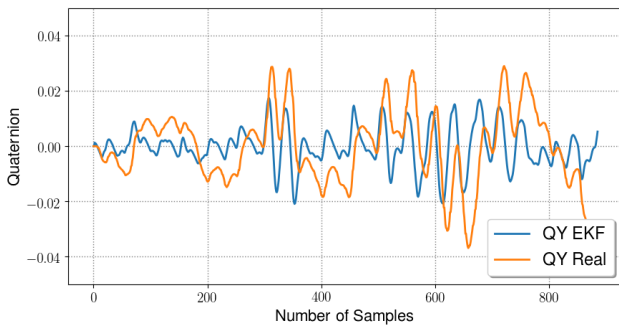
This experiment, tests this configuration's capability to estimate the quadrotor's attitude quaternion during the simulation. Then these results are compared to the decentralized configuration in Section 6.2.3. As in the previous section, the PID controller is set to move the quadrotor to the (0,0) position in the simulated environment, starting in the (0,0) position.



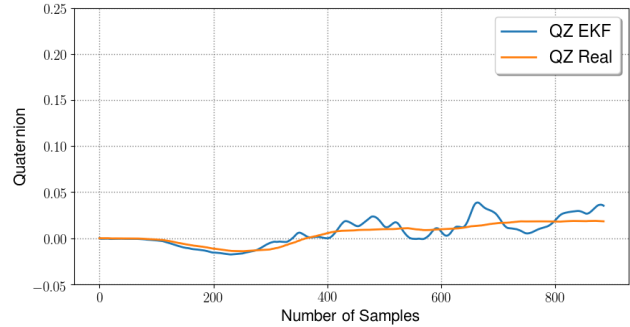
(a) Comparison of "QW" component between the EKF estimation and simulated real value.



(b) Comparison of "QX" component between the EKF estimation and simulated real value.



(c) Comparison of "QY" component between the EKF estimation and simulated real value.



(d) Comparison of "QZ" component between the EKF estimation and simulated real value.

Figure 6.8: Results of Experiment 2.B.A - Graphical comparison between the EKF estimation of the attitude quaternion and the real value presented by the simulation, utilizing a centralized framework.

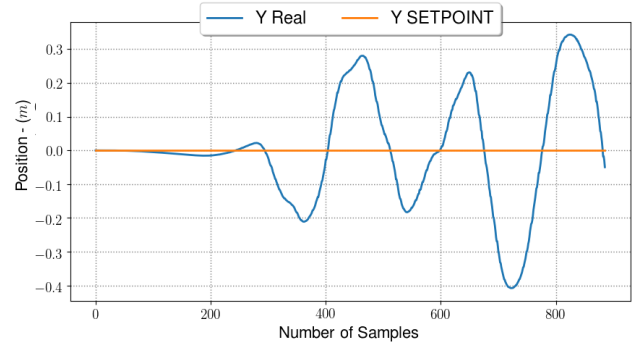
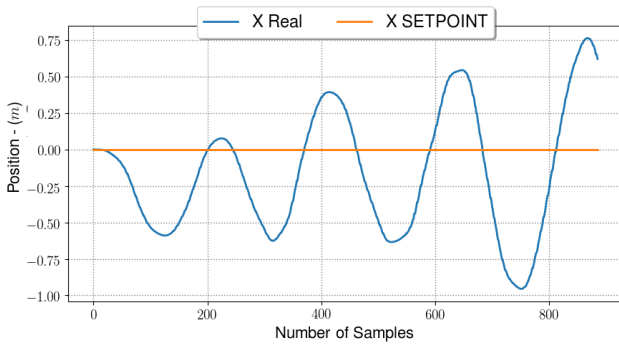
Table 6.3 shows all the estimated quaternion's mean errors by its components. These results are the baseline to determine the impact of the decentralization of this system.

Table 6.3: Table of mean errors of the quaternion estimation made by the EKF algorithm and the simulated real attitude quaternion.

	QW	QX	QY	QZ
EKF Mean Error	0.000146	0.00615	0.0108	0.00581

Experiment 2.B.B - EKF + KF + PID (XY Position Control Results)

This experiment tests the system’s capability to keep the quadrotor in a fixed position during flight. As for the previous test, these results are the baseline for determining the decentralized configuration impact on the control performance. Figure 6.9 shows the graphical results for each measured axis.



(a) Comparison between the X position estimation by the KF algorithm and the simulated real position, utilizing a centralized framework.

(b) Comparison between the Y position estimation by the KF algorithm and the simulated real position, utilizing a centralized framework.

Figure 6.9: Results of Experiment 2.B.C - Graphical comparison between the XY position estimation by the KF algorithm, and the simulated real position, utilizing a centralized framework.

Based on these graphs, it is possible to determine that the system can not stabilize in the desired position. This determination can be made since the position error keeps increasing with time. Moreover, Table 6.4 shows the mean error of the quadrotor’s position to the setpoint. On axis X, the system presents a mean error of 0.38 meters, and on the Y axis, a mean error of 0.11 meters.

Table 6.4: Table of mean errors of the XY position estimation made by the KF algorithm and the simulated real position, utilizing a centralized framework.

	X Position	Y Position
PID Mean Error (m)	0.3803432246049657	0.11861218623024829

Experiment 2.B.C - EKF + KF + PID (Height Control Results)

This experiment assesses the system’s capability to stabilize the quadrotor at a specific height during flight. Figure 6.10 shows the results of the system’s attempt to get to the height setpoint while trying to keep the quadrotor at 1m over the ground.



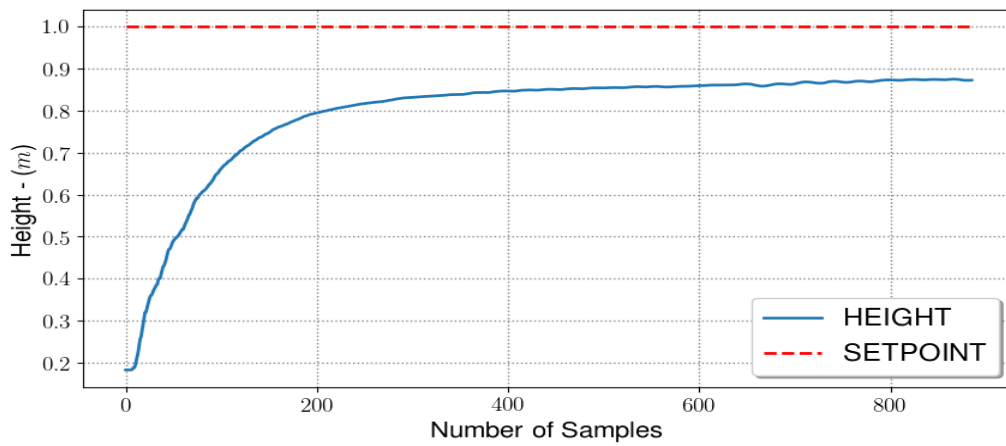


Figure 6.10: Results of Experiment 2.B.B - Height measured in meters representing the output of PID controller, utilizing a centralized framework.

The graph shows that, although the quadrotor keeps a stable Z-axis trajectory, this process is relatively slow and does not reach the determined setpoint during the simulation time. As the XY controller gains were set, the height controller was significantly impacted due to the time constraint that the EKF+KF process creates.

#### Experiment 2.B.D - EKF + KF + PID (Energy Estimation Results)

This experiment creates a baseline to compare the system's centralized and decentralized configuration energy consumption. Figure 6.11 shows the energy consumption in each core during flight. The blue curve shows the mean energy that the processing of the KF+EKF takes in each iteration. The orange curve shows how much energy the PID algorithm consumes in core two in each iteration.

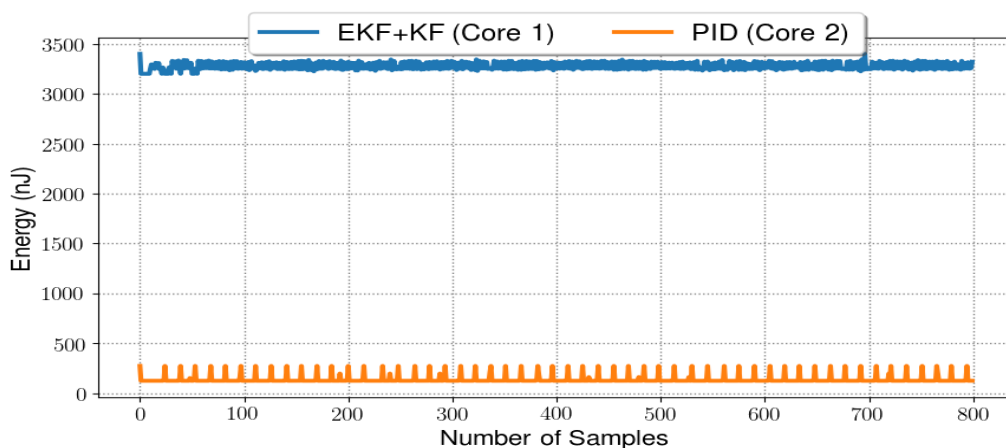


Figure 6.11: Energy estimation results by algorithm within cores.

Table 6.5 explicitly shows the mean energy consumed by each core. These results are also be the baseline for the subsequent section.

Table 6.5: Table of the results of energy estimation results by algorithm within cores.

	Mean Energy (nJ)
EKF + KF (Core 1)	3270.130133562798
PID (Core 2)	137.72794549280152

### Experiment 2.B.E - EKF + KF + PID (Latency Results)

This experiment measures the time it takes each iteration of the algorithms to run in the cores. As previously explained, this also serve as the baseline for comparing the decentralized configuration. Based in this experiment, it was possible to approximately assess the mean time for each process. The time of each iteration is not constant. This system is running in a simulation on a personal computer. Therefore, these variations can be attributed to the fact that this computer is also running other processes that impact this simulation. Finally, the time that the EKF+KF takes to process is approximately 40 ms. Moreover, the PID process takes 2ms to run. To determine the overall time, it is added the time of cores 1 and 2 since they constitute a data pipeline, as shows the Figure 6.12.

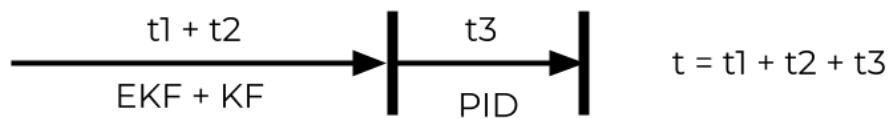


Figure 6.12: Overall time diagram for centralized processing configuration.

### 6.2.3 Experiment 2.C - EKF + KF (Decentralized Processing)

In this experiment, the KF and EKF are implemented in core 1 and 3, respectively, and they feed their output simultaneously data to the PID controller. This test creates a dataset used to determine if there was any gain in using the decentralized system. An overview of this test is described in Figure 6.13.

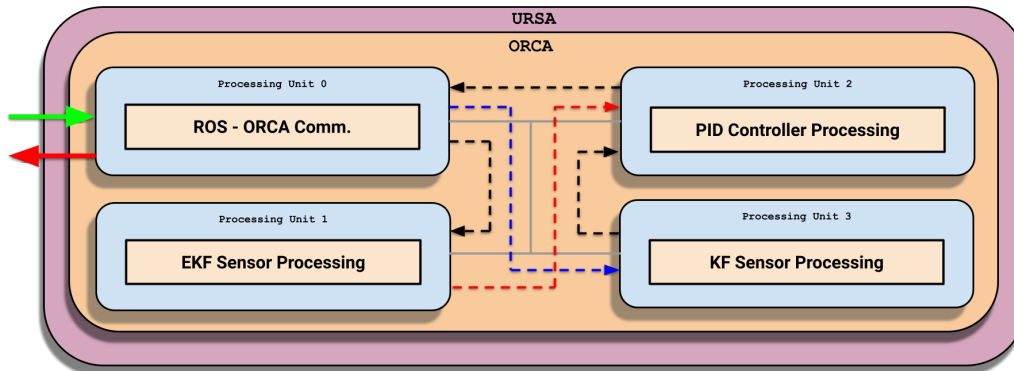


Figure 6.13: EKF + KF Decentralized Processing Implementation Diagram.

As for the experiment in Section 6.2.2, here is extracted the same information for later comparison.

#### Experiment 2.C.A - EKF + KF + PID (Attitude Control Results)

This experiment tests this configuration's capability to estimate the quadrotor's attitude quaternion during the simulation using the decentralized configuration. As in the Section 6.2.2, the PID controller sets the quadrotor to move to the (0,0) position in the simulated environment, starting in the (0,0) position. Figure 6.14 shows the graphical results of this experiment.

Making a graphical analysis, it is possible to, at first, determine that the EKF running by itself in a single core has a similar result as the combination of EKF and KF running in a single core. And Table 6.6 shows all the estimated quaternion's mean errors by its components.

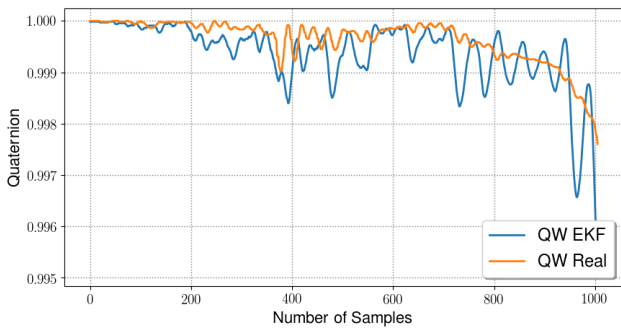
Table 6.6: Table of mean errors of the quaternion estimation made by the EKF algorithm and the simulated real attitude quaternion.

	QW	QX	QY	QZ
EKF Mean Error	0.0003285	0.012777	0.011616	0.01068

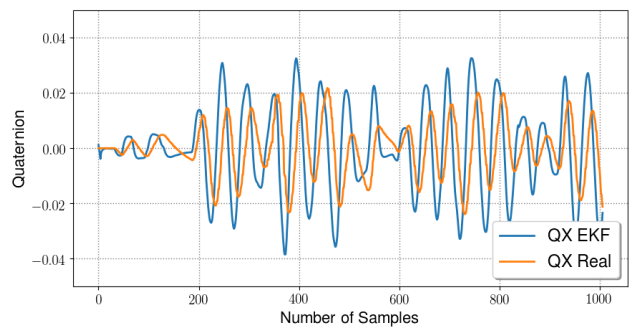
To better understand the results of the differences between the centralized and decentralized configurations regarding the EKF processing, it is possible to point to Table 6.7. In this table, is given both configurations' data and give the difference between them.

Table 6.7: Table comparing the mean errors of the quaternion estimation made by the EKF algorithm in both configuration.

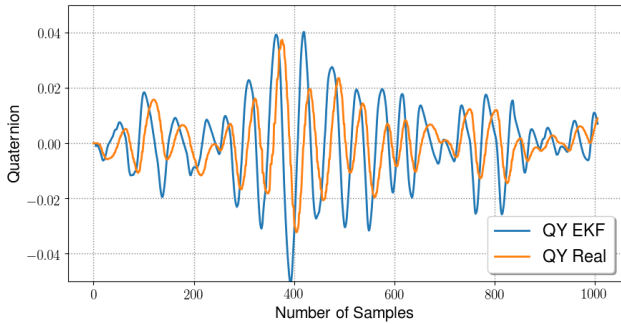
	QW	QX	QY	QZ
EKF Mean Error (Centralized)	0.000146	0.00615	0.0108	0.00581
EKF Mean Error (Decentralized)	0.0003285	0.012777	0.011616	0.01068
EKF Mean Difference	0.0001825	0.006627	0.000816	0.00487



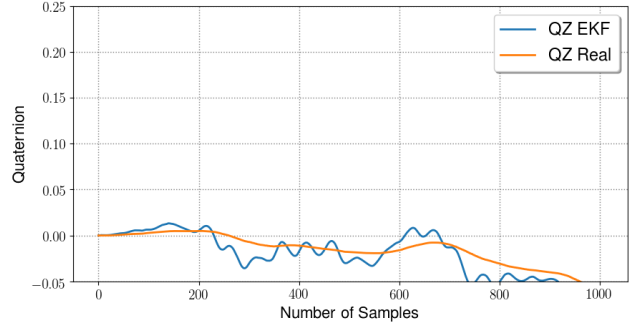
(a) Comparison of "QW" component between the EKF estimation and simulated real value.



(b) Comparison of "QX" component between the EKF estimation and simulated real value.



(c) Comparison of "QY" component between the EKF estimation and simulated real value.



(d) Comparison of "QZ" component between the EKF estimation and simulated real value.

Figure 6.14: Results of Experiment 2.C.B - Graphical comparison between the EKF estimation of the attitude quaternion and the real value presented by the simulation, utilizing a decentralized framework.

When translating these quaternion's results to Euler angles, it is possible to determine that the difference between both configurations is lower than  $1^\circ$ . These results show that, at least in this experiment, there are no significant differences between configurations.

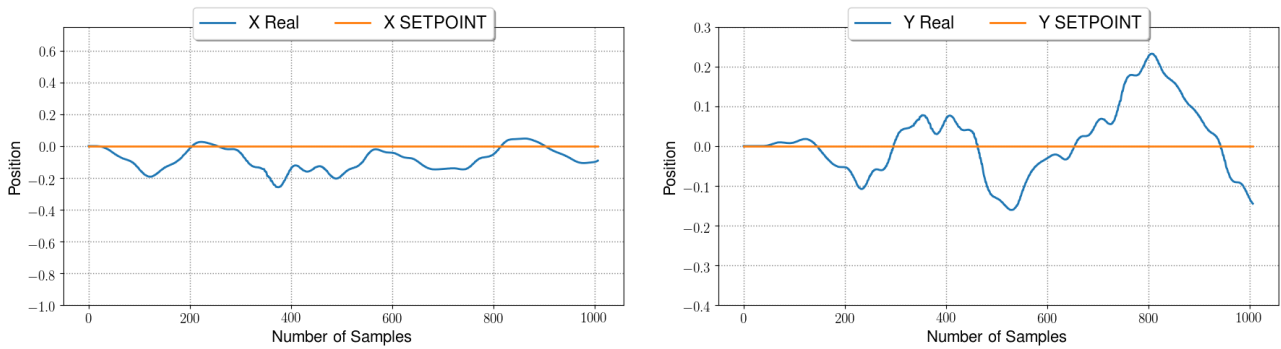
Experiment 2.C.B - EKF + KF + PID (XY Position Control Results)

This experiment, as for Section 6.2.2, tests the system's capability to keep the quadrotor in a fixed position during flight. Figure 6.15 shows the graphical results for each measured axis.

Based on these graphs, it is possible to determine that the system can not stabilize in the desired position. Moreover, Table 6.8 shows the mean error of the quadrotor's position to the setpoint. On axis X, the system presents a mean error of 0.09 meters, and on the Y axis, a mean error of 0.06 meters.

Table 6.8: Table of mean errors of the XY position estimation made by the KF algorithm and the simulated real position, utilizing a decentralized framework.

	X Position	Y Position
PID Mean Error (m)	0.09015359745821944,	0.0686649148080268



(a) Comparison between the X position estimation by the KF algorithm and the simulated real position, utilizing a decentralized framework.

(b) Comparison between the Y position estimation by the KF algorithm and the simulated real position, utilizing a decentralized framework.

Figure 6.15: Results of Experiment 2.B.C - Graphical comparison between the XY position estimation by the KF algorithm, and the simulated real position, utilizing a centralized framework.

Although this configuration could not also stabilize the quadrotor in the XY position, the mean error of the positions was significantly reduced. In the X position, the mean error reduced from 0.38m to 0.09m, a 123% decrease. In the Y position, the mean error reduced from 0.11m to 0.06m, a 58% decrease.

#### Experiment 2.C.C - EKF + KF + PID (Height Control Results)

As for Section 6.2.2, this experiment assesses the system's capability to stabilize the quadrotor at a specific height during flight. Figure 6.16 shows the results of the system's attempt to get to the height setpoint while trying to keep the quadrotor at 1m over the ground.

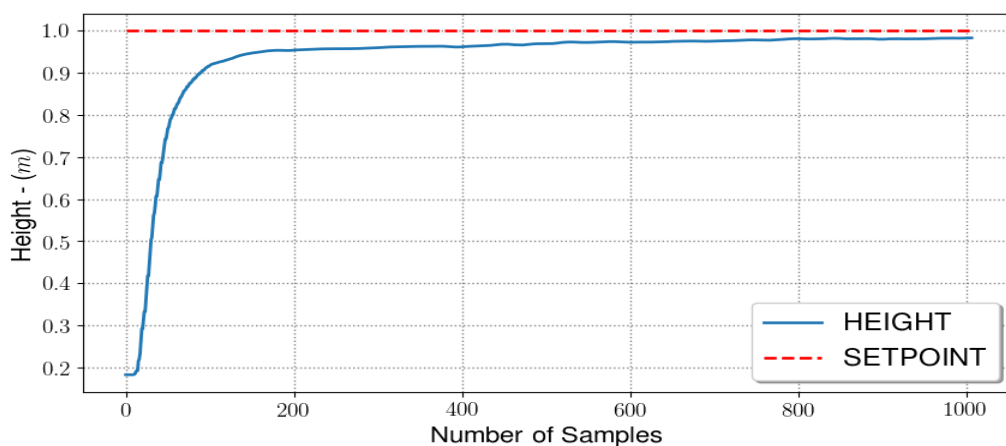


Figure 6.16: Results of Experiment 2.C.A - Height measured in meters representing the output of PID controller, utilizing a decentralized framework.

The graph shows that, although the quadrotor keeps a stable Z-axis trajectory, this process performs significantly better compared to the centralized configuration. The rise

time decreased, and the stable state error decreased from approximately 10cm to less than 1cm. Separating these algorithms and running them in parallel decreased the time constraint the centralized configuration brought.

#### Experiment 2.C.D - EKF + KF + PID (Energy Estimation Results)

This experiment, runs these distributed algorithms to determine the energy consumption in each core. Figure 6.17 shows the energy consumption in each core during flight. The blue curve shows the mean energy that the processing of the EKF takes in each iteration. The orange curve shows how much energy the KF algorithm consumes in core three in each iteration. The green curve shows the mean energy that the processing of the PID takes in each iteration.

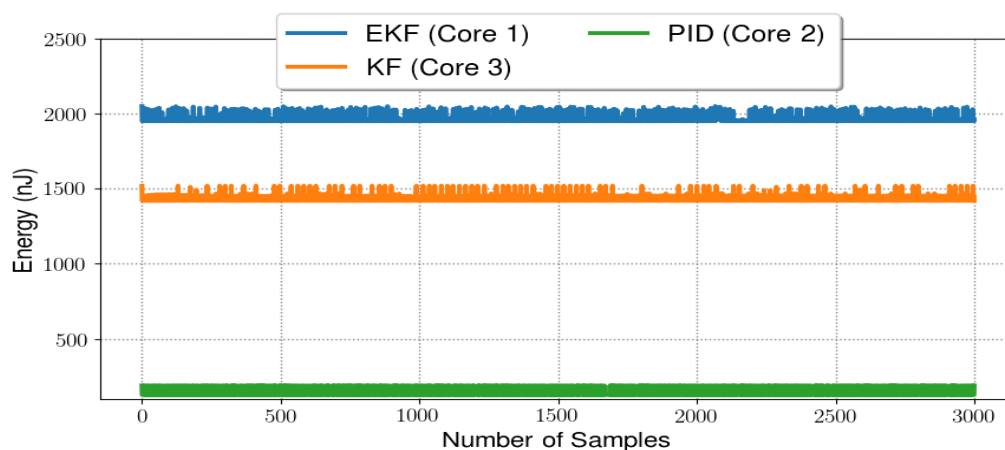


Figure 6.17: Energy estimation results by algorithm within cores.

Table 6.9 explicitly shows the mean energy consumed by each core.

Table 6.9: Table of the results of energy estimation results by algorithm within cores.

	Mean Energy (nJ)
EKF (Core 1)	1960.444
PID (Core 2)	136.688
KF (Core 3)	1429.942

As for energy consumption, the power used to run the processor is higher than the centralized configuration. The energy consumed increased from 3407 nJ to 4049 nJ. An increase of 17%. This increase is explained by the fact that even in idle, the core consumes energy running code to wait for the next package to be read. Although the decentralized configuration consumes more energy than the centralized configuration, this consumption is distributed in more cores.

### Experiment 2.C.E - EKF + KF + PID (Latency Results)

This experiment, measures the time it takes each iteration of the algorithms to run in the cores. And it was possible to approximately assess the mean time for each process, determining that the EKF process takes approximately 22 ms, the KF process takes approximately 15 ms, and the PID process takes 2ms to run. To determine the overall time, instead of adding all of the processes' time, only the more significant time value between the EKF and the KF algorithms are added. This calculation is done because the EKF and KF are done in parallel, as the Figure 6.18 shows. When the centralized configuration is running, the process took approximately 42ms; now, decentralizing the process, it ran in approximately 24ms.

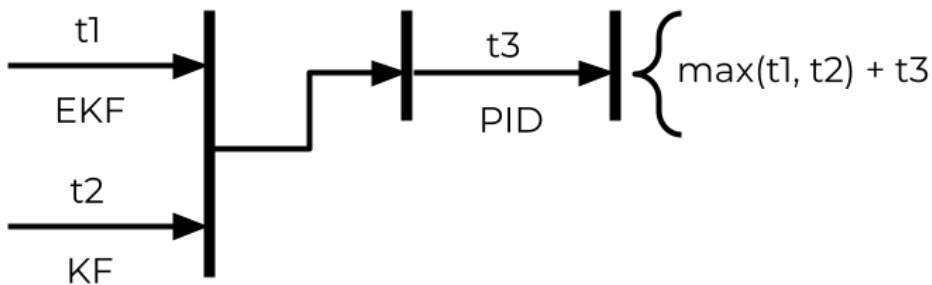


Figure 6.18: Overall time diagram for decentralized processing configuration.

These experiments show that although there is a slight increase in energy consumption with a decentralized setup, the performance gains compensate for this drawback.

### 6.3 Experiment 3 - Capability of More Complex Control Implementation Evaluation

This Section determines the capability of the proposed system to accept more complex and demanding control options. In Section 6.3.1, it is possible to implement and evaluate an LQR controller for the quadrotor. Furthermore, in Section 6.3.2, an MPC controller for the quadrotor is implemented and evaluated. Figure 6.19 shows an overview of these experiments.

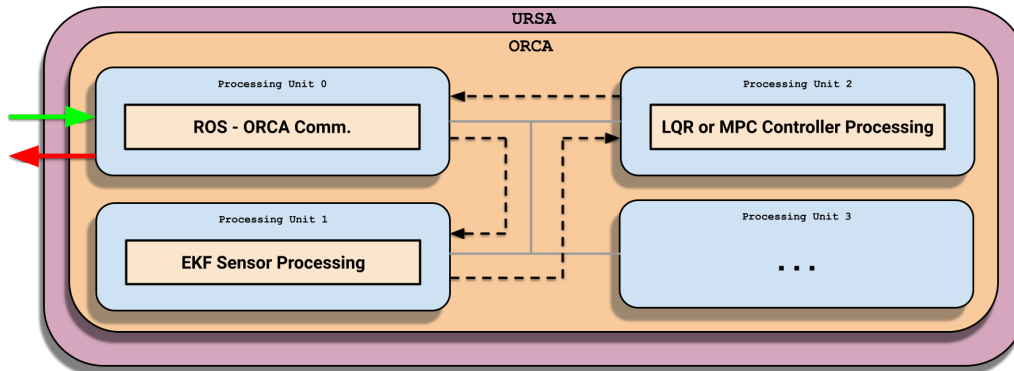


Figure 6.19: LQR/MPC Control Implementation Diagram.

### 6.3.1 Experiment 3.A - LQR

This section takes the scheme of experiment 1 and substitutes the PID controller for an LQR controller. Then, rerun the simulations and gather the same information from experiment 1.

#### Experiment 3.A.A - LQR Performance

Here, it is set the the setpoint for the quadrotor stabilization in  $q = \{1, 0, 0, 0\}$ , which represents the configuration: roll angle ( $\phi$ ) =  $0^\circ$ , pitch angle ( $\theta$ ) =  $0^\circ$  and yaw angle ( $\psi$ ) =  $0^\circ$ , the same as in Section 6.1.2. Figure 6.20 shows the results from this test.

Graphically analyzing the 6.20, it is possible to deduce that the LQR control kept the quadrotor fairly steady with minimal roll and pitch variations. The LQR explains this improvement over the PID's quaternion control. The control forces all states to reach zero, adding only 6ms of latency to the control task.

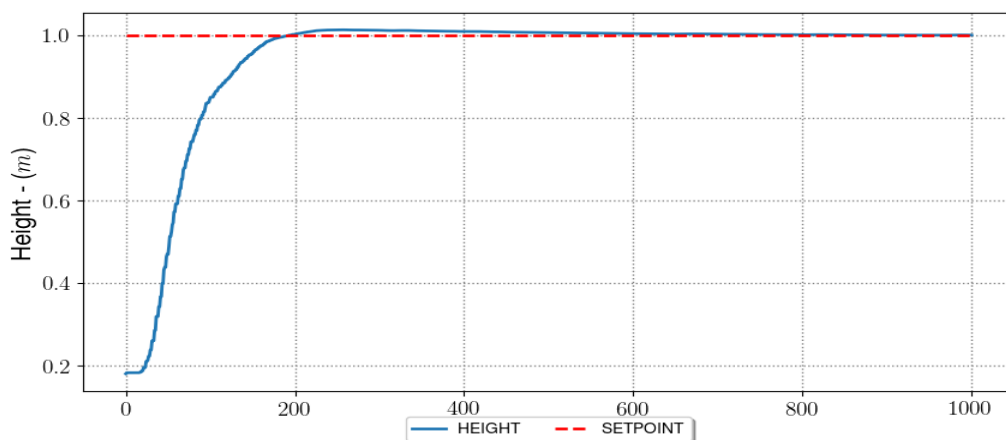
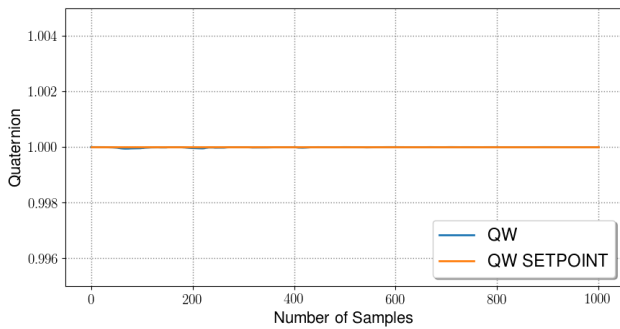
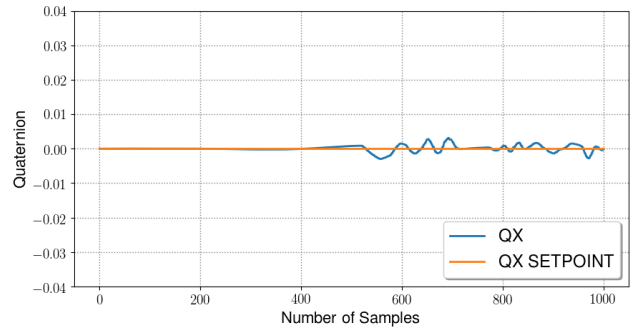


Figure 6.21: Results of Experiment 3.A.A - Height measured in meters representing the output of LQR controller.

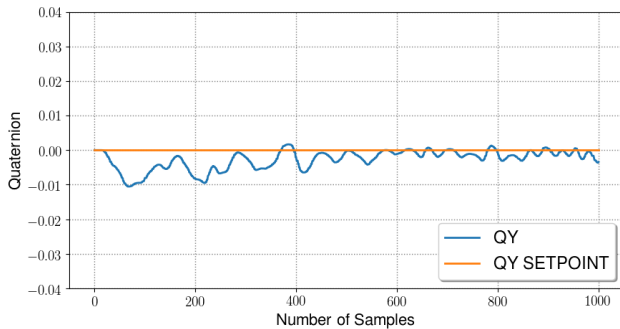




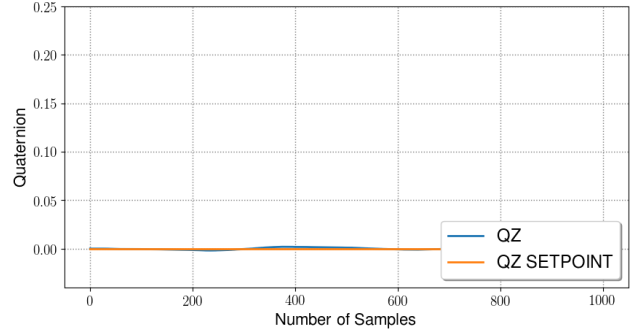
(a) Representation of "QW" quaternion component resulting from the LQR control.



(b) Representation of "QX" quaternion component resulting from the LQR control.



(c) Representation of "QY" quaternion component resulting from the LQR control.



(d) Representation of "QZ" quaternion component resulting from the LQR control.

Figure 6.20: Results of Experiment 3.A.A - Graphical representation of the measured quaternion of the LQR controller output.

Comparing the results of the height control in Figure 6.21 and the PID control in Figure 6.3. It is possible to determine that the LQR outperformed the PID in the height control task, with minimal overshoot and better stabilization time.

### 6.3.2 Experiment 3.B - MPC

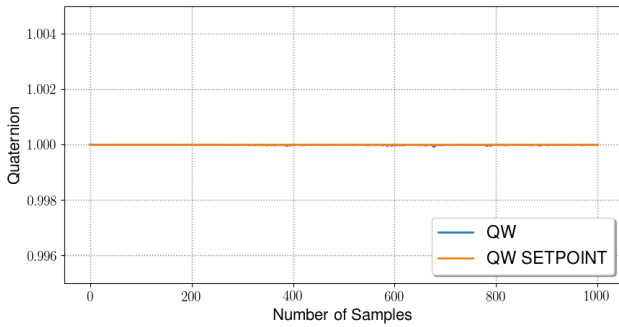
This section takes the scheme of experiment 1 and substitutes the PID controller for an MPC controller. Then, rerun the simulations and gather the same information from experiment 1.

#### Experiment 3.B.A - MPC Performance

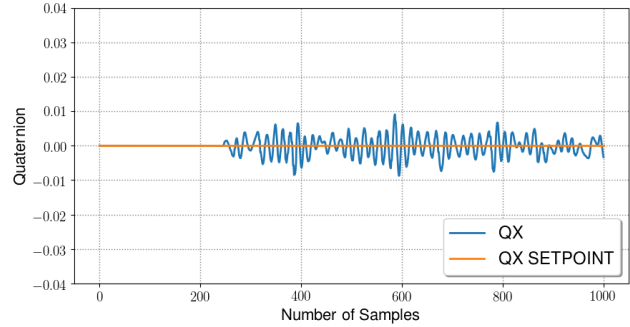
Here, it is set the the setpoint for the quadrotor stabilization in  $q = \{1, 0, 0, 0\}$ , which represents the configuration: roll angle ( $\phi$ ) =  $0^\circ$ , pitch angle ( $\theta$ ) =  $0^\circ$  and yaw angle ( $\psi$ ) =  $0^\circ$ , the same as in Section 6.1.2. Figure 6.22 shows the results from this test.

To guarantee that the latency was kept under 30ms, which would seriously affect the system's capability to perform the control, the prediction horizon was set in  $N_p = 5$ . The

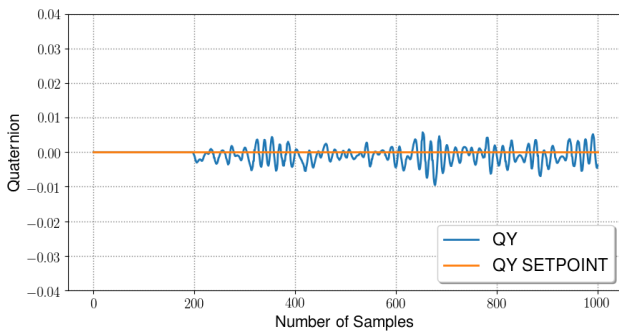
control horizon was set in  $N_c = 2$ . Performing some tests was determined that anything above this configuration would cause congestion in the NoC.



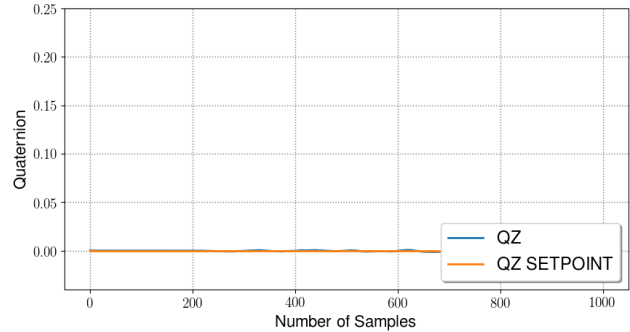
(a) Representation of "QW" quaternion component resulting from the MPC control.



(b) Representation of "QX" quaternion component resulting from the MPC control.



(c) Representation of "QY" quaternion component resulting from the MPC control.



(d) Representation of "QZ" quaternion component resulting from the MPC control.

Figure 6.22: Results of Experiment 3.A.A - Graphical representation of the measured quaternion of the MPC controller output.

Graphically analyzing the Figure 6.22, it is possible to deduce: that although the MPC control kept the quadrotor steady in the yaw angle, roll and pitch presented considerable jitter, which indicates marginal stability. The high latency of this controller, around 28ms, is the cause of this effect.

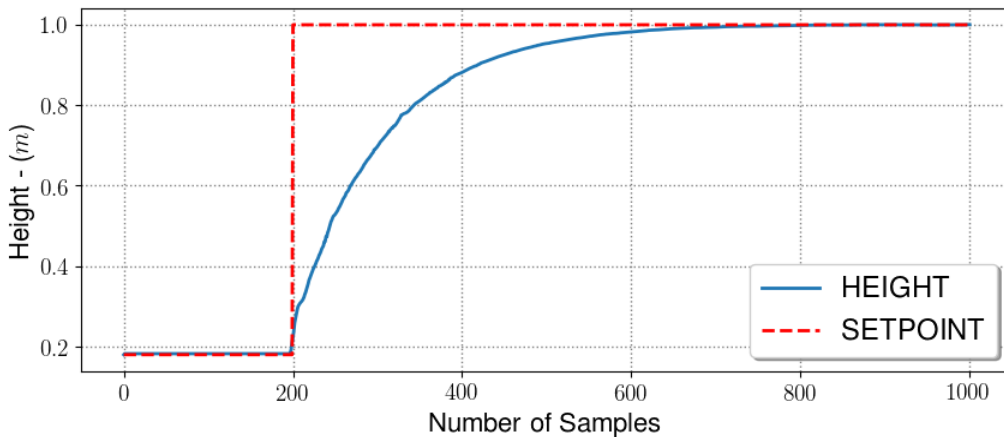


Figure 6.23: Results of Experiment 3.B.A - Height measured in meters representing the output of MPC controller.

To demonstrate the prediction aspect of the MPC, instead of starting the simulation with the setpoint in 1 meter, the setpoint was described as a step function, where  $h_{setpoint} = 0$  when  $sample \leq 200$  and  $h_{setpoint} = 1$  when  $sample > 200$ . Based on the results of the height control in Figure 6.23 and the LQR control in Figure 6.21. It is possible to determine that the MPC did not outperform the LQR in the height control task. Although it did not present overshoot, the stabilization increased. Also, its effects were not apparent due to the small prediction window.

### 6.3.3 Experiment 3.C - LQR and MPC - Energy Estimation

As for the energy consumption, the power used to run each task by the processor is represented in Figure 6.24. In this graphic, we compare the LQR and MPC task's energy consumption this the PID task. Based on the analysis of the figure, it is possible to determine that the PID consumes around 140nJ, the LQR consumes around 525nJ, and the MPC consumes around 5000nJ.

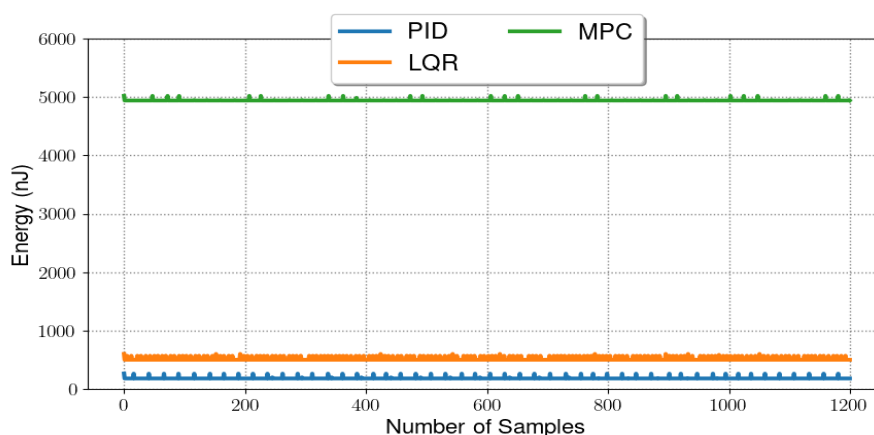


Figure 6.24: Energy estimation results by algorithm within cores.

After all these analyses, it is possible to conclude that the LQR, in this case, has the best cost benefits of the selected controllers.

## 6.4 Experiment 4 - Fault Tolerance on PID Control

This experiment takes the decentralized nature of the MPSoC and addresses the topic of fault tolerance. In the tests performed in Sections 6.4.1, 6.4.2 and 6.4.3, this section takes propose three possible recovery tactics for a task "freezing" while running in a core. These tests are based on migrating the control task from the "frozen" core to a new one.

Referring to Figure 6.25 to better understand this idea. This diagram shows the framework of an EKF-PID control system. Core zero still runs the input-output communication with the ROS system. However, it is now also responsible for detecting a fault in the PID controller output and determining the task migration path. The system must preload all tasks into their respective cores to set up all experiments. As for the previous tests, core zero and core one run ROS Comm. and EKF. Moreover, core two and core three are preloaded with the PID tasks.

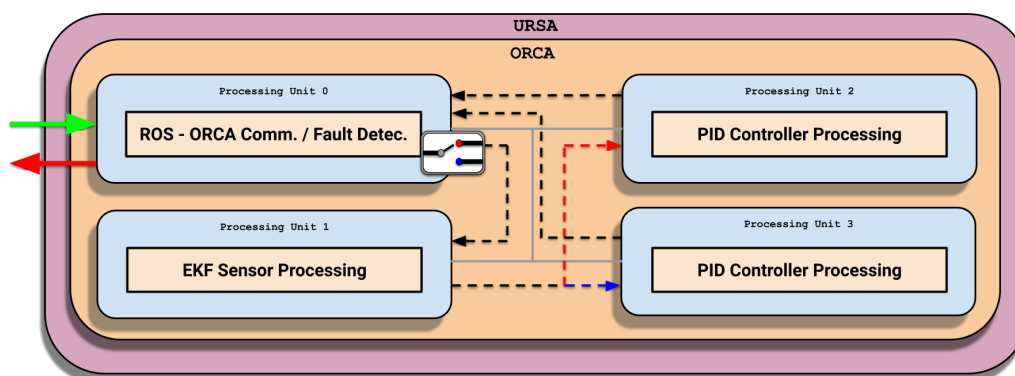


Figure 6.25: Controller Migration Implementation Diagram.

In these experiments, the fault is generated in the PID core. As a PID task starts, the PWM signal is published with a sequential serial number. For each iteration of the PID task, this serial number is increased by 1. Due to the nature of the ROS topics and messages, this robotic operating system keeps publishing the last sent information if a task stops running.

Then, core zero subscribes to the topic publishing this serial number to detect the fault and act on it. Suppose its information is the same for the last few iterations. In that case, the system detects a fault, and all the data flow are diverted to a functioning PID core.

To approach the topic of control redundancy, it is necessary to address the topic of control context. To illustrate this point: let us imagine a PID controller running in the quadrotor. As the quadrotor is stable in hover mode, it uses information generated during the whole process, like the sum of the errors and the error derivatives. In case of a forced reset, this information would be lost, the system would exit this stable state, and the physical consequences would be unknown. In this thesis, this information is called "control context."

#### 6.4.1 Experiment 4.A - PIDs Running in Parallel

This first experiment tests the idea of migrating the control tasks with the respective cores having the same control context. To achieve that, the PID runs the task concurrently in two cores. As the process runs, the core responsible for the EKF sends its output data

for cores two and three simultaneously. Therefore, ensuring that both PIDs have the same context at all times.

Although both cores are running the PID task, only one of them is publishing the information. That is determined by core zero, as explained previously. Figure 6.26, shows a diagram of this experiment.

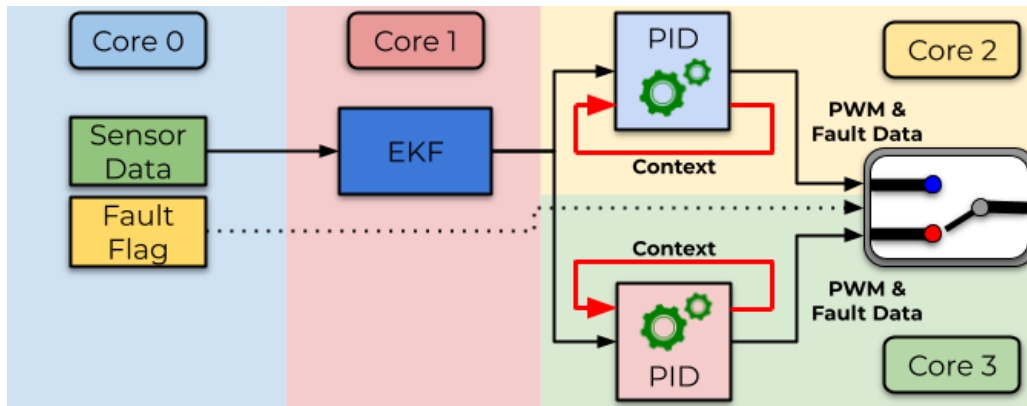


Figure 6.26: Diagram of Experiment 4.A, where two cores run PID tasks in parallel.

The ROS system does not consume the faulted core output when detecting a fault. And switches to the other core's output. As the same for the previous hovering tests, it is set the setpoint for the quadrotor stabilization in  $q = \{1, 0, 0, 0\}$ . Figure 6.27 shows the results from this test.

In these tests, the fault occurs in the sample of number 500. Moreover, graphically analyzing Figure 6.27, it is possible to deduce: that the PID control kept a consistent result with the previous PID tests, showing a minor variation in roll and pitch, and a slight drift in yaw.

The change of the cores did not cause an apparent effect on this test.

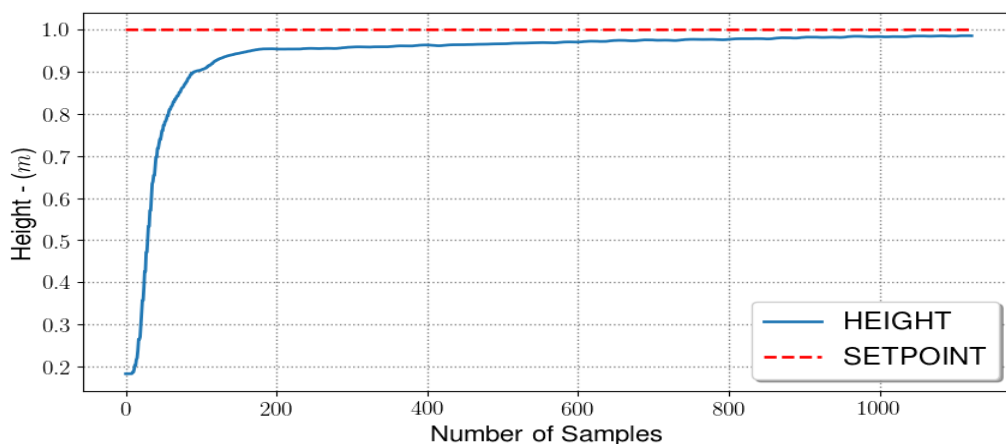
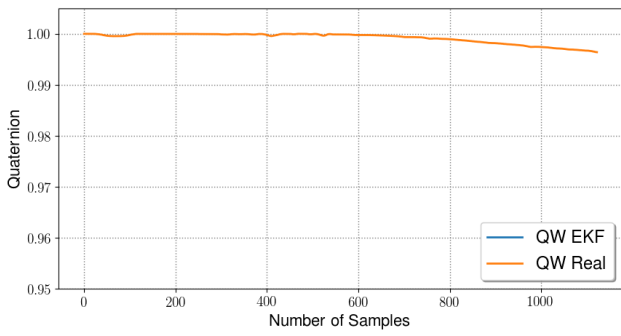
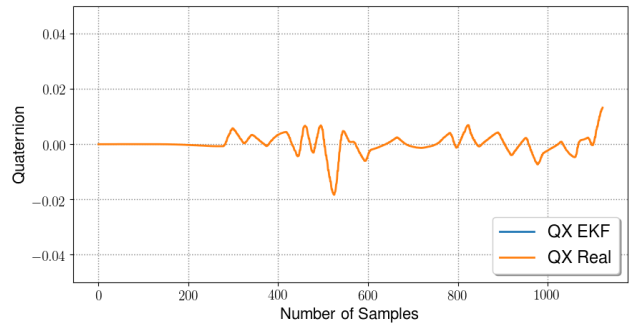


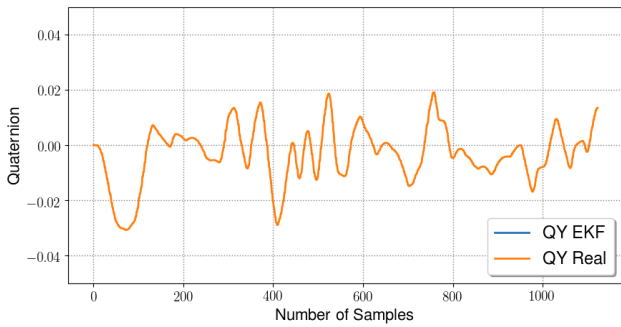
Figure 6.28: Results of Experiment 4.A - Height measured in meters representing the output of PID controller running in parallel cores.



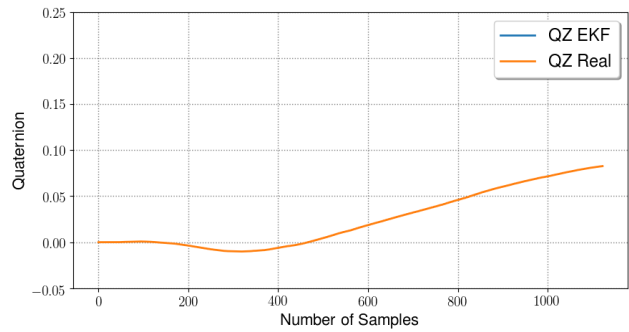
(a) Representation of "QW" quaternion component resulting from the PID control running in parallel cores.



(b) Representation of "QX" quaternion component resulting from the PID control running in parallel cores.



(c) Representation of "QY" quaternion component resulting from the PID control running in parallel cores.



(d) Representation of "QZ" quaternion component resulting from the PID control running in parallel cores.

Figure 6.27: Results of Experiment 4.A - Graphical representation of the measured quaternion of the PID controller the running in parallel cores.

The PID's height control presented a similar performance to the previous PID tests, showing no overshoot and a longer stabilization time.

The change of the cores also did not cause an apparent effect on this test.

Figure 6.29 shows the energy estimation results for each of the PID tasks' cores.

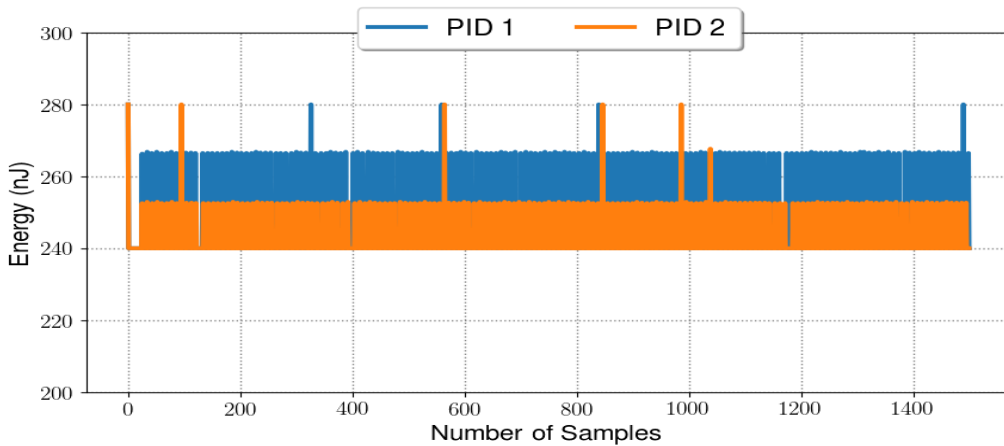


Figure 6.29: Energy estimation results by algorithm within cores.

Since the two cores are running the same data at all times, they have similar energy profiles. The only difference is that the core one also runs a function to create the fault, as mentioned earlier. Core 1 consumes about 270nJ, while core 2 consumes about 250nJ.

#### 6.4.2 Experiment 4.B - PIDs - No Context Migration

As we determined in the experiment presented Section 6.4.1. The performance results were satisfactory, but the energy consumption overhead gives that solution a significant drawback. This section tests the possibility of keeping one core idle without context. Then, when the fault occurs, the system switches to the idle core and observes if the system continues in a stable state, saving on energy.

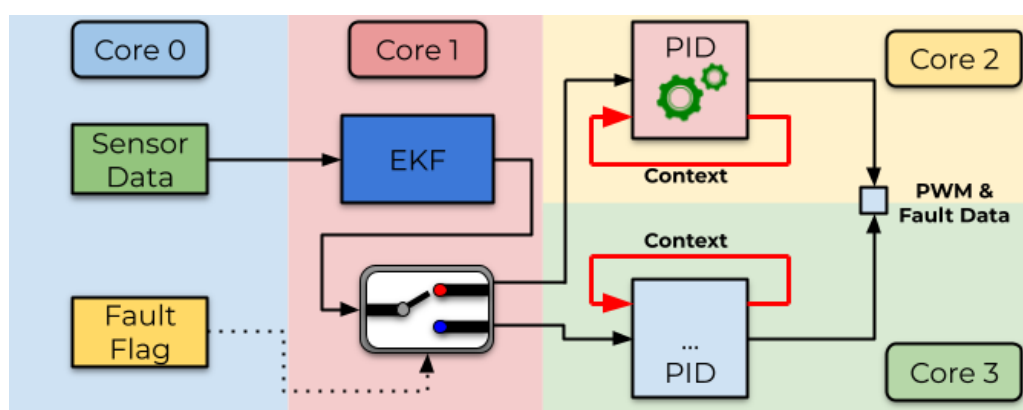
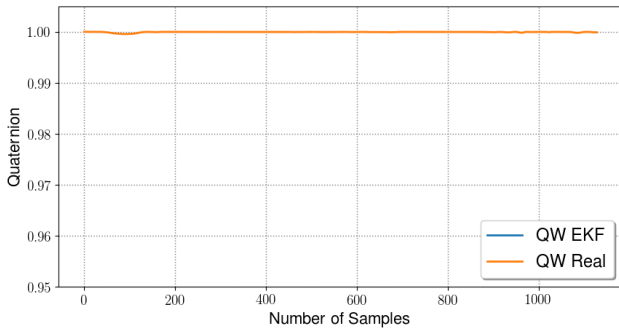


Figure 6.30: Diagram of Experiment 4.B, where two cores run PID tasks with no context.

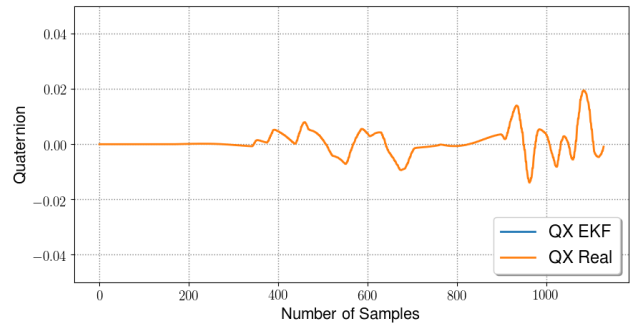
When detecting a fault, core 0 sends a flag to core 1, determining which core receives the EKF information. The core that does not receive the EKF information is kept idle. As the same for the previous hovering tests, it is set the setpoint for the quadrotor stabilization in  $q = \{1, 0, 0, 0\}$ . Figure 6.31 shows the results from this test.

In these tests, the fault occurs in the sample of number 500. Moreover, graphically analyzing Figure 6.31, it is possible to deduce: that the PID control kept a consistent result with the previous PID tests, showing a minor variation in roll and pitch, and a slight drift in yaw.

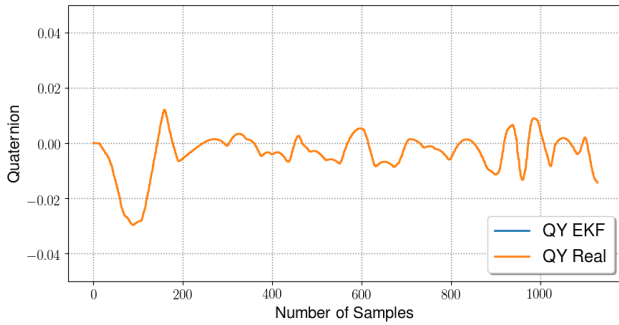
The change of the cores did not cause an apparent effect on the quaternion control. Figure 6.32 shows the results of the fault injection in height control.



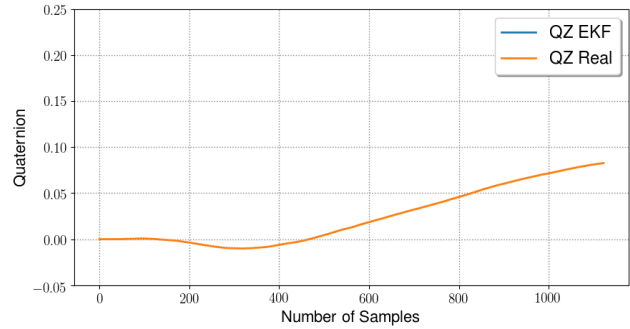
(a) Representation of "QW" quaternion component resulting from the PID control running in separate cores with no context migration.



(b) Representation of "QX" quaternion component resulting from the PID control running in separate cores with no context migration.



(c) Representation of "QY" quaternion component resulting from the PID control running in separate cores with no context migration.



(d) Representation of "QZ" quaternion component resulting from the PID control running in separate cores with no context migration.

Figure 6.31: Results of Experiment 4.A - Graphical representation of the measured quaternion of the PID controller the running in separate cores with no context migration.

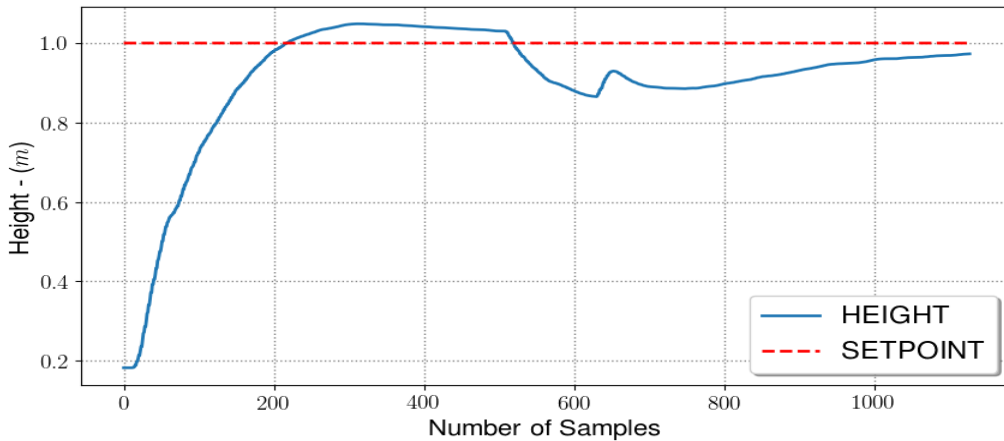


Figure 6.32: Results of Experiment 4.B - Height measured in meters representing the output of PID controller running in separate cores with no context migration.

Here, it is possible to conclude that the height control was severely affected by the change in processors. The quadrotor fell almost 0.20 meters before starting to return to the setpoint.

Figure 6.33 shows the energy estimation results for each of the PID tasks' cores.



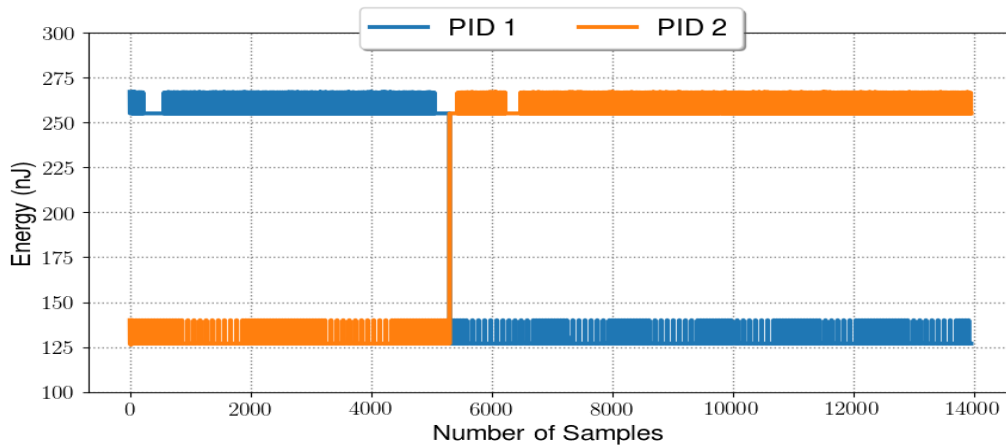


Figure 6.33: Energy estimation results by algorithm within cores with no context migration.

While one core is running, it is consuming about 260nJ, and the core kept idle is consuming about 130nJ.

### 6.4.3 Experiment 4.C - PIDs - Context Migration

This experiment was devised to mitigate the result shown in the previous section without creating a significant energy overhead. Then, this experiment proposes that the backup core is kept idle; however, the control context is published with the PID output data. Then, when the task is migrated, the idle core have the necessary information to keep the system in a stable state.

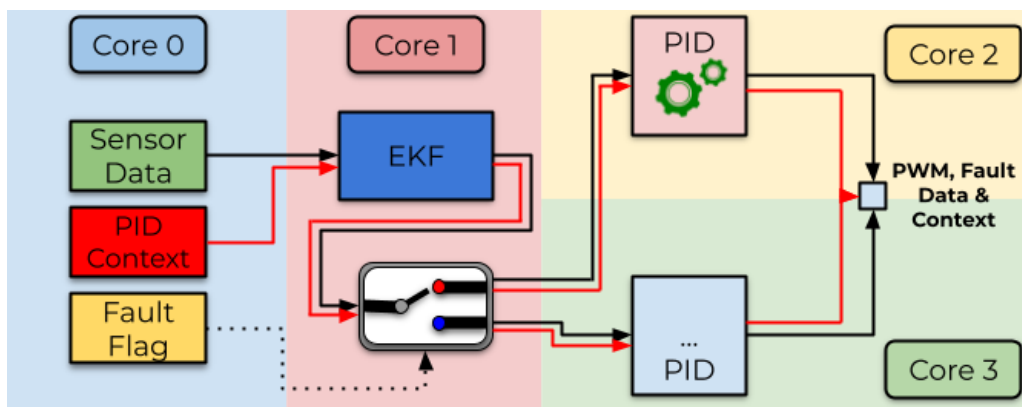
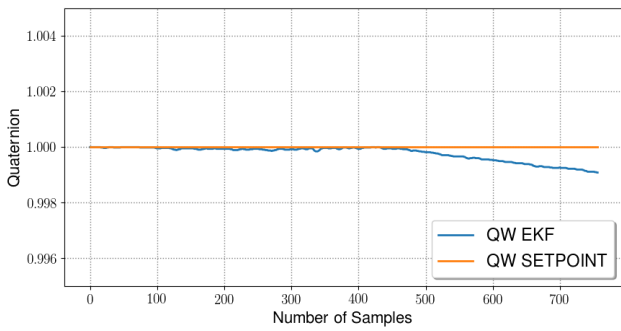


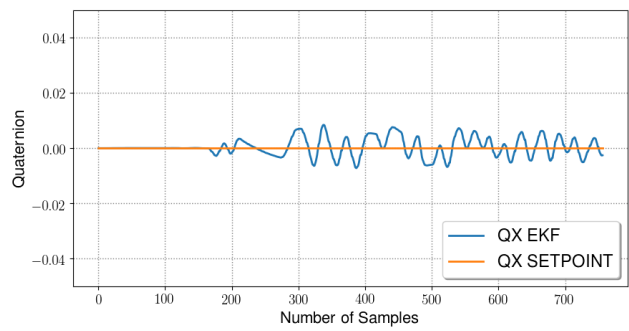
Figure 6.34: Diagram of Experiment 4.C, where two cores run PID tasks with context.

When detecting a fault, core 0 sends a flag to core 1 plus the control context, determining which core receives the EKF information. The core that does not receive the EKF information is kept idle. As the same for the previous hovering tests, it is set the setpoint for the quadrotor stabilization in  $q = \{1, 0, 0, 0\}$ . Figure 6.35 shows the results from this test.

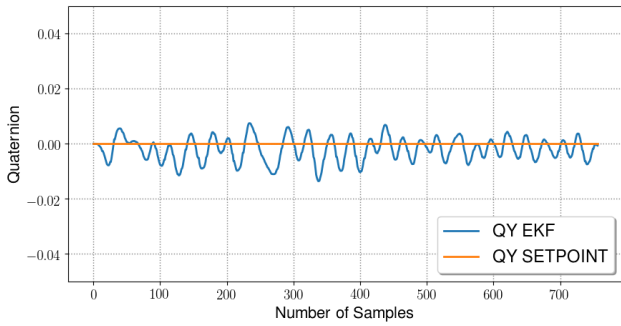
In these tests, the fault occurs in the sample of number 500. Moreover, graphically analyzing Figure 6.35, it is possible to deduce: that the PID control kept a consistent result



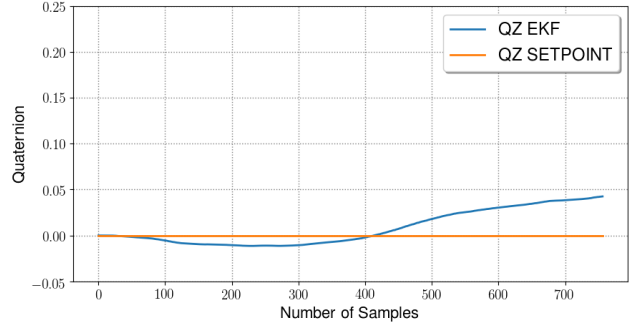
(a) Representation of "QW" quaternion component resulting from the PID control in separate cores with context migration.



(b) Representation of "QX" quaternion component resulting from the PID control running in separate cores with context migration.



(c) Representation of "QY" quaternion component resulting from the PID control running in separate cores with context migration.



(d) Representation of "QZ" quaternion component resulting from the PID control running in separate cores with context migration.

Figure 6.35: Results of Experiment 4.A - Graphical representation of the measured quaternion of the PID controller the running in separate cores with context migration.

with the previous PID tests, showing a minor variation in roll and pitch, and a slight drift in yaw.

The change of the cores did not cause an apparent effect on the quaternion control.

Figure 6.36 shows the results of the fault injection in height control.

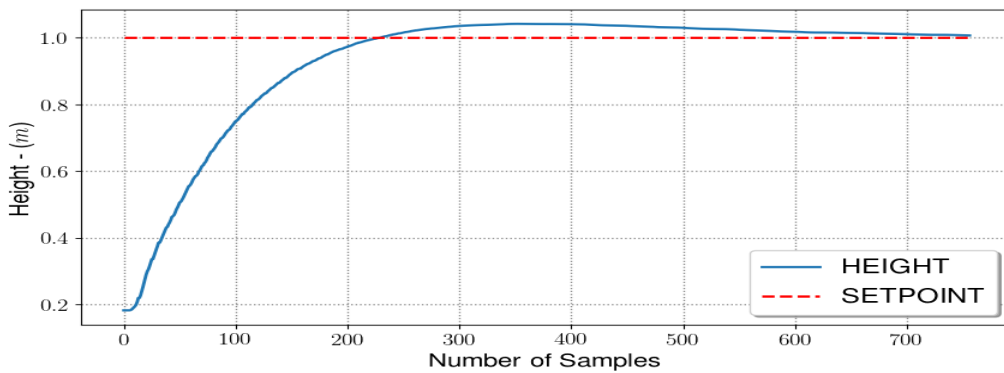


Figure 6.36: Results of Experiment 4.A - Height measured in meters representing the output of PID controller running in separate cores with context migration.

The PID's height control presented a similar performance to the successful PID tests, showing a slight overshoot and a longer stabilization time.

The change of the cores did not cause an apparent effect on this test.

Figure 6.37 shows the energy estimation results for each of the PID tasks' cores.

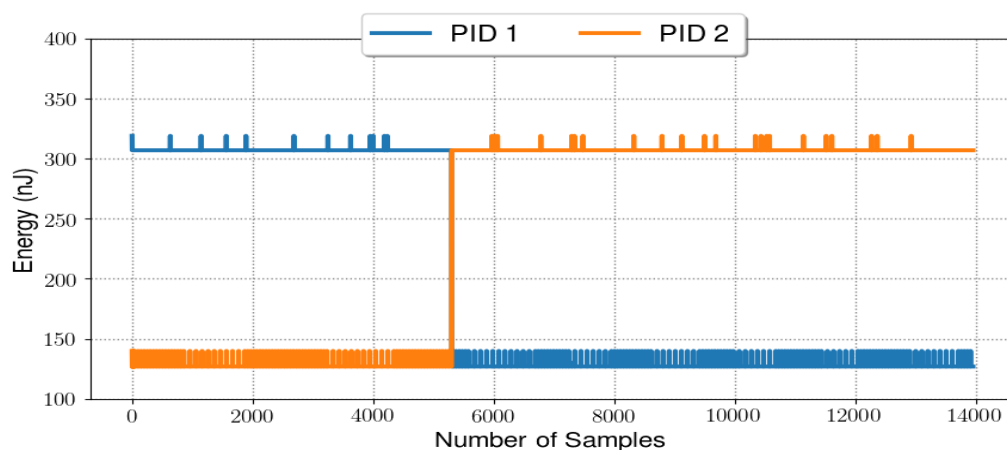


Figure 6.37: Energy estimation results by algorithm within cores with no context migration.

While one core is running, it is consuming about 320nJ, and the core kept idle is consuming about 130nJ. This increase in energy consumption in the active core is due to the processing needed for the extra data from the context.

## 6.5 Experiment 5 - Energy Management

This section proposes a software implementation of an energy management technique. This technique does not require that the user changes any parameters of the hardware settings, only changing the data input frequency.

As observed in the previous sections, the cores only run tasks once new data is incoming. Meanwhile, the core is kept idle, running only a function that detects if new data have arrived. Therefore, it is possible to control the overall energy consumed by the core by spacing the frequency of data injection into it. Figure 6.38 shows a diagram of this experiment.

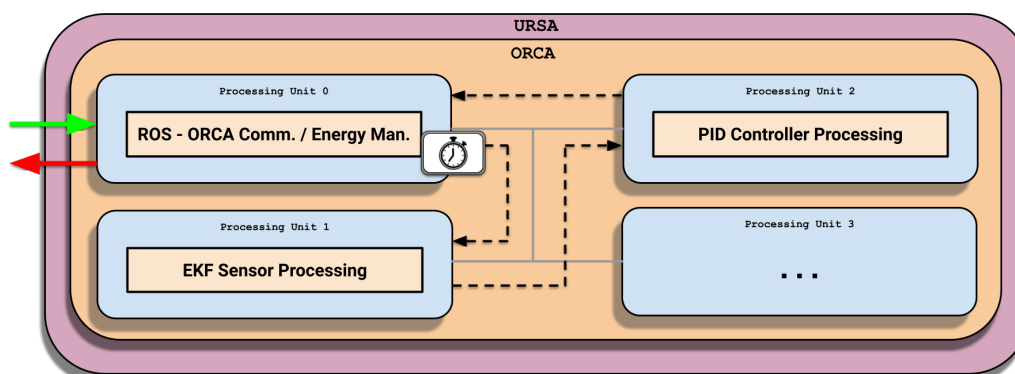


Figure 6.38: Energy Management Implementation Diagram.

This experiment is based on the idea that running the system at full power is not needed for every case. Let us take the example of the EKF: in hover mode, the changes on the Euler angles are minimal; therefore, running the attitude estimation at its maximum performance is unnecessary. Although the quadrotor at motion mode, the attitude estimation becomes much more significant, which demands a greater processing speed and more energy consumed.

As stated before, two modes of operation are developed: energy-saving mode and max power mode. In energy-saving mode, the data injection frequency into the EKF until the system decreases, saving energy without impacting the estimation quality while hovering. In max power mode, the frequency injection is maxed out. And then, it is performed four tests:

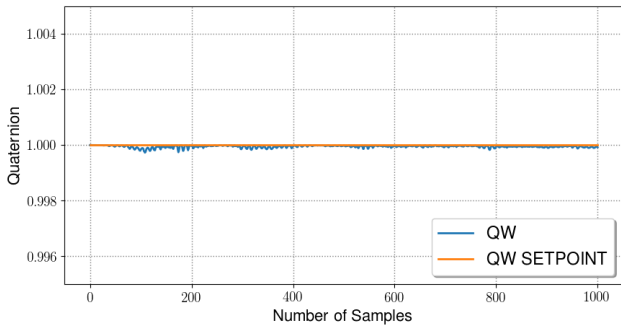
1. Energy management system is in saving mode while the quadrotor is hovering;
2. Energy management system is in max power mode while the quadrotor is hovering;
3. Energy management system is in saving mode while the quadrotor is in motion;
4. Energy management system is in max power mode while the quadrotor is in motion.

#### 6.5.1 Experiment 5.A - Saving Mode/Hovering

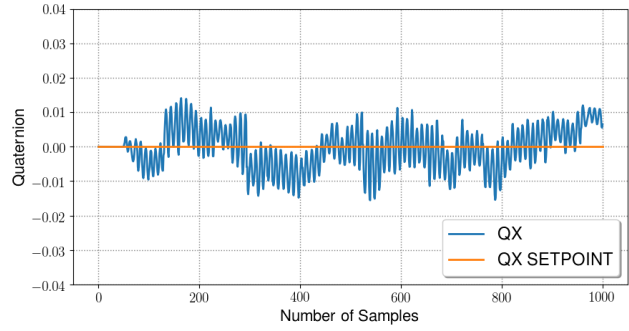
In this first experiment, it is fixed that the data injection rate in the EKF is 12Hz. That means that the core running the EKF is kept idle for longer, consuming less energy. However, this lower frequency reduces the control precision. Figure 6.39 shows the attitude quaternion output while the system runs on the saving mode and in hover.

Graphically analyzing Figure 6.39, it is possible to detect that this test presents the same results as the experiment of Figure 6.22. The high latency of this system could stabilize the yaw angle. However, roll and pitch presented considerable jitter, indicating marginal stability.

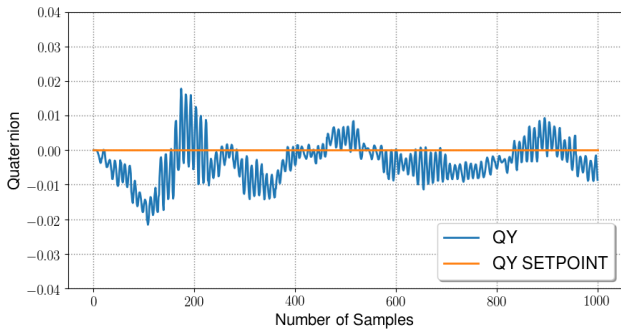
Figure 6.40 shows the results of the energy saving mode in the height control.



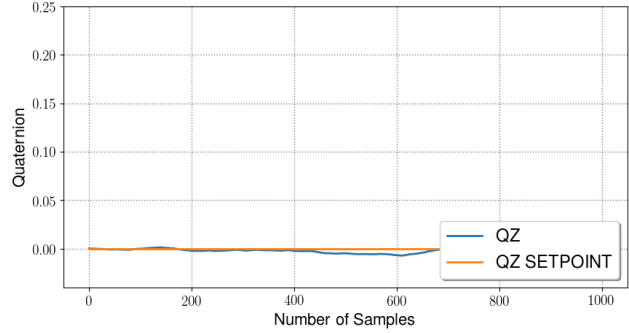
(a) Representation of "QW" quaternion component resulting from the PID control running in energy saving mode while hovering.



(b) Representation of "QX" quaternion component resulting from the PID control running in energy saving mode while hovering.



(c) Representation of "QY" quaternion component resulting from the PID control running in energy saving mode while hovering.



(d) Representation of "QZ" quaternion component resulting from the PID control running in energy saving mode while hovering.

Figure 6.39: Results of Experiment 5.A - Graphical representation of the measured quaternion of the PID controller the running in energy saving mode while hovering.

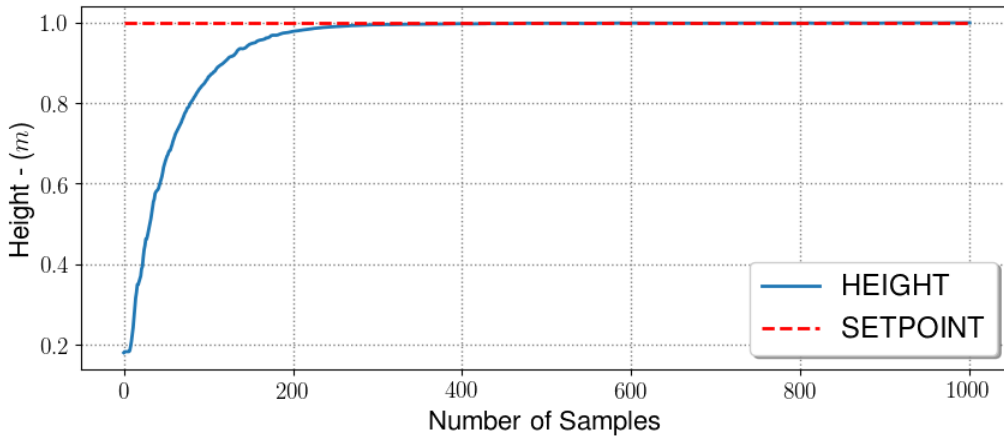
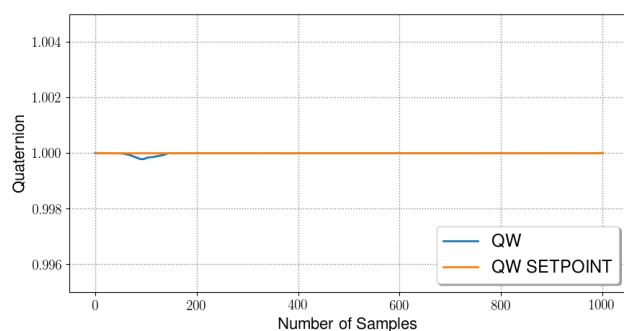


Figure 6.40: Results of Experiment 5.A - Height measured in meters representing the output of PID controller running in energy saving mode while hovering.

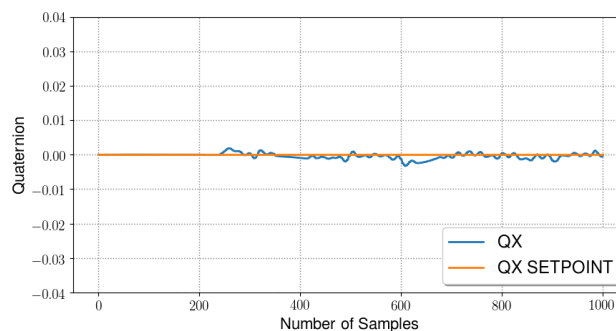
The height curve shows an adequate response time with no overshoot or steady-state error in this figure.

## 6.5.2 Experiment 5.B - Max Power Mode/Hovering

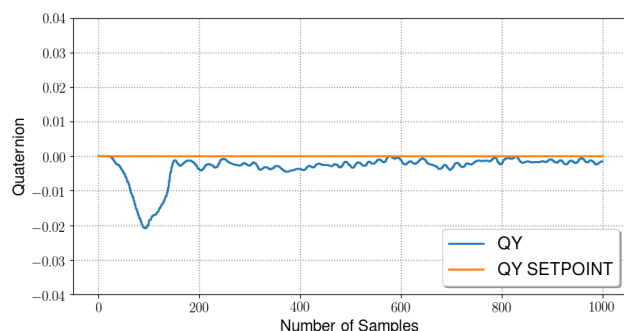
In this experiment, it is fixed that the data injection rate in the EKF is 40Hz. That means that the core running the EKF is kept idle for a shorter time, consuming more energy. However, this higher frequency increases the control precision. Figure 6.41 shows the attitude quaternion output while the system runs on the maximum energy mode and in hover.



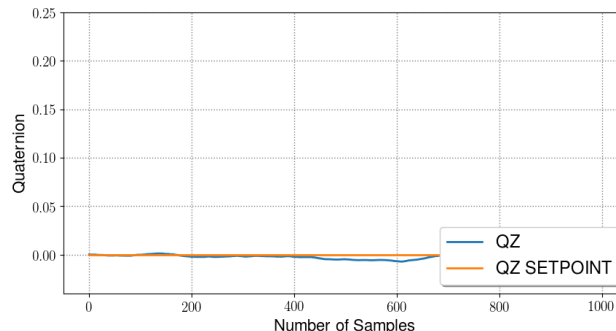
(a) Representation of "QW" quaternion component resulting from the PID control running in maximum energy mode while hovering.



(b) Representation of "QX" quaternion component resulting from the PID control running in maximum energy mode while hovering.



(c) Representation of "QY" quaternion component resulting from the PID control running in maximum energy mode while hovering.



(d) Representation of "QZ" quaternion component resulting from the PID control running in maximum energy mode while hovering.

Figure 6.41: Results of Experiment 5.B - Graphical representation of the measured quaternion of the PID controller the running in maximum energy mode while hovering.

Graphically analyzing the Figure 6.41, it is possible to detect a significant improvement between the test in Figure 6.39. The smaller latency of this system could stabilize the roll, pitch, and yaw angle, with little to no variation. However, between samples 0 and 200, it is possible to detect a larger error in component QY; the fast rise of the quadrotor explains this.

Figure 6.42 shows the results of the energy saving mode in the height control.

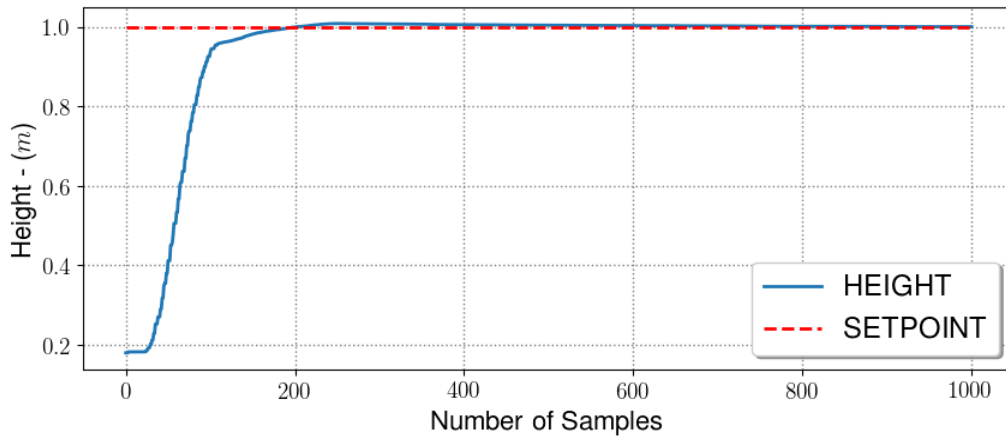
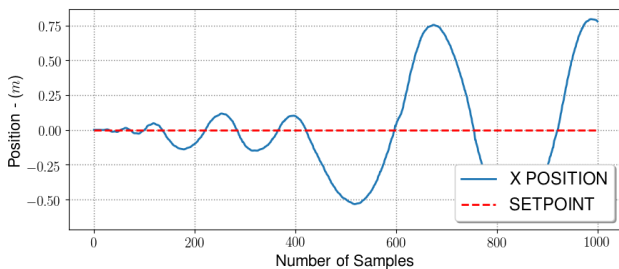


Figure 6.42: Results of Experiment 5.B - Height measured in meters representing the output of PID controller running in maximum energy mode while hovering.

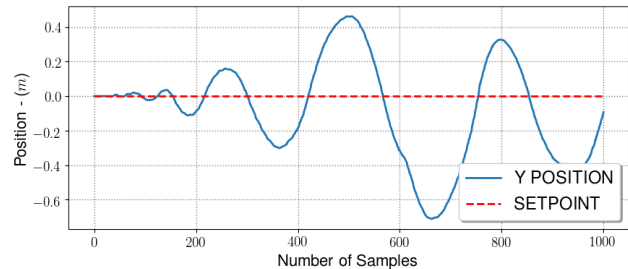
The height curve shows an adequate response time with no overshoot or steady-state error and a rise time faster than the saving energy mode.

### 6.5.3 Experiment 5.C - Saving Mode/Motion

This experiment tests the system's capability to keep the quadrotor in a fixed position during the flight in energy saving mode. Figure 6.43 shows the graphical results for each measured axis.



(a) X position control by the PID running in energy saving mode while in motion.



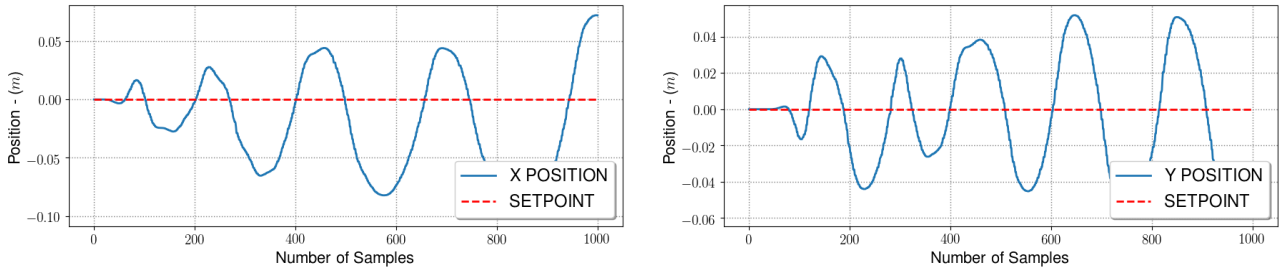
(b) Y position control by the PID running in energy saving mode while in motion.

Figure 6.43: Results of Experiment 5.C - XY position control by the PID running in energy saving mode while in motion

Based on these graphs, it is possible to determine that the system can not stabilize in the desired position. Moreover, this figure shows the maximum errors of the quadrotor's position to the setpoint. On axis X, the system presents a maximum error of 0.75 meters, and on the Y axis, a maximum error of 0.6 meters. The high latency of this system imposes unacceptable errors in this context.

#### 6.5.4 Experiment 5.D - Max Power Mode/Motion

This experiment tests the system's capability to keep the quadrotor in a fixed position during the flight in maximum energy mode. Figure 6.44 shows the graphical results for each measured axis.



(a) X position control by the PID running in maximum energy mode while in motion.

(b) Y position control by the PID running in maximum energy mode while in motion.

Figure 6.44: Results of Experiment 5.D - XY position control by the PID running in maximum energy mode while in motion

Based on these graphs, it is possible to determine that the system improved significantly over the last experiment. Moreover, this figure shows the maximum errors of the quadrotor's position to the setpoint. On axis X, the system presents a maximum error of 0.10 meters, contrasting with the 0.75 meters of the last experiment. And on the Y axis, a maximum error of 0.05 meters, reduction from 0.6 meters. The low latency of this system improved this control significantly.

#### 6.5.5 Experiment 5 - Energy Analysis

This section reports the energy estimation for both flight modes: energy saving and maximum energy.

Figure 6.45 shows the accumulated energy estimated in the core running the PID task in energy saving mode.



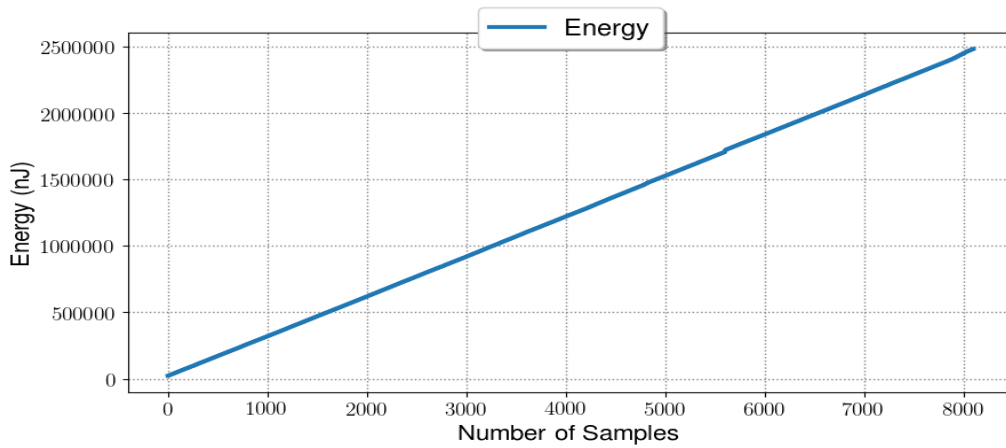


Figure 6.45: Accumulated energy estimated in the core running the PID task in energy saving mode.

This figure shows that the entirety of the experiment in saving mode consumed 2500000 nJ.

Figure 6.46 shows the accumulated energy estimated in the core running the PID task in maximum energy mode.

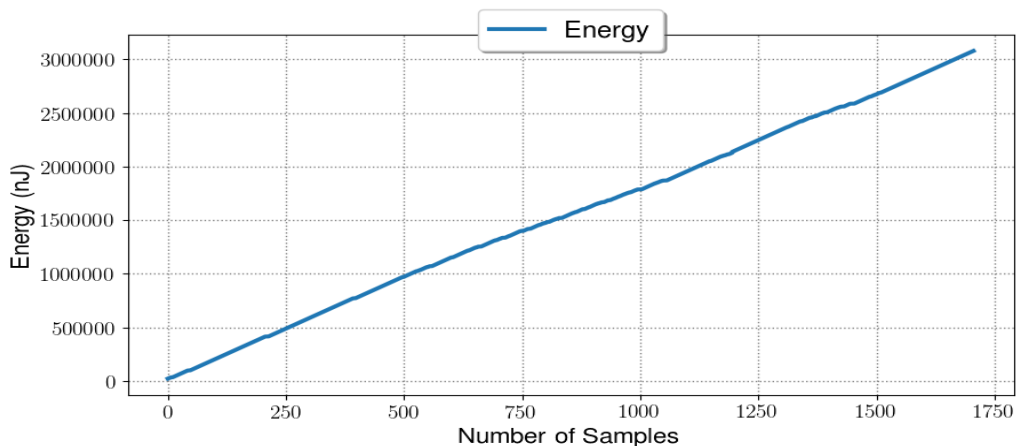


Figure 6.46: Accumulated energy estimated in the core running the PID task in maximum energy mode.

This figure shows that the entirety of the experiment in saving mode consumed 3 mJ. The analysis of these two figures shows that it is possible to save almost 20 percent in energy consumption only by changing the data injection rate in one core.

In practical terms, it shows the possibility of this system to operate in several modes, depending on the application and power and performance requirements. The case just showed: that it is possible to operate in saving mode while in take-off and hover. When it is needed to move, the maximum power setting is triggered.

## 6.6 Experiment 6 - Controller Online Adaptation

This section explores another feature that the decentralization nature of the MPSoC provides: the possibility of runtime controller adaptation. An external function updates a functioning controller to better adapt to external conditions. Section 3.9.1 already discussed the theory of the Fuzzy controller, and here it is applied to the framework.

The last section presented the energy management functionality of this framework. There, the injection frequency is changed into the core running the EKF. However, it is essential to mention that this change affects the EKF and the following processes. These tests constantly changes the values of the PID gains to stabilize the quadrotor for each computing demand.

With the proposed algorithm, the Fuzzy controller would mitigate small changes in the frequency. Figure 6.47 shows a diagram of this functionality.

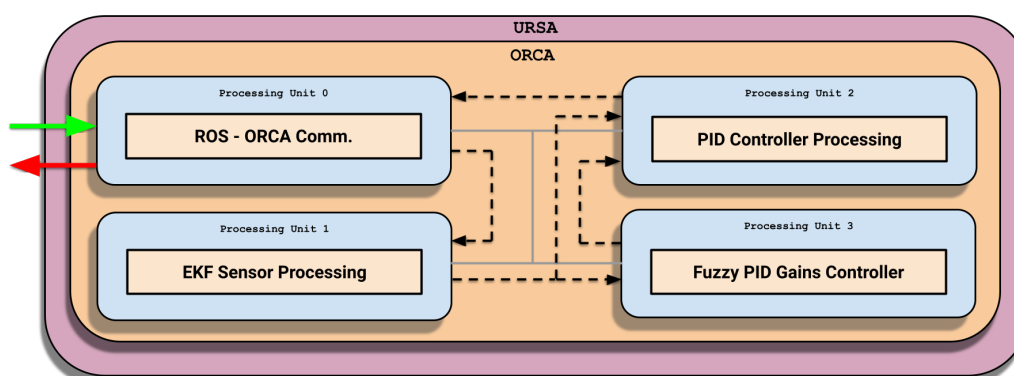


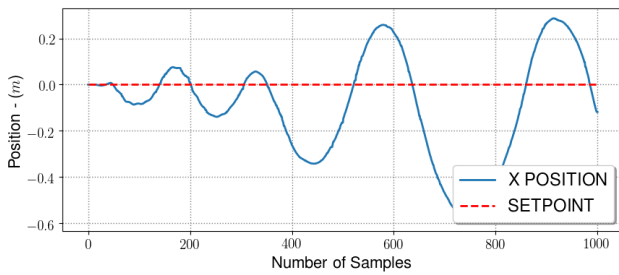
Figure 6.47: Controller Online Adaptation Implementation Diagram.

Section 6.6.1 repeats the experiment of Section 6.5.3, but now with the fuzzy controller.

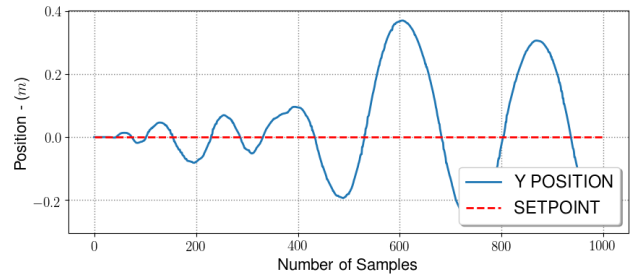
### 6.6.1 Experiment 6 - Saving Mode/Motion - With Fuzzy Control

This experiment tests the system's capability to update the PID gains during runtime. This test utilizes the setup from Section 6.5.3 where the system tries to stabilize in a fixed position while in the energy saving mode. Figure 6.44 shows the graphical results for each measured axis.

In the experiment of Section 6.5.3, the mean errors of the controller were: 0.27 meters for the X axis and 0.23 meters for the Y axis. Analyzing the numerical values and graphics of Figure 6.48 of experiment 6, the mean errors were reduced to 0.18 meters on the



(a) X position control by the PID running with a parallel Fuzzy controller.

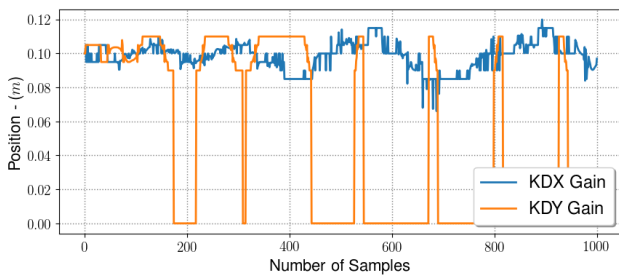


(b) Y position control by the PID running with a parallel Fuzzy controller.

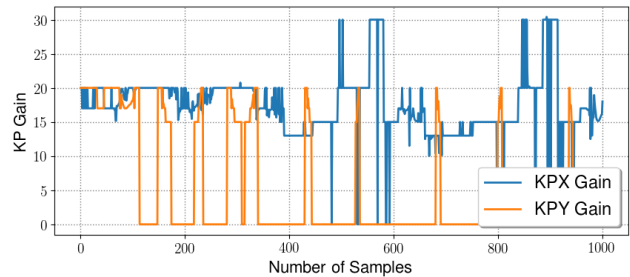
Figure 6.48: XY position control by the PID running with a parallel Fuzzy controller.

X axis and 0.12 meters on the Y axis. They were also significantly reducing the maximum errors.

Figure 6.49 shows the calculated KP and KD gains for each XY axis.



(a) Calculated KD gains for PID controller. KDX refers to the KD gain for the X position control. KDY refers to the KD gain for the Y position control.



(b) Calculated KP gains for PID controller. KPX refers to the KP gain for the X position control. KPY refers to the KP gain for the Y position control.

Figure 6.49: Calculated KP gains for PID controller by the Fuzzy controller running in parallel to the PID task.

The KI gains were not included in this analysis because the values were relatively low and did not present significant variations.

## 6.7 Final Remarks

This chapter proposed six different experiments to validate the thesis's primary goals. These experiments were developed based on the theory of Chapter 3, the hardware infrastructure presented in Chapter 4 and the original work of Chapter 5.

Section 6.1 presents a PoC of this system, evaluating the precision of the algorithms like the EKF and PID. It also justifies the choice of utilizing a fixed-point format and multiplier unit. The results showed that the fixed-point and multiplier unit was the most efficient setup and yielded the best results regarding sensor fusion, attitude, and height con-

trol. In both setups, the difference between the output of the system and actual values was acceptable.

Section 6.2 presents the first proposed feature in the thesis' goals. The decentralization of the control system. The first experiment process the sensor fusion for attitude and XY position estimation in a single task. It checks the performance for attitude estimation, XY, and height control. This experiment served as a baseline for the next one. Since the idea of the experiment is to test the software framework capability of decentralization, the tasks of the EKF and KF were separated into two cores running in parallel. These experiments not only showed that the software framework was capable of doing so, but it increased the performance of the whole control system.

Section 6.3 presents the capabilities of the software framework in implementing more complex control schemes. The LQR and MPC control were chosen for these tests. Both applications showed that the software framework was capable of implementing functional controllers that were more complex than the PID. The only caveat of these experiments was the limitations imposed by the experimental setup regarding the calculations of larger matrices required by the MPC. This problem can be mitigated for future works by implementing this software framework in a more robust setup.

Section 6.4 presents the topic of fault tolerance in the software framework. This experiment is constituted of three alternative fault tolerance setups. All of these sub-experiments deal with the case of a processor core running a quadrotor controller "freezing" mid-run. The first sub-experiment proposes running two PID controllers in parallel in separate cores. This experiment shows that the performance is not affected by a fault. However, it consumes more energy than the other solutions. The second solution proposes that while one processor core is running the controller, a second is kept idle until the fault in the first processor core is detected. When the second core starts to run, it has no context to the first processor core state, so it tries to stabilize the quadrotor with the wrong information. That fact led to a transitory destabilization. The only advantage of this method over the first was the energy consumption reduction. Lastly, the third sub-experiment treated the previous drawbacks. It solve the issue of energy consumption, as in the 2nd FT setup, by keeping the redundant processor core idle. Moreover, it addresses the destabilization issue by transmitting the context information with the control and sensor signals.

Section 6.5 presents the topic of energy management. Here, four experiments were devised to determine modes of operation regarding energy consumption and performance. This section aims to show that this software framework can implement a software solution for energy management. The first experiment shows the results when the data injection rate is lowered when the quadrotor is hovering, which showed a satisfactory result. However, the second experiment significantly improved when it ran on the maximum frequency. The third and fourth experiments showed the results for the position controller in the two energy modes. The energy-saving mode showed itself not to be suitable for this purpose. At the

same time, the maximum power mode showed a satisfactory result. All of these experiments present the possibility of energy management for different applications; for example: while in hover, the processor can function in an energy saving mode, and when it needs to control its position, it triggers the maximum power mode.

Lastly, Section 6.6 presents the capability of the software framework to create a system capable of updating the controller based on the immediate requirements. In this case, a fuzzy controller updates the PID controller gains mid-run. The results showed that this system performed well when applied to the problem found in experiment 5.

## 7. CONCLUSION AND FUTURE WORK

This Thesis stated the following hypothesis: *"Applying digital control systems into a heterogeneous computing framework increases control efficiency and provides the possibility to apply other techniques, like decentralized control and controller self-adaptation. Also this is expected to be achievable while taking into account parameters such as energy consumption and fault tolerance."*

In order to validate the hypothesis of the Thesis, a software framework was proposed as the initial step, utilizing an MPSoC platform intended for integration into a robotic application.

The quadrotor UAV was selected as the study case for the robotic application in this research due to its suitability for the proposed idea presented in the Thesis. Quadrotors require robust processing and are sensitive to faults due to their fast dynamics, and benefit from energy management techniques common to mobile robotic applications. Additionally, the nonlinear nature of quadrotors requires robust controllers. To implement the quadrotor, a computational simulation was chosen, allowing for rapid implementation of the software framework and low-cost testing. However, the simulation adds computational demands to a system already simulating another process. Based on relevant literature, three controllers were selected for testing, including the most commonly used controllers and a set of controllers varying in complexity.

The hardware selected for this Thesis was an MPSoC, chosen for its ability to support real-time embedded applications with low power consumption, small size, and the ability to handle multiple processes simultaneously. The RISC-V architecture was selected for modeling the MPSoC, as it offers a realistic ISA suitable for direct hardware implementation, a small but complete base ISA to avoid over-architecting, and support for highly-parallel multicore or manycore implementations, including heterogeneous multiprocessors. The ORCA platform was chosen as the development platform for the software framework design, as it provides abstractions for self-adaptation complexity, including a configurable hardware architecture, operating system, and software libraries. To simulate and model the ORCA MPSoC, the URSA application programming interface was selected.

Once the base architecture and robotic application were established, the software framework was developed. A communication infrastructure was constructed on the ORCA MPSoC and ROS operating system to enable data transmission between the processor and application via the ROS publish-subscribe protocol and UDP communication. This allowed for the exchange of information between the processor and quadrotor. In addition, mathematical libraries were developed to support the project's control libraries, including matrix operations and fixed-point math.

In consideration of time constraints, the initial experiment involved a proof-of-concept test consisting of a basic setup of the EKF + PID combination. The objective was to determine the optimal hardware configuration and number format to minimize latency when running a specific task, while also measuring the precision, effectiveness, and performance of the control algorithms. The results indicated that configuring the Multiplier Unit and utilizing fixed-point format was the most efficient choice, providing acceptable performance for the control algorithms.

An experiment was conducted to evaluate the ability of the software framework to support a decentralized control paradigm (Sensing + Control + Actuation) within the MP-SoC architecture. To achieve this, a comparison was made between a traditional control system implementation, where all designs were executed on a single central processor, and a distribution of tasks across multiple processors. The results demonstrated that although energy consumption was not improved, there were significant gains in sensor fusion precision and control of the quadrotor's attitude, height, and XY position.

The third experiment was designed to evaluate the MPSoC's ability to support controllers with more complex algorithms than PID, as well as to assess the software framework's capability to implement such controllers. The experiment demonstrated that the controller could run on the MPSoC with acceptable results, indicating the software framework's feasibility for implementing more complex controllers.

An experiment was conducted to investigate fault tolerance in the software framework. The experiment involved inducing a "freeze" on the PID controller's processor, and three potential solutions were proposed to mitigate the sudden stop in the process. The first solution involved running two redundant processors simultaneously, receiving the same data from the EKF process. When a fault occurs in one processor, the system starts to read the output from the other. This solution yielded satisfactory control performance but consumed the most energy. The second solution proposed keeping one processor idle while the other runs the PID control, with the idle processor waiting for a flag indicating a fault in the active processor. This solution had the best energy consumption profile, but the quadrotor momentarily destabilized during the fault. The third solution involved transmitting the control context with the sensor and controller data to the idle processor so that it could start with the necessary information to avoid destabilizing the system. This configuration proved to be the best solution, with the same performance as the parallel configuration but with slightly higher energy consumption than the second experiment.

The implementation of energy management was tested using an application that offers users two operational modes. The first mode maintains a low data injection rate into the processor, thereby keeping the controller stable and idle for longer periods to save energy. The second mode increases the data injection rate to the processor as much as possible without causing congestion in the NoC, resulting in better performance at a higher energy cost. This approach can be applied to various robotic demands, including the quadrotor,

where the processor can run on lower energy during low-intensity activities, such as hovering. In contrast, the energy supply can be increased during more demanding tasks. The impact of energy management on the quadrotor system was found to be insignificant. However, the goal of this thesis is to develop a software framework that can be applied to different robotics applications, including those that involve computationally intensive tasks such as computer vision and neural networks that may result in higher energy demand, which this software framework can mitigate.

In this thesis, a new feature was proposed whereby an application can run in a separate processor capable of controlling and updating a robotic control application running in parallel. Specifically, a fuzzy controller was introduced to run in parallel with the PID controller, allowing for real-time updates to the PID gains depending on the errors and derivative errors of the quadrotor. The experiment aimed to demonstrate the potential performance improvement of the PID controller compared to a previous test, where the processor's energy consumption had been reduced.

The aforementioned paragraphs validate the propositions put forward in the Thesis hypothesis. Through these experiments, it was demonstrated that integrating digital control systems into a heterogeneous computing framework utilizing MPSoC resulted in improved control efficiency, enabling the application of decentralized control and controller self-adaptation. Furthermore, the framework allowed for effective management of energy consumption and fault tolerance within the system.

## 7.1 Future Work

As a guideline for future works, this Thesis has room for improvements as follows:

- Improve performance of the most demanding algorithms;
- In this Thesis the combination of many or all features into one application was not possible due to limitation of the experimental setup;
- Although the system energy consumption was estimated, aspects like the processor wear was not considered;

Future works to continue the research in control system in heterogeneous computing and to fulfill the improvements previously mentioned are as follows:

- Import the system into a real chip;
- Embed the chip into the Hector Quadrotor simulation (*hardware-on-the-loop*) and compare the results;



- Migrate the whole system to a real quadrotor;
- Add more applications related to sensing that demands more computational power (Computer vision, LIDARs, Particle Filters and more.);
- Migrate this system to other robotics applications, like ground and water vehicles.
- Integrate into the system applications like: path planning and obstacle detection;
- Integrate prediction models to estimate the processor's wear;
- Implement this system where the energy consumption of the processor impacts more in the overall system, like nano satellites.

## REFERENCES

- [AAS15] Ali, O. A. M.; Ali, A. Y.; Sumait, B. S. “Comparison between the effects of different types of membership functions on fuzzy logic controller performance”, *International Journal*, vol. 76, 2015, pp. 76–83.
- [AB09] Alessio, A.; Bemporad, A. “A survey on explicit model predictive control”, *Nonlinear Model Predictive Control: Towards New Challenging Applications*, vol. 384, 2009, pp. 345–369.
- [AC18] Andrade, H.; Crnkovic, I. “A review on software architectures for heterogeneous platforms”. In: *IEEE Asia-Pacific Software Engineering Conference (APSEC)*, 2018, pp. 209–218.
- [ACL05] Ang, K. H.; Chong, G.; Li, Y. “PID control system analysis, design, and technology”, *IEEE transactions on control systems technology*, vol. 13–4, 2005, pp. 559–576.
- [ACVR05] Albertos, P.; Crespo, A.; Vallés, M.; Ripoll, I. “Embedded control systems: some issues and solutions”, *IFAC Proceedings Volumes*, vol. 38–1, 2005, pp. 203–208.
- [AH12] Aguiar, A.; Hessel, F. P. “Exploring embedded software concepts using the Hellfire platform with undergraduate students”. In: *IEEE Interdisciplinary Engineering Design Education Conference (IEDEC)*, 2012, pp. 1–4.
- [AJS14] Afram, A.; Janabi-Sharifi, F. “Theory and applications of HVAC control systems—A review of model predictive control (MPC)”, *Building and Environment*, vol. 72, 2014, pp. 343–355.
- [AMY13] Abbasi, E.; Mahjoob, M.; Yazdanpanah, R. “Controlling of quadrotor uav using a fuzzy system for tuning the pid gains in hovering mode”. In: *ACE International Conference on Advances in Computer Entertainment (ISACA)*, 2013, pp. 1–6.
- [Arm10] Armoush, A. “Design patterns for safety-critical embedded systems”, Ph.D. Thesis, RWTH Aachen University, 2010, 197p.
- [Aro12] Arora, M. “The architecture and evolution of cpu-gpu systems for general purpose computing”, *High Performance Computing on Graphics Processing Units*, vol. 27, 2012, pp. 1–12.
- [ASFM<sup>+</sup>10] Aguiar, A.; Sérgio Filho, J.; Magalhães, F. G.; Casagrande, T. D.; Hessel, F. “Hellfire: A design framework for critical embedded systems’ applications”. In:

IEEE International Symposium on Quality Electronic Design (ISQED), 2010, pp. 730–737.

- [ATR<sup>+</sup>17] Abdi, F.; Tabish, R.; Rungger, M.; Zamani, M.; Caccamo, M. “Application and system-level software fault tolerance through full system restarts”. In: ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), 2017, pp. 197–206.
- [AVN16] Anjali, B.; Vivek, A.; Nandagopal, J. “Simulation and analysis of integral LQR controller for inner control loop design of a fixed wing micro aerial vehicle (MAV)”, *Procedia Technology*, vol. 25, 2016, pp. 76–83.
- [ÅW13] Åström, K. J.; Wittenmark, B. “Computer-controlled systems: theory and design”. Courier Corporation, 2013, 899p.
- [BDH<sup>+</sup>10] Brodtkorb, A. R.; Dyken, C.; Hagen, T. R.; Hjelmervik, J. M.; Storaasli, O. O. “State-of-the-art in heterogeneous computing”, *Scientific Programming*, vol. 18–1, 2010, pp. 1–33.
- [Ben96] Bennett, S. “A brief history of automatic control”, *IEEE Control Systems Magazine*, vol. 16–3, 1996, pp. 17–25.
- [Ben20] Bengtsson, L. “Implementation of Control Algorithms in Small Embedded Systems”, *Engineering*, vol. 12–9, 2020, pp. 623–639.
- [BH05] Boyer, W. F.; Hura, G. S. “Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments”, *Journal of Parallel and Distributed Computing*, vol. 65–9, 2005, pp. 1035–1046.
- [BHKW12] Buchty, R.; Heuveline, V.; Karl, W.; Weiss, J.-P. “A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators”, *Concurrency and Computation: Practice and Experience*, vol. 24–7, 2012, pp. 663–675.
- [CAV<sup>+</sup>14] Codrescu, L.; Anderson, W.; Venkumanhanti, S.; Zeng, M.; Plondke, E.; Koob, C.; Ingle, A.; Tabony, C.; Maule, R. “Hexagon DSP: An architecture optimized for mobile multimedia and communications”, *IEEE Micro*, vol. 34–2, 2014, pp. 34–43.
- [CCF<sup>+</sup>16] Choi, Y.-k.; Cong, J.; Fang, Z.; Hao, Y.; Reinman, G.; Wei, P. “A quantitative analysis on microarchitectures of modern CPU-FPGA platforms”. In: IEEE/ACM Design Automation Conference (DAC), 2016, pp. 1–6.
- [CCF<sup>+</sup>19] Choi, Y.-K.; Cong, J.; Fang, Z.; Hao, Y.; Reinman, G.; Wei, P. “In-depth analysis on microarchitectures of modern heterogeneous CPU-FPGA platforms”, *ACM*

*Transactions on Reconfigurable Technology and Systems (TRETs)*, vol. 12–1, 2019, pp. 1–20.

- [CDK05] Coulouris, G. F.; Dollimore, J.; Kindberg, T. “Distributed systems: concepts and design”. Pearson Education, 2005, 744p.
- [CECK12] Cho, H.-D.; Engineer, P. D. P.; Chung, K.; Kim, T. “Benefits of the big. LITTLE Architecture”. Accessed: February 14, 2012, Source: <https://www.eetimes.com/>, 2012.
- [CFLW04] Chu, E.-W.; Fan, H.-Y.; Lin, W.-W.; Wang, C.-S. “Structure-preserving algorithms for periodic discrete-time algebraic Riccati equations”, *International Journal of Control*, vol. 77–8, 2004, pp. 767–788.
- [Cha14] Chang, X.-H. “Robust output feedback H-infinity control and filtering for uncertain linear systems”. Springer Science & Business, 2014, 245p.
- [Che21] Check, N. “Qualcomm Snapdragon 800 MSM8974”. Accessed: September 15, 2021, Source: <https://www.notebookcheck.net/Qualcomm-Snapdragon-800-MSM8974-SoC.103706.0.html>, 2021.
- [CHL+05] Choset, H. M.; Hutchinson, S.; Lynch, K. M.; Kantor, G.; Burgard, W.; Kavraki, L. E.; Thrun, S. “Principles of robot motion: theory, algorithms, and implementation”. MIT press, 2005, 603p.
- [CKC12] Chung, H.; Kang, M.; Cho, H.-D. “Heterogeneous multi-processing solution of Exynos 5 Octa with ARM big. LITTLE technology”, Technical Report, Samsung, 2012, 8p.
- [CLW+07] Carpin, S.; Lewis, M.; Wang, J.; Balakirsky, S.; Scrapper, C. “USARSim: a robot simulator for research and education”. In: IEEE International Conference on Robotics and Automation (ICRA), 2007, pp. 1400–1405.
- [CSvAK13] Chitchian, M.; Simonetto, A.; van Amesfoort, A. S.; Keviczky, T. “Distributed computation particle filters on GPU architectures for real-time control applications”, *IEEE Transactions on Control Systems Technology*, vol. 21–6, 2013, pp. 2224–2238.
- [D+20] Domingues, A. R. P.; et al.. “ORCA: a self-adaptive, multiprocessor system-on-chip platform”, Master’s Thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 2020, 110p.
- [DAK+21] Dörflinger, A.; Albers, M.; Kleinbeck, B.; Guan, Y.; Michalik, H.; Klink, R.; Blochwitz, C.; Nechi, A.; Berekovic, M. “A comparative survey of open-source

application-class RISC-V processor implementations”. In: ACM International Conference on Computing Frontiers (CF), 2021, pp. 12–20.

- [DBD16] Demir, B. E.; Bayir, R.; Duran, F. “Real-time trajectory tracking of an unmanned aerial vehicle using a self-tuning fuzzy proportional integral derivative controller”, *International Journal of Micro Air Vehicles*, vol. 8–4, 2016, pp. 252–268.
- [DDAB+13] De Dinechin, B. D.; Ayrignac, R.; Beaucamps, P.-E.; Couvert, P.; Ganne, B.; de Massas, P. G.; Jacquet, F.; Jones, S.; Chaisemartin, N. M.; Riss, F.; et al.. “A clustered manycore processor architecture for embedded and accelerated applications”. In: IEEE High-Performance Extreme Computing Conference (HPEC), 2013, pp. 1–6.
- [De 13] De Nardi, R. “The qrsim quadrotors simulator”, *UCL Department of Computer Science Research Note*, vol. 13–08, 2013, pp. 1–51.
- [DISW14] DiStefano III, J. J.; Stubberud, A. R.; Williams, I. J. “Schaum’s outline of feedback and control systems”. McGraw-Hill Education, 2014, 512p.
- [DJSFA19] Domingues, A. R.; Jurak, D. A.; Sergio Filho, J.; Amory, A. D. M. “Integrating an MPOSoC to a Robotics Environment”. In: IEEE Latin American Robotics Symposium (LARS), 2019, pp. 204–209.
- [DO18] Deng, H.; Ohtsuka, T. “A parallel code generation toolkit for nonlinear model predictive control”. In: IEEE Conference on Decision and Control (CDC), 2018, pp. 4920–4926.
- [DP+17] Dharmawan, A.; Priyambodo, T. K.; et al.. “Model of linear quadratic regulator (lqr) control method in hovering state of quadrotor”, *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9–3, 2017, pp. 135–143.
- [dRdS21] del Rio, D. G.; da Silva, D. S. “Estudo De Aplicação de Lógica Fuzzy em Qualidade Energia: Uma Revisão”, *Engenharia Elétrica: o Caminho para o Desenvolvimento Sustentável*, vol. 1–1, 2021, pp. 51–71.
- [DYZG15] Danjun, L.; Yan, Z.; Zongying, S.; Geng, L. “Autonomous landing of quadrotor based on ground effect modelling”. In: IEEE Chinese Control Conference (CCC), 2015, pp. 5647–5652.
- [ELLSV97] Edwards, S.; Lavagno, L.; Lee, E. A.; Sangiovanni-Vincentelli, A. “Design of embedded systems: Formal models, validation, and synthesis”, *Proceedings of the IEEE*, vol. 85–3, 1997, pp. 366–390.

- [EN18] Emran, B. J.; Najjaran, H. “A review of quadrotor: An underactuated mechanical system”, *Annual Reviews in Control*, vol. 46, 2018, pp. 165–180.
- [Eng20] Engine, U. “Unreal Engine: The most powerful real-time 3D creation platform”. Accessed: January 17, 2020, Source: <https://www.unrealengine.com/>, 2020.
- [FBAS16] Furrer, F.; Burri, M.; Achtelik, M.; Siegwart, R. “RotorS—A modular Gazebo MAV simulator framework”. In: *Robot Operating System (ROS)*, Koubaa, A. (Editor), Springer, 2016, chap. 7, pp. 595–625.
- [Fou14] Foundation, O. S. R. “OSRF Gazebo”. Accessed: October 7, 2019, Source: <http://gazebosim.org>, 2014.
- [Fou20] Foundation, D. “jMAVSim Sim with SITL”. Accessed: April 7, 2022, Source: <https://docs.px4.io/master/en/simulation/jmavsim.html>, 2020.
- [GAAA16] Ghazbi, S. N.; Aghli, Y.; Alimohammadi, M.; Akbari, A. A. “Quadrotors unmanned aerial vehicles: A review”, *International Journal on Smart Sensing & Intelligent Systems*, vol. 9–1, 2016, pp. 309–333.
- [Gaj03] Gajic, Z. “Linear dynamic systems and signals”. Prentice Hall/Pearson Education Upper Saddle River, 2003, 700p.
- [GKFF20] Giardino, M.; Klawitter, E.; Ferri, B.; Ferri, A. “A Power-and Performance-Aware Software Framework for Control System Applications”, *IEEE Transactions on Computers*, vol. 69–10, 2020, pp. 1544–1555.
- [GPM89] Garcia, C. E.; Prett, D. M.; Morari, M. “Model predictive control: Theory and practice—A survey”, *Automatica*, vol. 25–3, 1989, pp. 335–348.
- [GQC+20] Gan, Y.; Qiu, Y.; Chen, L.; Leng, J.; Zhu, Y. “Low-latency proactive continuous vision”. In: *ACM International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2020, pp. 329–342.
- [GR15] Ghaffar, A. A.; Richardson, T. “Model reference adaptive control and LQR control for quadrotor with parametric uncertainties”, *International Journal of Mechanical and Mechatronics Engineering*, vol. 9–2, 2015, pp. 244–250.
- [Gre11] Greenhalgh, P. “Big. little processing with arm cortex-a15 & cortex-a7”, *ARM White paper*, vol. 17, 2011, pp. 1–8.
- [Gre13] Greenhalgh, P. “big. LITTLE technology: The future of mobile”, Technical Report, ARM Limited, 2013, 12p.

- [GS98] Gardner, W. B.; Serra, M. “An object-oriented layered approach to interfaces for hardware/software codesign of embedded systems”. In: Hawaii International Conference on System Sciences (HICSS), 1998, pp. 197–206.
- [GSBT16] Gandolfo, D. C.; Salinas, L. R.; Brandão, A.; Toibero, J. M. “Stable path-following control for a quadrotor helicopter considering energy consumption”, *IEEE Transactions on Control Systems Technology*, vol. 25–4, 2016, pp. 1423–1430.
- [HCEH<sup>+</sup>19] Huang, S.; Chang, L.-W.; El Hajj, I.; Garcia de Gonzalo, S.; Gómez-Luna, J.; Chalamalasetti, S. R.; El-Hadedy, M.; Milojevic, D.; Mutlu, O.; Chen, D.; et al.. “Analysis and modeling of collaborative execution strategies for heterogeneous CPU-FPGA architectures”. In: ACM/SPEC International Conference on Performance Engineering (ICPE), 2019, pp. 79–90.
- [Hea02] Heath, S. “Embedded systems design”. Elsevier, 2002, 430p.
- [Hel01] Hellmann, M. “Fuzzy logic introduction”, *Université de Rennes*, vol. 1, 2001, pp. 1–9.
- [HGH<sup>+</sup>19] He, Z.; Gao, W.; He, X.; Wang, M.; Liu, Y.; Song, Y.; An, Z. “Fuzzy intelligent control method for improving flight attitude stability of plant protection quadrotor UAV”, *International Journal of Agricultural and Biological Engineering*, vol. 12–6, 2019, pp. 110–115.
- [HGL<sup>+</sup>20] Hu, W.; Gan, Y.; Lv, X.; Wang, Y.; Wen, Y. “A Improved List Heuristic Scheduling Algorithm for Heterogeneous Computing Systems”. In: IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2020, pp. 1111–1116.
- [HHWT09] Huang, H.; Hoffmann, G. M.; Waslander, S. L.; Tomlin, C. J. “Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering”. In: IEEE International Conference on Robotics and Automation (ICRA), 2009, pp. 3277–3282.
- [HJ03] Hagaras, T.; Janecek, J. “A simple scheduling heuristic for heterogeneous computing environments”. In: IEEE International Symposium on Parallel and Distributed Computing (ISPDC), 2003, pp. 104–104.
- [HLM<sup>+</sup>14] Hartman, D.; Landis, K.; Mehrer, M.; Moreno, S.; Kim, J. “Quadcopter dynamic modeling and simulation (Quad-Sim)(2014)”. Accessed: April 12, 2020, Source: <https://github.com/dch33/Quad-Sim>, 2014.
- [HS14] Huang, H.; Sturm, J. “tum\_simulator - ROS”. Accessed: February 4, 2020, Source: [http://wiki.ros.org/tum\\_simulator](http://wiki.ros.org/tum_simulator), 2014.

- [Ian12] Iancu, I. "A Mamdani type fuzzy logic controller", *Fuzzy logic-controls, concepts, theories and applications*, vol. 15–2, 2012, pp. 325–350.
- [JB20] Jiang, Z.; Behbahani, A. R. "Multi-Layer Model Predictive Control for Integrated Power/Propulsion Systems". In: *AIAA Propulsion and Energy Forum*, 2020, pp. 1–16.
- [JC18] Joseph, L.; Cacace, J. "Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System". Packt Publishing Ltd, 2018, 570p.
- [JCH<sup>+</sup>17] Jing, X.; Cui, J.; He, H.; Zhang, B.; Ding, D.; Yang, Y. "Attitude estimation for UAV using extended Kalman filter". In: *IEEE Chinese Control And Decision Conference (CCDC)*, 2017, pp. 3307–3312.
- [Jia05] Jiang, J. "Fault-tolerant control systems-an introductory overview", *Acta Automatica Sinica*, vol. 31–1, 2005, pp. 161–174.
- [JMdMA<sup>+</sup>21] Juracy, L. R.; Moreira, M. T.; de Morais Amory, A.; Hampel, A. F.; Moraes, F. G. "A high-level modeling framework for estimating hardware metrics of CNN accelerators", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68–11, 2021, pp. 4783–4795.
- [Joh77] Johnson, G. R. "A microprocessor based solar controller". In: *IEEE Conference on Decision and Control (CDC)*, 1977, pp. 336–340.
- [JS20] Jain, A.; Sharma, A. "Membership function formulation methods for fuzzy logic systems: A comprehensive review", *Journal of Critical Reviews*, vol. 7–19, 2020, pp. 8717–8733.
- [JTRH14] Jalil, M. H. A.; Taib, M. N.; Rahiman, M. F.; Hamdan, R. "Back calculation Anti Windup PID controller on Several Well-Known Tuning Method for Glycerin Bleaching Process Temperature Regulation", *International Journal of Integrated Engineering*, vol. 6–3, 2014, pp. 39–50.
- [JW05] Jerraya, A. A.; Wolf, W. "The what, why, and how of MPSoCs". In: *Multiprocessor Systems-on-Chips*, Jerraya, A. A.; Wolf, W. (Editors), Elsevier, 2005, chap. 1, pp. 1–18.
- [Kal60] Kalman, R. E. "A new approach to linear filtering and prediction problems", *Journal of basic Engineering*, vol. 82–1, 1960, pp. 35–45.
- [Kan16] Kanter, D. "RISC-V offers simple, modular ISA", *Microprocessor Report*, vol. 1, 2016, pp. 1–5.



- [KGD14] Khatoon, S.; Gupta, D.; Das, L. "PID & LQR control for a quadrotor: Modeling and simulation". In: IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2014, pp. 796–802.
- [KGW19] Kim, J.; Gadsden, S. A.; Wilkerson, S. A. "A comprehensive survey of control strategies for autonomous quadrotors", *Canadian Journal of Electrical and Computer Engineering*, vol. 43–1, 2019, pp. 3–16.
- [KH04] Koenig, N.; Howard, A. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2004, pp. 2149–2154.
- [KK12] Kaur, A.; Kaur, A. "Comparison of fuzzy logic and neuro-fuzzy algorithms for air conditioning system", *International journal of soft computing and engineering*, vol. 2–1, 2012, pp. 417–20.
- [KKSC18] Khusainov, B.; Kerrigan, E.; Suardi, A.; Constantinides, G. "Nonlinear predictive control on a heterogeneous computing platform", *Control Engineering Practice*, vol. 78, 2018, pp. 105–115.
- [KM79] Klein, C. A.; Maney, J. J. "Real-time control of a multiple-element mechanical linkage with a microcomputer", *IEEE Transactions on Industrial Electronics and Control Instrumentation*, vol. IECI-26–4, 1979, pp. 227–234.
- [KNG05] Kristiansen, R.; Nicklasson, P. J.; Gravdahl, J. T. "Satellite attitude tracking by quaternion-based backstepping", *IFAC Proceedings Volumes*, vol. 38–1, 2005, pp. 175–180.
- [Kni02] Knight, J. C. "Safety critical systems: challenges and directions". In: ACM/IEEE International Conference on Software Engineering (ICSE), 2002, pp. 547–550.
- [KO19] Kose, O.; Oktay, T. "Dynamic Modeling and Simulation of Quadrotor for Different Flight Conditions", *European Journal of Science and Technology*, vol. 15, 2019, pp. 132–142.
- [Kou15] Koubaa, A. "ROS as a service: web services for robot operating system", *Journal of Software Engineering for Robotics*, vol. 6–1, 2015, pp. 1–14.
- [LC09] Levy, M.; Conte, T. M. "Embedded multicore processors and systems", *IEEE micro*, vol. 29–3, 2009, pp. 7–9.
- [LCA+11] Li, S.; Chen, K.; Ahn, J. H.; Brockman, J. B.; Jouppi, N. P. "CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques". In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2011, pp. 694–701.

- [Lei09] Leite, L. d. C. M. “Geração e Simplificação da Base de Conhecimento de um Sistema Híbrido Fuzzy-Genético”, Master’s Thesis, Universidade do Estado do Rio de Janeiro, 2009, 112p.
- [LHL<sup>+</sup>12] Li, X.; Hu, J.; Lv, W.; Wang, G.; Cai, X. “Research on DSP-GPU Heterogeneous Computing System”. In: National Conference on Information Technology and Computer Science (NCCIT), 2012, pp. 423–425.
- [Li05] Li, Q. “Fundamentals of rtos-based digital controller implementation”. In: *Handbook of networked and embedded control systems*, Hristu-Varsakelis, D. (Editor), Springer, 2005, chap. 3, pp. 353–375.
- [LLW16] Liu, F.; Liang, Y.; Wang, L.-z. “A survey of the heterogeneous computing platform and related technologies”, *Proc. DEStech Trans. Eng. Technol. Research IMEIA*, vol. 1, 2016, pp. 6–12.
- [LLZ<sup>+</sup>20] Li, W.; Liang, J.; Zhang, Y.; Jia, H.; Xiao, L.; Li, Q. “Accelerated LiDAR data processing algorithm for self-driving cars on the heterogeneous computing platform”, *IET Computers & Digital Techniques*, vol. 14–5, 2020, pp. 201–209.
- [LSJ15] Li, L.; Sun, L.; Jin, J. “Survey of advances in control algorithms of quadrotor unmanned aerial vehicle”. In: IEEE International Conference on Communication Technology (ICCT), 2015, pp. 107–111.
- [LSN14] Li, A.; Serban, R.; Negrut, D. “An Overview of NVIDIA Tegra K1 Architecture”. Accessed: August 22, 2021, Source: <http://sbel.wisc.edu/documents/TR-2014-17.pdf>.--2014, 2014.
- [LTL<sup>+</sup>21] Liu, L.; Tang, J.; Liu, S.; Yu, B.; Xie, Y.; Gaudiot, J.-L. “ $\pi$ -rt: A runtime framework to enable energy-efficient real-time robotic vision applications on heterogeneous architectures”, *Computer*, vol. 54–4, 2021, pp. 14–25.
- [LWM08] Ling, K.-V.; Wu, B. F.; Maciejowski, J. “Embedded model predictive control (MPC) using a FPGA”, *IFAC Proceedings Volumes*, vol. 41–2, 2008, pp. 15250–15255.
- [M<sup>+</sup>18] Martins, A. L. D. M.; et al.. “Multi-objective resource management for many-core systems”, Ph.D. Thesis, PPGCC - Pontifícia Universidade Católica do Rio Grande do Sul, 2018, 149p.
- [Ma19] Ma, L. “Low power and high performance heterogeneous computing on FPGAs”. Accessed: November 2, 2019, Source: <https://webcache.googleusercontent.com/search?q=cache:0Z7dTCEo0sIJ:https://iris.polito.it/retrieve/handle/11583/2727228/235579/summary.pdf+&cd=1&hl=en&ct=clnk&gl=br>, 2019.

- [Mat05] MathWorks, I. "MATLAB: The Language of Technical Computing. Getting started with MATLAB, version 7". MathWorks, Incorporated, 2005, 240p.
- [MCDH17] Ma, Y.; Chantem, T.; Dick, R. P.; Hu, X. S. "Improving system-level lifetime reliability of multicore soft real-time systems", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25–6, 2017, pp. 1895–1905.
- [MCL16] Morbidi, F.; Cano, R.; Lara, D. "Minimum-energy path generation for a quadrotor UAV". In: IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 1492–1498.
- [MCL<sup>+</sup>18] Müller, M.; Casser, V.; Lahoud, J.; Smith, N.; Ghanem, B. "SIM4CV: A Photo-Realistic Simulator for Computer Vision Applications", *International Journal of Computer Vision- IJCV*, vol. 126–1, 2018, pp. 902–919.
- [MCM<sup>+</sup>04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. "HERMES: an infrastructure for low area overhead packet-switching networks on chip", *Integration*, vol. 38–1, 2004, pp. 69–93.
- [MDM<sup>+</sup>18] Mück, T.; Donyanavard, B.; Moazzemi, K.; Rahmani, A. M.; Jantsch, A.; Dutt, N. "Design methodology for responsive and robust mimo control of heterogeneous multicores", *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4–4, 2018, pp. 944–951.
- [Meg06] Megretski, A. "MIT EECS 6.24 - LECTURE NOTES BY A. MEGRETSKI". Accessed: June 10, 2020, Source: [http://web.mit.edu/course/6/6.241/ameg\\_www\\_fall2006/www/images/L06L2gain.pdf](http://web.mit.edu/course/6/6.241/ameg_www_fall2006/www/images/L06L2gain.pdf), 2006.
- [MF19] Mo, H.; Farid, G. "Nonlinear and adaptive intelligent control techniques for quadrotor uav—a survey", *Asian Journal of Control*, vol. 21–2, 2019, pp. 989–1008.
- [MGEY06] Markaroglu, H.; Guzelkaya, M.; Eksin, I.; Yesil, E. "Tracking time adjustment in back calculation anti-windup scheme". In: European Conference on Modeling and Simulation (ECMS), 2006, pp. 613–614.
- [MILH18] Mishra, N.; Imes, C.; Lafferty, J. D.; Hoffmann, H. "Caloree: Learning control for predictable latency and low energy", *ACM SIGPLAN Notices*, vol. 53–2, 2018, pp. 184–198.
- [MLSF<sup>+</sup>12] Magalhaes, F. G.; Longhi, O.; Sérgio Filho, J.; Aguiar, A.; Hessel, F. "Noc-based platform for embedded software design: An extension of the hellfire framework". In: IEEE International Symposium on Quality Electronic Design (ISQED), 2012, pp. 97–102.

- [MMY<sup>+</sup>19] Moazzemi, K.; Maity, B.; Yi, S.; Rahmani, A. M.; Dutt, N. "HESSLE-FREE: Heterogeneous Systems Leveraging Fuzzy Control for Real-time Resource Management", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18–5s, 2019, pp. 1–19.
- [MSC<sup>+</sup>14] Martins, A. L.; Silva, D. R.; Castilhos, G. M.; Monteiro, T. M.; Moraes, F. G. "A method for NoC-based MPSoC energy consumption estimation". In: *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2014, pp. 427–430.
- [MSFLH14] Magalhaes, F. G.; Sérgio Filho, J.; Longhi, O.; Hessel, F. "Embedded cluster-based architecture with high level support-presenting the hc-mpsoc". In: *IEEE International Symposium on Rapid System Prototyping (RSP)*, 2014, pp. 100–106.
- [MSK<sup>+</sup>12] Meyer, J.; Sendobry, A.; Kohlbrecher, S.; Klingauf, U.; Von Stryk, O. "Comprehensive simulation of quadrotor uavs using ros and gazebo". In: *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAN)*, 2012, pp. 400–411.
- [MV15] Mittal, S.; Vetter, J. S. "A survey of CPU-GPU heterogeneous computing techniques", *ACM Computing Surveys (CSUR)*, vol. 47–4, 2015, pp. 1–35.
- [MWH13] Monson, J.; Wirthlin, M.; Hutchings, B. L. "Implementing high-performance, low-power FPGA-based optical flow accelerators in C". In: *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2013, pp. 363–369.
- [Nis20] Nise, N. S. "Control systems engineering". John Wiley & Sons, 2020, 800p.
- [NPRC17] Noori, F. M.; Portugal, D.; Rocha, R. P.; Couceiro, M. S. "On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo". In: *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2017, pp. 19–24.
- [NQN<sup>+</sup>20] Nguyen, H. T.; Quyen, T. V.; Nguyen, C. V.; Le, A. M.; Tran, H. T.; Nguyen, M. T. "Control Algorithms for UAVs: A Comprehensive Survey", *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 7–23, 2020, pp. 1–11.
- [NR19] Nithya, M.; Rashmi, M. "Gazebo-ROS-Simulink Framework for Hover Control and Trajectory Tracking of Crazyflie 2.0". In: *IEEE Region 10 Conference (TENCON)*, 2019, pp. 649–653.

- [NS19] Nascimento, T. P.; Saska, M. "Position and attitude control of multi-rotor aerial vehicles: A survey", *Annual Reviews in Control*, vol. 48, 2019, pp. 129–146.
- [Oli04] Olivier, M. "Cyberbotics LTD-webotstm: Professional mobile robot simulation", *International Journal of Advanced Robotic Systems*, vol. 1–1, 2004, pp. 40–43.
- [OMT97] Olson, C. L.; Moore, T.; Turner, J. "FlightGear Flight Simulator". Accessed: January 22, 2020, Source: <https://www.flightgear.org/>, 1997.
- [OWJ82] Ohlson, K. B.; Westenskow, D. R.; Jordan, W. S. "A microprocessor based feedback controller for mechanical ventilation", *Annals of biomedical engineering*, vol. 10–1, 1982, pp. 35–48.
- [PG19] Podlubne, A.; Göhringer, D. "FPGA-ROS: Methodology to augment the robot operating system with FPGA designs". In: IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2019, pp. 1–5.
- [PSY88] Psaltis, D.; Sideris, A.; Yamamura, A. A. "A multilayered neural network controller", *IEEE control systems magazine*, vol. 8–2, 1988, pp. 17–21.
- [PUC19] PUCRS, H. D. S. G. "HeMPS - Hermes Multiprocessor System on Chip". Accessed: May 20, 2019, Source: <https://www.inf.pucrs.br/hemps/>, 2019.
- [QCG<sup>+</sup>09] Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A. Y. "ROS: an open-source Robot Operating System". In: ICRA Workshop on Open Source Software, 2009, pp. 1–5.
- [QGS15] Quigley, M.; Gerkey, B.; Smart, W. D. "Programming Robots with ROS: a practical introduction to the Robot Operating System". " O'Reilly Media, Inc, 2015, 130p.
- [RC18] Raju, K.; Chiplunkar, N. N. "A survey on techniques for cooperative CPU-GPU computing", *Sustainable Computing: Informatics and Systems*, vol. 19, 2018, pp. 72–85.
- [RFXP12] Rui, W.; Fei, L.; Xiaohong, H.; Peng, L. "New elevator energy feedback control system design based on fuzzy PID controller". In: IEEE Power Engineering and Automation Conference (PEAM), 2012, pp. 1–6.
- [RM77] Reed, M.; Mergler, H. "A microprocessor-based control system", *IEEE Transactions on Industrial Electronics and Control Instrumentation*, vol. IECI-24–3, 1977, pp. 253–257.
- [RP01] Roychowdhury, S.; Pedrycz, W. "A survey of defuzzification strategies", *International Journal of intelligent systems*, vol. 16–6, 2001, pp. 679–695.

- [SAY18] Shraim, H.; Awada, A.; Youness, R. "A survey on quadrotors: Configurations, modeling and identification, control, collision avoidance, fault diagnosis and tolerant control", *IEEE Aerospace and Electronic Systems Magazine*, vol. 33–7, 2018, pp. 14–33.
- [SDLK18] Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and Service Robotics (FSR)*, 2018, pp. 621–635.
- [Sei75] Seim, T. A. "Automation of a brazing process with an Intel 8008 microprocessor", *IEEE Transactions on Industrial Electronics and Control Instrumentation*, vol. IECI-22–3, 1975, pp. 303–307.
- [SEMDI16] Seborg, D. E.; Edgar, T. F.; Mellichamp, D. A.; Doyle III, F. J. "Process dynamics and control". John Wiley & Sons, 2016, 514p.
- [SFAM+12] Sérgio Filho, J.; Aguiar, A.; Magalhães, F. G.; Longhi, O.; Hessel, F. "An RTOS Methodology for NoC Based Systems' Support–The HellfireOS Case Study". In: *IEEE Brazilian Conference on Critical Embedded Systems (CBSEC)*, 2012, pp. 82–87.
- [SGP20] Saraf, P.; Gupta, M.; Parimi, A. M. "A Comparative Study Between a Classical and Optimal Controller for a Quadrotor". In: *IEEE India Council International Conference (INDICON)*, 2020, pp. 1–6.
- [She78] Shepperd, S. W. "Quaternion from rotation matrix", *Journal of Guidance and Control*, vol. 1–3, 1978, pp. 223–224.
- [SJK12] Sepulchre, R.; Jankovic, M.; Kokotovic, P. V. "Constructive nonlinear control". Springer Science & Business Media, 2012, 313p.
- [SL+91] Slotine, J.-J. E.; Li, W.; et al.. "Applied nonlinear control". Prentice hall Englewood Cliffs, NJ, 1991, 476p.
- [SM12] Sousa, É. R.; Meloni, L. G. P. "High-Performance Computing Based on Heterogeneous and Reconfigurable Architectures". In: *IEEE International Conference on Computational Intelligence and Communication Networks (CICN)*, 2012, pp. 530–534.
- [SMRD18] Shahhosseini, S.; Moazzemi, K.; Rahmani, A. M.; Dutt, N. "On the feasibility of SISO control-theoretic DVFS for power capping in CMPs", *Microprocessors and Microsystems*, vol. 63, 2018, pp. 249–258.
- [Son13] Sontag, E. D. "Mathematical control theory: deterministic finite dimensional systems". Springer Science & Business Media, 2013, 532p.

- [Spu14] Spurgeon, S. "Sliding mode control: a tutorial". In: IEEE European Control Conference (ECC), 2014, pp. 2272–2277.
- [Sze10] Szeliski, R. "Computer vision: algorithms and applications". Springer Science & Business Media, 2010, 812p.
- [TAK06] Tong, J. G.; Anderson, I. D.; Khalid, M. A. "Soft-core processors for embedded systems". In: IEEE International Conference on Microelectronics (ICM), 2006, pp. 170–173.
- [TBF05] Thrun, S.; Burgard, W.; Fox, D. "Probabilistic robotics". MIT press, 2005, 646p.
- [TDSP19] Terzo, O.; Djemame, K.; Scionti, A.; Pezuela, C. "Heterogeneous Computing Architectures: Challenges and Vision". CRC Press, 2019, 338p.
- [TLG17] Tang, J.; Liu, S.; Gaudiot, J. "Embedded Systems Architecture for SLAM Applications", *CoRR*, vol. abs/1702.01295, 2017, pp. 1–4.
- [TLL+20] Tang, J.; Liu, S.; Liu, L.; Yu, B.; Shi, W. "Lopecs: A low-power edge computing system for real-time autonomous driving services", *IEEE Access*, vol. 8, 2020, pp. 30467–30479.
- [USK19] Ulbricht, M.; Syed, R. T.; Krstic, M. "Developing a Configurable Fault Tolerant Multicore System for Optimized Sensor Processing". In: IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2019, pp. 1–4.
- [VDP+20] Vancin, P. H.; Domingues, A. R.; Paravisi, M.; Johann, S. F.; Calazans, N. L.; Amory, A. M. "Towards an Integrated Software Development Environment for Robotic Applications in MPSoCs with Support for Energy Estimations". In: IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–5.
- [Wan09] Wang, L. "Model predictive control system design and implementation using MATLAB®". Springer Science & Business Media, 2009, 378p.
- [WJM08] Wolf, W.; Jerraya, A. A.; Martin, G. "Multiprocessor system-on-chip (MPSoC) technology", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27–10, 2008, pp. 1701–1713.
- [WLPA11] Waterman, A.; Lee, Y.; Patterson, D. A.; Asanovic, K. "The risc-v instruction set manual, volume i: Base user-level isa", *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011, pp. 1–32.

- [Xia16] Xiao, T. "A Literature Review of Learning and Optimization Methods Applied to Quadrotor Control". Accessed: March 7, 2019, Source: <https://tedxiao.me/>, 2016.
- [Xil18] Xilinx, I. "Zynq-7000 SoC Technical Reference Manual", Technical Report, Technical report, July 2018. URL <https://www.xilinx.com/support>, 2018, 1825p.
- [XM15] Xu, D.; Mandic, D. P. "The theory of quaternion matrix derivatives", *IEEE Transactions on Signal Processing*, vol. 63–6, 2015, pp. 1543–1556.
- [YB19] Yilmaz, Z.; Bayindir, L. "57. Simulation of Lidar-Based Robot Detection Task using ROS and Gazebo", *European Journal of Science and Technology (EJOSAT)*, vol. 1–1, 2019, pp. 513–529.
- [YBDM03] Ye, T. T.; Benini, L.; De Micheli, G. "Packetized on-chip interconnect communication analysis for MPSoC". In: IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE), 2003, pp. 344–349.
- [YDG04] Yang, C.-H.; Deconinck, G.; Gui, W.-H. "Fault-tolerant scheduling for real-time embedded control systems", *Journal of Computer Science and Technology*, vol. 19–2, 2004, pp. 191–202.
- [Yin00] Ying, H. "Fuzzy control and modeling: analytical foundations and applications". IEEE press, 2000, 342p.
- [YMK14] Youness, H.; Moness, M.; Khaled, M. "MPSoCs and multicore microcontrollers for embedded PID control: A detailed study", *IEEE Transactions on Industrial Informatics*, vol. 10–4, 2014, pp. 2122–2134.
- [Zad65] Zadeh, L. A. "Information and control", *Fuzzy sets*, vol. 8–3, 1965, pp. 338–353.
- [Zah17] Zahran, M. "Heterogeneous computing: Here to stay", *Communications of the ACM*, vol. 60–3, 2017, pp. 42–45.
- [Zah19] Zahran, M. "Heterogeneous computing: Hardware and software perspectives". Morgan & Claypool, 2019, 127p.
- [Zak03] Zak, S. H. "Systems and control". Oxford University Press New York, 2003, 788p.
- [ZCF20] Zoni, D.; Cremona, L.; Fornaciari, W. "All-digital control-theoretic scheme to optimize energy budget and allocation in multi-cores", *IEEE Transactions on Computers*, vol. 69–5, 2020, pp. 706–721.



- [ZJ14] Zulu, A.; John, S. “A Review of Control Algorithms for Autonomous Quadrotors”, *Open Journal of Applied Sciences*, vol. 04–14, 2014, pp. 547–556.
- [ZWT+18] Zhang, Y.; Wang, Y.; Tang, X.; Yuan, X.; Xu, Y. “Energy-efficient task scheduling on heterogeneous computing systems by linear programming”, *Concurrency and Computation: Practice and Experience*, vol. 30–19, 2018, pp. 1–17.

## APPENDIX A – MATRIX LIBRARY ALGORITHMS

### Algorithm A.1: SET VALUES

```

1 algorithm SET_VALUES( $n_{rows}$ ,  $n_{columns}$ , vector) :
2      $k = 0$ ;
3     for ( $i \leq n_{rows}$ ) {
4         for ( $j \leq n_{columns}$ ) {
5              $A(i, j) = \text{vector}[k]$ ;
6              $k = k + 1$ ;
7              $j = j + 1$ ;
8         }
9          $i = i + 1$ ;
10    }
11 return (Matrix A)

```

### Algorithm A.2: TRANSPOSED MATRIX

```

1 algorithm TRANSPOSED_MATRIX( $n_{rows}$ ,  $n_{columns}$ , vector) :
2     Set size of  $A^T$  as size of matrix A
3     for ( $i \leq \text{numbers of rows of } A$ ) {
4         for ( $j \leq \text{numbers of columns of } A$ ) {
5              $A^T(i, j) = A(j, i)$ ;
6              $k = k + 1$ ;
7              $j = j + 1$ ;
8         }
9          $i = i + 1$ ;
10 return (Matrix AT)

```

### Algorithm A.3: SUM OF MATRICES

```

1 algorithm SUM_OF_MATRICES(Matrices A, B) :
2     Set size of C as size of matrices A and B;
3     for ( $i \leq \text{numbers of rows of } A$ ) {
4         for ( $j \leq \text{numbers of columns of } A$ ) {
5              $C(i, j) = A(i, j) + B(i, j)$ ;
6              $j = j + 1$ ;
7         }
8          $i = i + 1$ ;
9     }
10 return (Matrix C = A + B)

```

## Algorithm A.4: SUBTRACTION OF MATRICES

```

1 algorithm SUBTRACTION_OF_MATRICES (Matrices A, B) :
2   Set size of C as size of matrices A and B;
3   for (i <= numbers of rows of A) {
4     for (j <= numbers of columns of A) {
5        $C(i, j) = A(i, j) - B(i, j)$ ;
6        $j = j + 1$ ;
7     }
8      $i = i + 1$ ;
9   }
10 return (Matrix  $C = A - B$ )

```

## Algorithm A.5: MULTIPLICATION OF MATRICES

```

1 algorithm MULTIPLICATION_OF_MATRICES (Matrices A, B) :
2   Set size of C as size of number of rows of A and columns of B;
3   if (numbers of columns of A == numbers of rows of B) {
4     for (i <= numbers of rows of A) {
5       for (j <= numbers of columns of B) {
6         for (x <= numbers of rows of B) {
7            $y = y + A(i, x) \times B(x, j)$ ;
8            $x = x + 1$ ;
9         }
10         $C(i, j) = y$ ;
11         $y = 0$ ;
12         $j = j + 1$ ;
13      }
14       $i = i + 1$ ;
15    }
16 return (Matrix  $C = A \times B$ )

```

## Algorithm A.6: MATRIX MULTIPLICATION BY A SCALAR

```

1 algorithm MATRIX_TIMES_X (Matrix A, scalar x) :
2   Set size of B as size of matrix A;
3   for (i <= numbers of rows of A) {
4     for (j <= numbers of columns of A) {
5        $B(i, j) = A(i, j) \times x$ ;
6        $j = j + 1$ ;
7     }
8      $i = i + 1$ ;
9   }
10 return (Matrix  $B = Ax$ )

```

## Algorithm A.7: MATRIX DIVISION BY A SCALAR

```

1 algorithm MATRIX_DIV_X(Matrix A, scalar x):
2   Set size of B as size of matrix A;
3   for ( $i \leq$  numbers of rows of A) {
4     for ( $j \leq$  numbers of columns of A) {
5        $B(i, j) = A(i, j) / x$ ;
6        $j = j + 1$ ;
7     }
8      $i = i + 1$ ;
9   }
10 return (Matrix B = A/x)

```

## Algorithm A.8: CREATE AN IDENTITY MATRIX

```

1 algorithm EYE(scalar x):
2   Set size of I as ( $x$  by  $x$ );
3   for ( $i \leq$  numbers of rows of I) {
4     for ( $j \leq$  numbers of columns of I) {
5       if ( $j == i$ ) {
6          $I(i, j) = 1$ ;
7       }
8       else{
9          $I(i, j) = 0$ ;
10      }
11      $j = j + 1$ ;
12   }
13    $i = i + 1$ ;
14 }
15 return (Matrix I)

```

## Algorithm A.9: CREATE A MATRIX OF ZEROS

```

1 algorithm ZEROS_MAT(scalars x, y):
2   Set size of Z as ( $x$  by  $y$ );
3   for ( $i \leq$  numbers of rows of Z) {
4     for ( $j \leq$  numbers of columns of Z) {
5        $Z(i, j) = 0$ ;
6        $j = j + 1$ ;
7     }
8      $i = i + 1$ ;
9   }
10 return (Matrix Z)

```

## Algorithm A.10: CREATE A MATRIX OF ONES

```

1 algorithm ONES_MAT(scalars  $x, y$ ):
2   Set size of  $O$  as ( $x$  by  $y$ );
3   for ( $i \leq$  numbers of rows of  $O$ ) {
4     for ( $j \leq$  numbers of columns of  $O$ ) {
5        $O(i, j) = 1$ ;
6        $j = j + 1$ ;
7     }
8      $i = i + 1$ ;
9   }
10 return (Matrix  $O$ )

```

Algorithm A.11: CREATE A  $3 \times 3$  SKEW-SYMMETRIC MATRIX

```

1 algorithm SKEW_MAT(scalars  $x, y, z$ ):
2   Set size of  $S$  as (3 by 3);
3    $S(0, 0) = 0$ ;
4    $S(0, 1) = -z$ ;
5    $S(0, 2) = y$ ;
6    $S(1, 0) = z$ ;
7    $S(1, 1) = 0$ ;
8    $S(1, 2) = -x$ ;
9    $S(2, 0) = -y$ ;
10   $S(2, 1) = x$ ;
11   $S(2, 2) = 0$ ;
12 return (Matrix  $S$ )

```

## Algorithm A.12: BLOCK DIAGONAL CONCATENATION (2 by 2)

```

1 algorithm BLKDIAG2_MAT(Matrices  $A, B$ ):
2   Set size of  $M$  as ( $[$ row of  $A$  + row of  $B$  $]$   $\times$   $[$ column of  $A$  + column of  $B$  $]$ );
3   Set all  $M$  values as zeros;
4   for ( $i \leq$  numbers of rows of  $A$ ) {
5     for ( $j \leq$  numbers of columns of  $A$ ) {
6        $M(i, j) = A(i, j)$ ;
7        $j = j + 1$ ;
8     }
9      $i = i + 1$ ;
10  }
11
12  for ( $i \leq$  numbers of rows of  $B$ ) {
13    for ( $j \leq$  numbers of columns of  $B$ ) {
14       $M([i + n_{\text{rows\_of\_}A}], [j + n_{\text{columns\_of\_}A}]) = B(i, j)$ ;
15       $j = j + 1$ ;

```

```

16     }
17     i = i + 1;
18 }
19 return (Matrix S)

```

### Algorithm A.13: CUSTOM MATRIX OF MATRICES

```

1 algorithm CUSTOM_MAT (
  Scalar  $n_{matrices}$ , Vector [confX, confY, row, column], list of matrices):
2   Set size of M as ([row × column]);
3   auxX = 0;
4   auxY = 0;
5   for (x ≤ confX) {
6     for (y ≤ confY) {
7       A ← get a matrix from list;
8       for (i ≤  $n_{rows\_of\_A}$ ) {
9         for (j ≤  $n_{columns\_of\_A}$ ) {
10          M([i + auxX], [j + auxY]) = A(i, j);
11          j = j + 1;
12        }
13        i = i + 1;
14      }
15      auxY = auxY +  $n_{columns\_of\_A}$ ;
16      if (auxY ≥  $n_{columns\_of\_M}$ ) {
17        auxY = 0;
18      }
19      y = y + 1;
20    }
21
22    auxX = auxX +  $n_{rows\_of\_A}$ ;
23    if (auxX ≥  $n_{rows\_of\_M}$ ) {
24      auxX = 0;
25    }
26    x = x + 1;
27  }
28 return (Matrix M)

```

### Algorithm A.14: DELETE ROW AND COLUMN OF A MATRIX

```

1 algorithm DEL_RC_MAT (Matrix A, Scalars r, c and order):
2   i = j = g = h = 0;
3   for (g < order) {
4     for (h < order) {
5       if (g! = r && h! = c) {

```

```

6         B[i][j++] = A[g][h];
7         if (j == order - 1) {
8             j = 0;
9             i = i + 1;
10        }
11    }
12 }
13 }
14 return (Matrix B)

```

#### Algorithm A.15: DETERMINANT OF A MATRIX

```

1 algorithm DET_MAT(Matrix A, Scalar order) :
2     sign = 1;
3     if (order == 1) {
4         det = A[0][0];
5     }
6     if (order == 2) {
7         det = A[0][0] × A[1][1] - A[0][1] × A[1][0];
8     }
9     if (order > 2) {
10        f = 0;
11        for (f < order) {
12            C = DEL_RC_MAT(A, 0, f, order); → Algorithm A.14
13            d = (sign × A[0][f]) × (DET_MAT(C, (order - 1)));
14            det = det + d;
15            sign = -sign;
16        }
17    }
18 return (Scalar det)

```

#### Algorithm A.16: INVERSE OF A MATRIX

```

1 algorithm INVERSE_MAT(Matrix A) :
2     order = nrows_of_A;
3     if (order == 1) {
4         Set B size as (1 × 1);
5         B[0][0] = 1/A[0][0];
6     }
7     if (order == 2) {
8         Set B size as (2 × 2);
9         det = DET_MAT(A, order); → Algorithm A.15
10        B[0][0] = A[1][1];
11        B[0][1] = -A[0][1];

```

```

12     B[1][0] = -A[1][0];
13     B[1][1] = A[0][0];
14     B = B × (1/det); → Algorithm A.6
15 }
16 if (order > 2) {
17     Set B size as (order × order);
18     Set factor size as (order × order);
19     Set X size as (order × order);
20     Set Y size as (order × order);
21     p = q = 0;
22     for (q < order) {
23         for (p < order) {
24             k = n = 0;
25             i = j = 0;
26             for (i < order) {
27                 for (j < order) {
28                     if (i! = q && j! = p) {
29                         X[n][k] = A[j][i];
30                         if (n < (order - 2)) {
31                             n = n + 1;
32                         }
33                         else{
34                             n = 0;
35                             k = k + 1;
36                         }
37                     }
38                 }
39             }
40             if (p == 0 && q == 0) {
41                 factor[p][q] = DET_MAT(X, (order - 1)); → AlgorithmA.15
42             }
43             else{
44                 if ((p + q)%2 == 0) {
45                     factor[p][q] = DET_MAT(X, (order - 1)); → AlgorithmA.15
46                 }
47                 else{
48                     factor[p][q] = -DET_MAT(X, (order - 1)); → AlgorithmA.15
49                 }
50             }
51         }
52     }
53     g = h = 0;

```



```

54   for (g < order) {
55       for (h < order) {
56           Y[h][g] = factor[g][h];
57       }
58   }
59   d = DET_MAT(A, order); → Algorithm A.15
60   g = h = 0;
61   for (g < order) {
62       for (h < order) {
63           B[h][g] = Y[h][g]/d;
64       }
65   }
66
67 }
68 return (Matrix B)

```

#### Algorithm A.17: COPY OF A MATRIX

```

1 algorithm COPY_MAT(Matrix A):
2   Set size of B as size of matrix A;
3   for (i <= numbers of rows of A) {
4       for (j <= numbers of columns of A) {
5           B(i, j) = A(i, j);
6           j = j + 1;
7       }
8       i = i + 1;
9   }
10  return (Matrix B)

```

#### Algorithm A.18: RANK OF A MATRIX

```

1 algorithm RANK_MAT(Matrix A):
2   rank ← column of matrix A;
3   r = 0;
4   for (r < rank) {
5       if (A[r][r] ≠ 0) {
6           c = 0;
7           for (c < nrows_of_A) {
8               if (c ≠ r) {
9                   m = A[c][r]/A[r][r];
10                  i = 0;
11                  for (i < rank) {
12                      A[c][i] = m × A[r][i];
13                      i = i + 1;

```

```

14         }
15     }
16     c = c + 1;
17 }
18 }
19 else{
20     reduce = 1;
21     i = r + 1;
22     for (i < n_rows_of_A) {
23         if (A[i][r] != 0) {
24             j = 0;
25             for (j < rank) {
26                 t = A[r][j];
27                 A[r][j] = A[i][j];
28                 A[i][j] = t;
29                 j = j + 1;
30             }
31             reduce = 0;
32             break;
33         }
34         i = i + 1;
35     }
36     if (reduce == 1) {
37         rank --;
38         i = 0;
39         for (i < n_rows_of_A) {
40             A[i][r] = A[i][rank];
41             i = i + 1;
42         }
43     }
44     r --;
45 }
46 r = r + 1;
47 }
48 return (Scalar rank)

```

#### Algorithm A.19: CHECK SYMMETRY OF A MATRIX

```

1 algorithm SYM_MAT (Matrix A) :
2     Set matrix cp ← same size as matrix A;
3     Set matrix tr ← same size as matrix A;
4     sum = 0;
5     if (n_rows_of_A != n_columns_of_A) {

```

```

6     sym = 0;
7 }
8 else{
9     i = 0;
10    for (i < nrows_of_A) {
11        j = 0;
12        for (j < ncolumns_of_A) {
13            cp[i][j] = A[i][j];
14            j = j + 1;
15        }
16        i = i + 1;
17    }
18    check = nrows_of_A × ncolumns_of_A;
19    tr = cp';
20    i = 0;
21    for (i < nrows_of_A) {
22        j = 0;
23        for (j < ncolumns_of_A) {
24            if (cp[i][j] == tr[i][j]) {
25                sum ++;
26            }
27            j = j + 1;
28        }
29        i = i + 1;
30    }
31    if (check == sum) {
32        sym = 1;
33    }
34    else{
35        sym = 0;
36    }
37
38 }
39 if (sym == 0) {
40     False;
41 }
42 else{
43     True;
44 }
45 return (Bool ← True or False)

```

```

1 algorithm LU_MAT(Matrix A) :
2   Set matrix L size the same as matrix A;
3   Set matrix U size the same as matrix A;
4   Set matrix L values to zero;
5   Set matrix U values to zero;
6   Set  $i = 0, j = 0, k = 0$ ;
7   for ( $i < n_{rows\_of\_A}$ ) {
8     for ( $j < n_{column\_of\_A}$ ) {
9       if ( $j < i$ ) {
10         $L[j][j] = 0$ ;
11      }
12      else{
13         $L[j][j] = A[j][j]$ ;
14        for ( $k < i$ ) {
15           $L[j][j] = L[j][j] - L[j][k] \times U[k][j]$ 
16        }
17      }
18    }
19  }
20  Set  $i = 0, j = 0, k = 0$ ;
21  for ( $i < n_{rows\_of\_A}$ ) {
22    if ( $j < i$ ) {
23       $U[j][j] = 0$ ;
24    }
25    else if ( $j == i$ ) {
26       $U[j][j] = 1$ ;
27    }
28    else{
29       $U[j][j] = A[j][j]/L[j][j]$ ;
30      for ( $k < i$ ) {
31         $U[j][j] = U[j][j] - [(L[j][k] \times U[k][j])/(L[j][j])]$ ;
32      }
33    }
34  }
35 return (Matrices L and U)

```

#### Algorithm A.21: GET A MATRIX DIAGONAL

```

1 algorithm DIAG_MAT(Matrix A) :
2   Set  $i = 0$ ;
3   for ( $i < n_{rows\_of\_A}$ ) {
4      $v[i] = A[i][i]$ ;
5   }

```

```
6 return (Vector v with matrix A diagonal values)
```

### Algorithm A.22: GET SECTION OF A MATRIX

```
1 algorithm SEC_MAT(MatrixA, scalars Ri, Rf, Ci, Cf) :
2   Set Matrix B row size ← Rf − Ri;
3   Set Matrix B row size ← Cf − Ci;
4   Set i = 0 and j = 0;
5   for (i < nrows_of_B) {
6     for (j < ncolumns_of_B) {
7       B[i][j] = A[i + Ri][j + Ci];
8     }
9   }
10 return (Matrix B)
```

### Algorithm A.23: QR DECOMPOSITION

```
1 algorithm QR(MatrixA) :
2   Set m as number of row of matrix A;
3   Set n as number of columns of matrix A;
4   Set matrix Q as an identity matrix with size m;
5   R = COPY_MAT(A);
6   i = 0;
7   for (i < n) {
8     z = GET_PART(R, i, m, i, i) → Algorithm A.22;
9     nz ← norm of z;
10    v = [(-sign(z[1]) × norm(z) − z[1]); (-z[2 : end])] → Algorithm A.13;
11    Set l as an identity matrix of size m − i + 1;
12    P = I − 2 × (v × vT) / (vT × v);
13    R[i : m][:] = P × R[i : m][:];
14    Q[i : m][:] = P × Q[i : m][:];
15  }
16  Q = QT;
17  R is set as R with all elements above the diagonal as zeros; → Algorithm A.29;
18 return (Matrix Q and R)
```

### Algorithm A.24: MATRIX TO THE POWER OF N

```
1 algorithm POWER_MAT(Matrix A, scalar N) :
2   Set B as matrix of ones with size of A;
3   if (N == 1) {
4     B = A;
5   }
6   i = 0;
```

```

7   else{
8       for( $i < N$ ) {
9            $B = B \times A$ ;
10           $i = i + 1$ ;
11      }
12  }
13  return (Matrix B)

```

#### Algorithm A.25: TRACE OF A MATRIX

```

1  algorithm TRACE_MAT(Matrix A) :
2      Set  $n$  as number of row of matrix A;
3       $t = 0$ ;
4       $i = 0$ ;
5      for( $i < n$ ) {
6           $t = t + A[i][i]$ ;
7      }
8  return (Scalar t)

```

#### Algorithm A.26: EIGENVALUES OF A MATRIX

```

1  algorithm EIG22_MAT(Matrix A) :
2       $t = \text{TRACE\_MAT}(A) \rightarrow$  Algorithm A.25
3       $d = \text{DET\_MAT}(A) \rightarrow$  Algorithm A.15
4       $B[0][0] = (t + \sqrt{t^2 - 4d})/2$ ;
5       $B[1][0] = (t - \sqrt{t^2 - 4d})/2$ ;
6  return (Matrix B)

```

#### Algorithm A.27: EIGENVALUES OF A MATRIX ( $2 \times 2$ )

```

1  algorithm EIG_MAT(Matrix A) :
2      Set  $n$  as number of row of matrix A;
3      Set matrix  $H$  as copy of matrix A  $\rightarrow$  Algorithm A.17
4      Set  $m$  as number of iterations for estimation of matrix  $H$ ;
5       $i = 0$ ;
6      for( $i < m$ ) {
7           $[Q, R] = \text{QR}(H) \rightarrow$  Algorithm A.23
8           $H = R \times Q$ ;
9      }
10      $c = 0$ ;
11      $j = 0$ ;
12     Set  $p$  the precision of the eigenvalues estimation;
13     for( $j < n$ ) {
14         if( $c < n$ ) {

```

```

15     if ( $H[c+1][c] \leq p$  &  $H[c+1][c] > p$ ) {
16          $B[c][0] = H[c][c]$ ;
17          $c = c + 1$ ;
18     }
19     else{
20          $He = [(H[c][c] \ H[c][c+1]); (H[c+1][c] \ H[c+1][c+1])]$ ;
21          $M = \text{eig22}(He) \rightarrow \text{Algorithm A.26}$ ;
22          $B[c][0] = M[0][0]$ ;
23          $B[c+1][0] = M[1][0]$ ;
24          $c = c + 2$ ;
25     }
26 }
27 }
28 return (Matrix B)

```

---

Algorithm A.28: ZEROS ALL THE ELEMENTS BELOW THE MATRIX DIAGONAL

---

```

1 algorithm TRIL_MAT(Matrix A) :
2     Set Matrix B size same as matrix A;
3     Set  $i = 0$  and  $j = 0$ ;
4     for ( $i < n_{\text{rows\_of\_B}}$ ) {
5         for ( $j < n_{\text{columns\_of\_B}}$ ) {
6             if ( $i < j$ ) {
7                  $B[i][j] = 0$ ;
8             }
9             else{
10                 $B[i][j] = A[i][j]$ ;
11            }
12        }
13    }
14 return (Matrix B)

```

---

Algorithm A.29: ZEROS ALL THE ELEMENTS ABOVE THE MATRIX DIAGONAL

---

```

1 algorithm TRIU_MAT(Matrix A) :
2     Set Matrix B size same as matrix A;
3     Set  $i = 0$  and  $j = 0$ ;
4     for ( $i < n_{\text{rows\_of\_B}}$ ) {
5         for ( $j < n_{\text{columns\_of\_B}}$ ) {
6             if ( $i > j$ ) {
7                  $B[i][j] = 0$ ;
8             }
9             else{
10                 $B[i][j] = A[i][j]$ ;

```

```

11     }
12   }
13 }
14 return (Matrix B)

```

### Algorithm A.30: 2-NORM OF A MATRIX

```

1 algorithm 2NORM_MAT (Matrix A) :
2   i = j = 0;
3   for (i < nrows_of_A) {
4     for (j < ncolumns_of_A) {
5       x = fabs(A[i][j]);
6       sum = sum + x2;
7     }
8   }
9   n = √sum;
10 return (Scalar n)

```

### Algorithm A.31: DISCRETE-TIME ALGEBRAIC RICCATI EQUATION SOLVER

```

1 algorithm RICCATI (Matrices A, B, Q, R) :
2   Set Matrix Gk as  $BR^{-1}B$ ;
3   Set Matrix Hk equal to Q ;
4   Set Matrix Ak equal to A;
5   n = nrows_of_A;
6   Set  $\epsilon$  as cut off point in X calculation;
7   ns = 1;
8   while (ns ≥  $\epsilon$ ) {
9     Ak1 = Ak × (EYE(n) + GkHk)-1 Ak;
10    Gk1 = Gk + Ak × (EYE(n) + GkHk)-1 GkAkT;
11    Hk1 = Hk + AkTHk × (eye(n) + GkHk)-1 Ak;
12    ns = 2NORM_MAT((Hk1 - Hk))/2NORM_MAT((Hk1)) → Algorithm A.30;
13    Hk = Hk1;
14    Ak = Ak1;
15    Gk = Gk1;
16  }
17  X = Hk;
18 return (Matrix X as  $X = A^T X A - (B^T X A)^T \times (R + B^T X B)^{-1} \times B^T X A + Q$ )

```



## APPENDIX B – PID - FUZZY LIBRARY ALGORITHMS

Algorithm B.1: Algorithm for open left fuzzyfication

```

1 algorithm openLeft (x, α, β) :
2   if (x < α) {
3     return (1);
4   }
5   if (x < α_&&_x <= β) {
6     return ((β - x)/(β - α));
7   }
8   else{
9     return (0);
10  }

```

Algorithm B.2: Algorithm for open right fuzzyfication

```

1 algorithm openRight (x, α, β) :
2   if (x < α) {
3     return (1);
4   }
5   if (x < α_&&_x <= β) {
6     return ((x - α)/(β - α));
7   }
8   else{
9     return (0);
10  }

```

Algorithm B.3: Algorithm for triangular fuzzyfication

```

1 algorithm triangular(x, a, b, c) :
2   return (max(min((x - a)/(b - a), (c - x)/(c - b)), 0));

```

Algorithm B.4: Algorithm for open left defuzzyfication

```

1 algorithm areaOL(μ, start, α, β) :
2   xOL = (β - start) - μ * (β - α);
3   return (0.5 * μ * (β + xOL), β/2);

```

Algorithm B.5: Algorithm for open right defuzzyfication

```

1 algorithm areaOR(μ, end, α, β) :
2   xOR = (β - α) * μ + α;
3   aOR = 0.5 * μ * ((end - α) + (end - xOR));
4   return (aOR, (end - α)/2 + α);

```

## Algorithm B.6: Algorithm for triangular defuzzification

```

1 algorithm areaTR( $\mu, a, b, c$ ):
2    $x_1 = \mu * (b - a) + a$ ;
3    $x_2 = c - \mu * (c - b)$ ;
4    $d_1 = (c - a)$ ;
5    $d_2 = x_2 - x_1$ ;
6    $area = 0.5 * \mu * (d_1 + d_2)$ ;
7   return ( $area, b$ );

```

## Algorithm B.7: Algorithm for mathematical representation of the fuzzy rules

```

1 algorithm rules( $NL_{error}, NL_{error_{derivative}}, \dots, X_{error}, X_{error_{derivative}}$ ):
2   # RULES FOR PL OUTPUT:
3      $PL_1 = \min(NL_{error}, NL_{error_{derivative}}) \rightarrow R1$ ;
4      $PL_2 = \min(NL_{error}, NM_{error_{derivative}}) \rightarrow R2$ ;
5      $PL_{output} = \min([PL_1, PL_2])$ ;
6
7   # RULES FOR PM OUTPUT:
8      $PM_1 = \min(NL_{error}, NS_{error_{derivative}}) \rightarrow R3$ ;
9     ...
10     $PM_{output} = \min([PM_1, \dots, PM_n])$ ;
11
12  # RULES FOR PS OUTPUT:
13    ...
14     $PS_{output} = \min([PS_1, \dots, PS_n])$ ;
15
16  # RULES FOR ZE OUTPUT:
17    ...
18     $ZE_{output} = \min([ZE_1, \dots, ZE_n])$ ;
19
20  # RULES FOR NS OUTPUT:
21    ...
22     $NS_{output} = \min([NS_1, \dots, NS_n])$ ;
23
24  # RULES FOR NM OUTPUT:
25    ...
26     $NM_{output} = \min([NM_1, \dots, NM_n])$ ;
27
28  # RULES FOR NL OUTPUT:
29    ...
30     $NL_{output} = \min([NL_1, \dots, NL_n])$ ;
31
32  return ( $[PL_{output}, PM_{output}, PS_{output}, ZE_{output}, NS_{output}, NM_{output}, NL_{output}]$ )

```

## Algorithm B.8: Algorithm for defuzzification output

```
1 algorithm defuzzification(PL, PM, PS, ZE, NS, NM, NL):  
2     (areaPL, cPL) = areaOR(PL, x, y, z);  
3     (areaPM, cPM) = areaTR(PM, x, y, z);  
4     (areaPS, cPS) = areaTR(PS, x, y, z);  
5     (areaZE, cZE) = areaTR(ZE, x, y, z);  
6     (areaNS, cNS) = areaTR(NS, x, y, z);  
7     (areaNM, cNM) = areaTR(NM, x, y, z);  
8     (areaNL, cNL) = areaOL(NL, x, y, z);  
9     num = areaPL × cPL + areaPM × cPM + ... + areaNL × cNL;  
10    den = areaPL + areaPM + ... + areaNL;  
11    return (num/den)
```



Pontifícia Universidade Católica do Rio Grande do Sul  
Pró-Reitoria de Pesquisa e Pós-Graduação  
Av. Ipiranga, 6681 – Prédio 1 – Térreo  
Porto Alegre – RS – Brasil  
Fone: (51) 3320-3513  
E-mail: [propesq@pucrs.br](mailto:propesq@pucrs.br)  
Site: [www.pucrs.br](http://www.pucrs.br)