

Predictive Indoor Navigation using Commercial Smart-phones *

Balajee Kannan[†]
GE Global Research
One Research Circle
Niskayuna, NY, USA
balajee.kannan@ge.com

Felipe Meneguzzi
PUCRS
Av. Ipiranga 6681
Porto Alegre, RS, Brasil
felipe.meneguzzi@pucrs.br

M. Bernardine Dias
Katia Sycara
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA, USA
{mbdias, katia}@cs.cmu.edu

ABSTRACT

Low-cost navigation solutions for indoor environments have a variety of real-world applications ranging from emergency evacuation to mobility aids for people with disabilities. Challenges for commercial indoor navigation solutions include robust localization, intuitive recognition of user navigation goals, and efficient route-planning and re-planning techniques for resource-constrained platforms like smart-phones and mobile phones. In this paper, we present an architecture for indoor navigation using an Android smartphone that integrates observed behavior for recognizing user navigation goals and estimating future paths without direct input from the user. Our architecture contains three core components: *plan recognition*, *map representation and route planning*, and *effective localization*. To evaluate the feasibility of our solution, we develop a prototype application on a commercial smart-phone and tested it in multiple indoor environments.

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous

Keywords

Proactive Assistance, Plan Recognition

1. INTRODUCTION

Current mobile phone technology has evolved to a point where affordable smart-phones with a variety of sensors are

*This work was partially sponsored by the Google Core AI gift from Google Inc. This paper does not necessarily reflect the opinion or policy of the sponsors; no official endorsement should be inferred. The authors also thank members of the rCommerce Laboratory at Carnegie Mellon University for their valuable contribution during development and testing, as well as the RI Summer Scholars program for making the author's collaboration possible.

[†]This work was done when author Balajee Kannan was a member of Robotics Institute, Carnegie Mellon University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

Copyright 2013 ACM 978-1-4503-1656-9/13/03 ...\$15.00.

readily available to the public. While outdoor navigation is a well established technological field, analogous techniques for indoor environments are still in their infancy. The problem of indoor navigation is further complicated when applied to the resource-constrained domain of commercial smart-phones. Limited computing power, restrictive memory access and the diversity of platform requires innovative adaptation of navigation algorithms to achieve a practical solution.

In this paper, we describe an architecture for indoor navigation using commercial smart-phones that integrates behavior recognition, multi-sensor indoor localization, and path-planning to pro-actively provide directions without direct input from users. The rest of the paper is organized as follows. In Section 2 we briefly review research efforts related to this paper. In Section 3 we describe the application architecture and detail the individual components, subsequently describing the experimentation results in Section 4. Finally, in Section 5, we conclude the paper with a summary of the results and outline future work directions.

2. RELATED WORK

Given a model of a user's planning process and a series of observations about recent actions, plan recognition and prediction is the process of identifying the plan a user is currently executing and predicting future steps [1]. Our system uses a model of plan recognition based on recent work on predicting human planning under the assumption of near-optimal rationality [11] within a Markov Decision Process (MDP). An MDP is a discrete-time stochastic control process where at each time step a rational agent transitions state based partially on its choice of action and partially on a random component. In order to use this model for intention prediction, previous work [14] has altered the MDP solution equations to generate a *stochastic policy*, whereas traditional MDP solutions consist of an *optimal policy*. Instead of returning the optimal action for each state of the world, a stochastic policy returns the *probability* with which an imperfectly rational agent selects an action. This approach assumes that a user, although rational, does not always select the *optimal* action, but rather selects an action with a probability proportional to its optimality.

Hierarchical maps are a popular technique to represent environments for dynamic domains such as robotics [4] and transit planning networks [6]. This representation reduces the search space to the sufficient sub-graph that is needed for a particular search. As the search goes down the hierarchy of the graph, once a node of interest, based on a

defined heuristic, is selected, the other nodes, with their whole sub-graphs and related arcs, are discarded. Accordingly, it reduces the computational cost from exponential to linear in the best case [4]. For route planning applications on a smart-phone, the speed and efficiency of the algorithms are restricted by the on-board computation capabilities. An effective route planner should have both high throughput and low delays in terms of query processing, and should be capable of fast dynamic re-planning. Prior work [8] in dynamic path-planning using the D^* algorithm is highly pertinent to the domain of indoor navigation, as it is capable of planning paths in changing environments and enabling rapid re-planning if changes in travel costs are discovered. Daniel and Cagigas [4] introduce a new hierarchical extension of the D^* algorithm for robot path-planning, where a down-top strategy and a set of pre-calculated paths are used to improve performance. The outlined approach has the drawbacks of needing pre-computed paths and operating on a uniform granularity of information representation across the different nodes of the graph. Our algorithm extends the hierarchal D^* approach and outlines a top-down multi-level planner for varying map granularities.

GSM signal strengths have been shown to accurately determine which floor of a building the user is in [12], while Wifi-based RSSI fingerprinting [3, 10] can be used to estimate pose with an accuracy range of 3-10 meters. Despite the relatively high-quality of solution, a major drawback to most RSSI-based existing solutions is that they are geared towards devices with significant computation capabilities and high-fidelity sensors. Furthermore, RSSI fingerprinting techniques use a pre-built signal-strength map of the environment. Most current methods for building an RSSI-to-position database are tedious, labor-intensive, and require a large number of samples, as well as extensive recalibration. Unlike RSSI methods, a dead-reckoning system [7] comprising of accelerometer, magnetometer, and gyroscope sensors can provide fast and accurate estimation of local pose. While effective over short distances, dead-reckoning solutions have the drawback of being local estimation techniques that have to be seeded with an accurate initial position for valid estimation and has unbounded sensory error over time and distance.

3. PREDICTIVE INDOOR NAVIGATION

Our integrated predictive indoor navigation architecture consists of three major components: user prediction; path-planning and map management; indoor localization, as illustrated in Figure 1. To achieve effective localization in the absence of GPS, we combine complementary localization al-

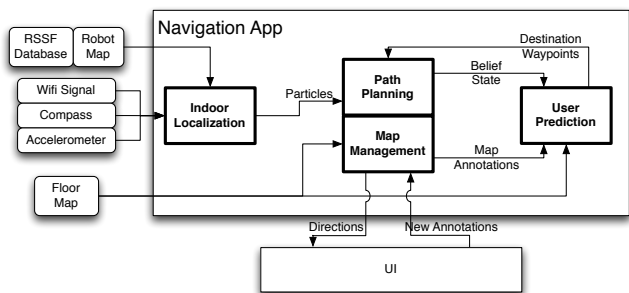


Figure 1: System Architecture

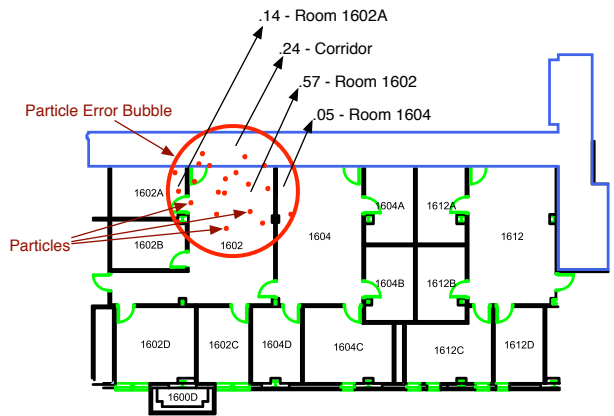


Figure 2: Particle Dispersion and Belief State.

gorithms of dead reckoning and Wifi signal strength fingerprinting. These measurements along with pre-built maps of the environment are combined using a particle filter for robust pose estimation. Towards effective route planning, we use a hierarchical map representation combined with an iterative path-planning algorithm for providing fast and efficient user specific routes. The plan recognizer uses a decision-theoretic planner to identify a user's high-level destination goals. Based on the observed state of a user's current location, the recognizer identifies potential future plans using a probabilistic tree of possible states, selects the path the user is most likely to take, and subsequently transforms it into an end user destination. These modules run independently and continuously exchange information as new data comes into the system. The data flow in the system originates with the localization module receiving information from the various sensors. The raw data, from Wifi and dead reckoning, is combined using a particle filter to provide an estimate of the user's location indicating the dispersion of probability of the user's presence within an area of the building. Once generated, the set of particles along with an uncertainty estimate is used to construct a *belief-state*, representing the probability distribution of the user's location within a building. The belief state is generated by calculating the proportion of the particles that are within any given room, generating a vector containing, the probability of the user being located in that room. This is illustrated in Figure 2. The belief state is then sent to the destination prediction module, which uses a probabilistic MDP policy to calculate the possible future paths that the user might take, sending the most likely destination to the path-planning component along with the high-level waypoints between the user's current location and it's destination. The path-planning component, in turn, uses the predicted destination as the end node for its path-planning algorithm, generating a new set of directions for the user. The User Interface (UI) component provides a visual representation of the collated information regarding current position, predicted destination, and the generated route overlaid in a floor map.

3.1 Decision-theoretic Path Prediction

Formally, an MDP is a tuple $\Sigma = \langle S, A, P, R \rangle$, where S is a finite set of *states*, A is a finite set of *actions*, P is a *state-transition function*, and R is a *reward function* [5]. The state-transition function defines the probability with which actions take the agent from one state into another.

Given $\{s, s'\} \in S$ and $a \in A$, $P_a(s'|s)$ denotes the probability of transitioning from state s to state s' when executing action a . The reward function assigns values to particular states¹. Thus, a rational agent must choose actions that maximize the long-term expected reward of its actions. A policy π is a total function $\pi : S \rightarrow A$ mapping states into actions, which allows an agent to decide which action to take at each state. One can calculate the value V^π of a state for an agent following a certain policy π , and given a certain discount factor γ , as follows (where $a = \pi(s)$):

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_a(s'|s) V^\pi(s')$$

An *optimal* policy $\pi^*(s)$ is one that maximizes this value, and can be found by various means, such as the value iteration algorithm. Value iteration uses the following two *Bellman equations* [2], where V^* and Q^* are the equations for the maximum reward of an action and a state: $Q^*(s, a) = R(s) + V^*(P(s, a))$, and $V^*(s) = \max_a Q^*(s, a)$. These equations are used iteratively as update rules until they converge, yielding an optimal policy where $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$.

As the optimal policy describes the choice of a *perfectly rational* decision maker, it is too inflexible to model a human's movement indoors. Consequently, we take inspiration from previous work [14, 11] to generate a *stochastic policy* $\pi^\approx : S \times A \rightarrow \mathbb{R}$ that returns the *probability* with which an imperfectly rational agent selects an action. The intuition behind the notion of stochastic policy is that, although a human is not perfectly rational, (s)he will select an action close to optimal, with a probability proportional to the reward value of this action against the alternatives. For example, if humans are faced with two choices with rewards 10 and 10.1, it is fair to assume that they will choose either one of the alternatives about half of the time, whereas if the two choices have rewards 10 and 1, they will choose the higher reward much more often. Using this approach, we solve the MDP problem using the policy iteration algorithm without modification to obtain an optimal policy, while keeping track of the long-term values of each state. From Q-values obtained by value iteration we calculate a stochastic policy as follows:

$$\pi^\approx(a|s) = \frac{Q^*(s, a)}{\sum_{a' \in A} Q^*(s, a')}$$

We assume that the state transition function depends on the topology of the indoor environment, whereas the reward function depends on individual profiles of human users. In this way, individual profiles can be created either manually, with a user specifying "favorite locations", or through learning by gathering data on a user's daily routine and applying an expectation maximization algorithm on this data [14].

The process of predicting future activities starts with a *belief state* containing probabilities of the agent being in a certain state. Formally, the belief state is a function from the MDP states into probability values $B : S \rightarrow [0, 1]$, where $B(s)$ returns the probability that the user is in state s . From this belief state and the obtained stochastic policy, we create a multi-rooted tree representing the possible future state sequences starting from likely state. The root node of this tree is connected to the states whose probability in the belief state are above a minimum threshold, as illustrated in

¹In some representations, rewards are assigned to state-action pairs, but it is trivial to convert from one to the other.

Algorithm 1 Predicting future user paths.

```

1: function PREDICTFUTUREPATHS( $B, \pi^\approx, thr, maxD$ )
2:    $n \leftarrow newTreeNode(nil, nil, 1, 0)$ 
3:   for all  $s$  such that  $B(s) > thr$  do
4:      $n' \leftarrow newTreeNode(nil, s, B(s), 0)$ 
5:     CREATEPLANTREE( $n', \pi^\approx, s, thr, 1, maxD, \emptyset$ )
6:   end for
7:   return  $n$ 
8: end function
9: procedure CREATEPLANTREE( $n, \pi^\approx, s, thr, d, maxD, visited$ )
10:  if  $d \leq maxD$  then
11:    for all  $a \in A$  such that  $\pi^\approx(s, a) \geq thr$  do
12:       $p \leftarrow \pi^\approx(s, a)$ 
13:       $node\ n' \leftarrow newTreeNode(a, s, p, d)$ 
14:       $n.ADDCHILD(n')$ 
15:       $v' \leftarrow visited \cup s$ 
16:      for all  $s'$  such that  $(P_a(s, s') > 0) \wedge$ 
           $(s'.getValue > s.getValue) \wedge$ 
           $(s' \notin visited)$  do
17:        CREATEPLANTREE( $n', \pi^\approx, s', thr, d + 1,$ 
           $maxD, v'$ )
18:      end for
19:    end for
20:  end if
21: end procedure

```

Algorithm 1. The algorithm takes as input a belief state B , a stochastic policy π^\approx , a probability threshold thr denoting the minimum probability for a state to be considered relevant for plan prediction and a maximum plan depth $maxD$ denoting the maximum number of future steps to predict.

The possible sequence of next states is computed recursively, by querying the actions a user is likely to take. This is illustrated in the CREATEPLANTREE procedure, which starts by checking that the maximum plan depth has not been reached, followed by selecting all actions that the user would choose with a probability higher than a set threshold. These actions lead to potential next states, and for each possible next state, this procedure repeats recursively. Notice that, in order to avoid creating loops in the predicted paths, we keep track of the nodes already visited. The successor states are computed assuming the user follows the gradient of increasing rewards towards an objective.

The tree resulting from the CREATEPLANTREE procedure contains a set of the most likely paths a user might take by following a stochastic policy describing his/her imperfect rationality. In order to provide a target destination to the path-planning algorithm, we need to select a single path that the user is most likely to take. Our approach, illustrated in Algorithm 2, consists of selecting the non-cyclic path with the largest long-term reward. The first level of the tree generated from Algorithm 1 contains a number of subtrees with starting states comprising the most likely current user states. Moreover, for each state s in the MDP model, we keep the long-term expected value $V^*(s)$ computed through value iteration. Given these possible initial states, and the value of subsequent states, the approach we follow is to calculate the maximum reward of each path, and weigh this reward by the probability of the user being in the state stored in each node at level 1 in the tree. The leaf node from the highest reward path then becomes the predicted destination.

For example, consider a policy π_a generated for a sample environment whereby the rooms are highlighted in the map subsection shown in Figure 3b have a reward of 10, and all

Algorithm 2 Predicted user’s destination.

```

1: function COMPUTEDESTINATION(PlanTree, V*)
2:   maxR  $\leftarrow$  0
3:   for all path  $P = [n_0, \dots, n_m] \in \text{PlanTree}$  do
4:      $r \leftarrow n_0.p * \sum_{n_1}^{n_m} V^*(n_m)$ 
5:     if  $r > \text{maxR}$  then
6:       maxR  $\leftarrow$   $r$ 
7:       bestPath  $\leftarrow$   $P$ 
8:     end if
9:   end for
10:  return  $n_m \in \text{bestPath}$ 
11: end function
  
```

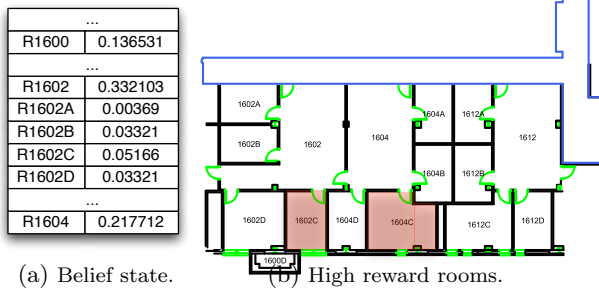


Figure 3: Example inputs.

other rooms have no reward. Additionally, we define a probability threshold value of 0.1 and a maximum depth of 10 future steps. Moreover, consider a belief state B_b with the probability of the user being in the rooms as highlighted in Figure 3a. Consequently, only the states having a probability higher than 0.1 are added to the tree (*i.e.* R1602 and R1604), which is then expanded using `CREATEPLANTREE` function. This expansion considers all actions that a user might choose to reach other rooms results in the tree of Figure 4, in which high reward paths are highlighted in green.

In order to select the most likely path predicted for the user, we sum the reward of every node on each path and weigh it by the probability of the state upon which it starts. In our example, Path 1’s root is R1602 and Path 2’s root is R1604. Note that based on Figure 3a, these states were picked because they have a probability value higher than the defined threshold. Since R1604 has the lower probability between the two, the expected reward of the path starting on that state is less than for the alternative path, R1602C is selected as the predicted state.

3.2 Hierarchical Path-planning

We use a hierarchal representation to encapsulate the en-

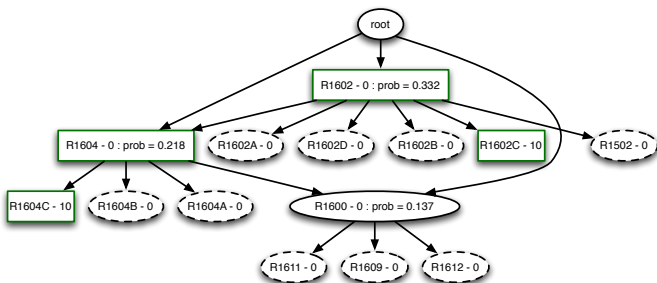


Figure 4: Predicted plan tree, given belief state of Figure 3a.

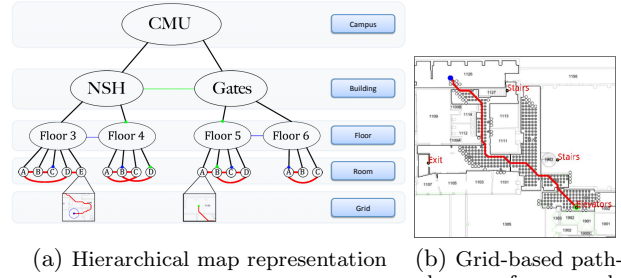


Figure 5: Hierarchical path-planning

vironment at two levels of granularity, as follows (see Figure 5a): Low-level maps are used to represent individual rooms in a large building with enhanced spatial detail, without representing the spatial relationships to locations in other rooms. High-level maps are then used to represent larger areas of a building without representing the exact spatial relationship of locations within rooms and corridors. Thus, the high-level maps are used to construct abstract plans to navigate between floors and rooms of a building whereas the higher resolution of the low-level maps are for precise navigation within rooms and hallways. Topologically, high-level maps are represented as a graph and low-level maps as a grid structure connected to the leaf nodes of the graph, each of which employed by a separate and iterative planner: a high-level planner and a grid planner. The high-level A^* -based planner handles building-wide connections accounting for exits, transitions points between floors (stairs, elevators), and planning across multiple floors. Further, the planner provides a restricted path for the grid planner, constraining the search space and reducing the computational overhead.

Similarly to many path-planning that treat a grid-based map as a graph, we leverage a D^* -type algorithm. D^*lite is a dynamic path-planner capable of handling changing environments in an efficient, optimal, and complete manner that searches for an optimal path from the destination to the start node. The grid planner (D^*lite) operates at the individual floor level and is used to find a fine-grained path to either the destination (if it is on the same floor) or an exit (if the destination is on another floor). This improves efficiency as the destination node is fixed, but the start node varies with the change in user’s location. Specific to our system, at the lowest level of the hierarchy there are rooms corresponding to vertices used in D^*lite ; rooms are connected by doorways, which correspond to the edges used in D^*lite . The hierarchy extends upwards to the floor level and building level, with floors and buildings corresponding to the vertices while exits (staircases and elevators) and roads corresponding to the edges at each respective level.

High-level Planner Based on user-location and the intended destination, the planner quickly searches the graph hierarchy, above the user and destination levels, until it finds a connecting node. After a route is established, the planner recursively searches for the shortest path through the network of edges and vertices moving down one step at a time to verify connections.

Grid Planner Once the high-level planner is run and restricted path generated, we then run an instance of the grid planner. By constraining the search space of the grid plan-

ner to only the generated path, we reduce the computational overhead of the grid planner. The planner has two core lists, *OPEN* and *CLOSED*, that determine whether the node has been added to the tree and subsequently evaluated. Optimality is determined by identifying the node, M , in *OPEN* that minimizes a defined cost function, $f(M)$, while ensuring all neighboring nodes to M are not blocked by a barrier. In the event that two nodes have an equal cost, a shortest-distance-to-goal heuristic is used for node selection. Algorithm 3 details the node selection criteria.

We further optimize the grid planner for the smart-phone platform by modifying the re-planning step of the algorithm by actively time-stamping the nodes that are moved into the *OPEN* or *CLOSED* lists. When re-planning after an arbitrary time T , we can quickly roll back nodes with a *CLOSED* time-stamp greater than a T , re-assign them to the *OPEN* set, remove all node with an *OPEN* time-stamp greater than T , and re-run the planners. This significantly reduces the computational overhead for re-planning. The adaptations to the search space and re-planning allows us to significantly speed-up the process of path-planning, especially across multiple floors.

Consider the following illustrative example. If the high-level planner determines that the user is on floor 1 of a building but the identified destination is on floor 3 of the same building, it will search the hierarchal graph until it finds that they are both in the same building (the common node). It will then verify the path at the next lowest level by identifying the connecting stairwell or elevator between floors 1 and 3. The high-level planner continues to run at the lowest level (the room level) until it finds the connection through the shortest number of rooms. It then restricts the search space of the grid planner to rooms along that path, providing a starting node and a destination node as well as a restricted path of rooms in which the algorithm can run.

3.3 Indoor Localization

We build on the general ideas of dead-reckoning and RSSI fingerprinting, from which we derive a baseline implementation using a particle filter and a k-nearest neighbor approach. A gait-based motion model combined with a heading estimator provides a pre-filtered dead-reckoning sensor estimate to the particle filter (PF) (see Figure 6). Simultaneously, pose

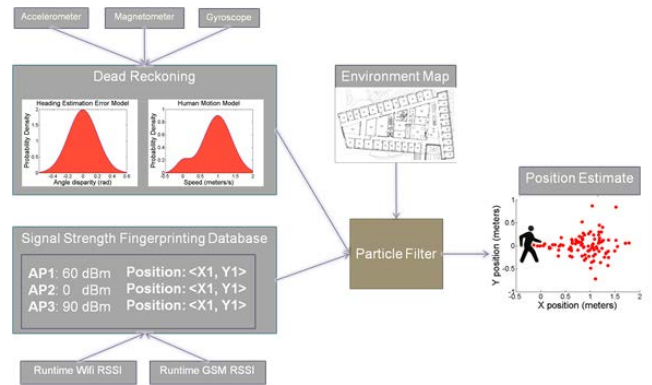


Figure 6: Localization System Architecture

is estimated based on fingerprinting between observed Wifi signal strength readings and pre-collected database of RSSI estimates. The combined sensor data is fused and filtered using a PF which results in a smooth and continuous pose estimation state. At runtime, Wifi signal strength fingerprinting is used to initialize the system and provide a rough global location estimate. The dead reckoning is performed in a pre-processing step, and all the particles in the filter are periodically updated based on a model of the variance of the dead reckoning estimate. The details of the localization technique are outlined in earlier work [9]. While the notion of combining Wifi fingerprinting with dead-reckoning using either a Particle filter or Kalman filter has been explored priorly [13], the uniqueness of our approach lies in the implementation of the complementary nature of the solution as well as in the efficient adaptation to the smart-phone platform. The adaptations allow for a fast and high-quality solution at a relatively low computation cost.

To perform the RSSI fingerprinting, it is necessary to create a database of signal strength information from the environment correlated to a free space map of the environment. Collecting this information by hand is laborious and error-prone, so we developed an automated solution that uses a robot equipped with a SICK LMS200 laser rangefinder and fiber optic gyroscope to collect the signal measurements. A phone is placed on the robot and the robot is tele-operated through the environment. The phone collects signal strength information over the course of the run, and at discrete intervals (1m) the readings from the phone are correlated with measured robot positions. Additionally, the robot builds a map of the free space in the environment using the laser rangefinder. The result is an accurate, high density sample of signal strength information in a short amount of time. Moreover, the shape and structure of the built laser map allows us to incorporate an additional heuristic to speed up our pose estimation algorithm. Particles from the PF that lie outside the bounds of the free space map are automatically discarded, allowing us to constrain the size of our comparison database during execution.

3.4 Implementation on a mobile platform

We developed a stand-alone application implementing the outlined system for an Android-based smart-phone. The primary issue encountered during development was load balancing between the different system components. Consequently, the resource intensive components of the system were developed as background services integrated using the

Algorithm 3 Path-Planning Algorithm Pseudo-code

```

1: function PLANPATH(Start, Goal)
2:   OPEN  $\leftarrow$  Goal           % If Goal is initially empty, exit
3:   while ( OPEN is not empty ) do
4:     Node N  $\leftarrow$  OPEN.getBestNode()
5:     Move N from OPEN to CLOSED
6:     for all (Node K : N.getNeighbors() ) do
7:       if K = Start then
8:         N.predecessor  $\leftarrow$  K
9:         cost(K)  $\leftarrow$  cost(N) + distance(N,K)
10:        return true           % Path Found
11:      else if K is not in OPEN or CLOSED then
12:        N.predecessor  $\leftarrow$  K
13:        OPEN  $\leftarrow$  K
14:        cost(K)  $\leftarrow$  cost(N) + distance(N,K)
15:      end if
16:    end for
17:  end while
18:  return false               % No Path
19: end function

```

main application that displays the user interface. The relatively simplistic interface provides the user with options to toggle path-prediction, select a destination, add annotations, manually re-plan the route, etc. Data communication between the different modules is handled via Android’s broadcast system. Using this system, a service sends a broadcast update that can be accessed by any registered application. This push-based data service affords an added layer of robustness to the system, in the event of computation back-log. In such an implementation, each of the different modules acts as an independent application depending only on periodic communication to further its own state. The prediction component contains a high-computation algorithm, namely the solution of the MDP problem in order to generate the stochastic policy needed for future path prediction. Under normal operation, the results of an MDP solver are not necessarily degraded by the amount it takes to finish processing. However, in situations where annotations change at a high rate, the lack of a correct MDP policy substantially degrades the user experience, since proactive navigation directions will be either non-existent, or outdated. To overcome this challenge, we have designed a remote MDP solver that can be used by the mobile application on request. Whenever the remote MDP-solver is available, value iteration is run on the server, speeding up prediction.

3.5 Automated Map Translation

We use the Scalable Vector Graphics (SVG) graphic representation (a W3C standard) for the floor plans of the building to model the indoor environment and extract a logical layout of the building. Using SVG we extract an interconnected network of rooms based on their adjacency information, which is useful to both the path-planner and the destination prediction modules. As we have seen in Section 3.1, an MDP problem is composed of a set of states, a set of actions, a transition function and a reward function. Thus, we generate the set of states for the MDP using the resulting set of rooms from the vector map representation, as well as the set of actions within the MDP using the connections between the rooms, with each doorway representing the action of moving from one room to another through it, and vice-versa. The transition function from this set of actions and states is generated by using a small ϵ “error” rate, representing a user’s indecision between moving from one room to the other, so that the probability of a user transitioning from one room to another is $1 - \epsilon$.

4. EXPERIMENTS AND RESULTS

We tested the system on a single floor of an indoor environment (see Figure 7) to analyze the feasibility of the implemented solution. Three sets of experiments involving navigation from a defined start to multiple destination locations were performed. Each set of experiments was repeated multiple times for consistency, for a total of 20 runs. There were approximately 20 Wifi access points (Figure 3b) observed in the environments. The dead-reckoning information was obtained at a faster rate (30Hz) than the Wifi signal measurement (1 HZ). An RSSI database was constructed prior to experimentation using a Pioneer 3DX robot, along with a laser map of the environment. In Experiment 1, the user heads towards destination 1 (room 1602 in Figure 7), while in experiment 2, the user heads towards destination 2 (room 1513). Finally, in experiment 3, the user starts head-

ing towards destination 2, but changes intent and heads to destination 1 during the course of the experiment. These routes are illustrated in Figure 7. Once the prediction module identifies the user destination intent, it is communicated to the path-planning module.

Table 1 outlines the profiled times for loading the environment map and for running path-planner on it. For path-planning, we compute a path stretched across the entire map. From the table we can see that the time taken to process a new map of relatively high-granularity is around 47s. Looking at the path-planning component we can see that while it takes a relatively large amount of time for the initial plan, future re-planning is significantly faster. We make the assumption that the base topography remains the same, so a connected graph is built once, and after the initial processing load times and re-planning drop dramatically.

In order to focus on analyzing the architecture, data from runs suffering from significant magnetic distortion was disregarded. The time to convergence is the time taken by the prediction module from initialization to when the first prediction was communicated to the user. The prediction is communicated as a destination location and the associated route. The prediction time includes time for map loading, initial pose estimation, belief state identification and intent recognition. After the initial recognition, subsequent recognitions happen at a significantly faster rate, thereby reducing overhead. For Experiment 1, the system predicted 3 state changes, where each state change correlates to an alternate destination than what is currently highlighted. For our experiments, the system toggled between destinations 1 and 2. The initial recognition of a state change took about 43s, while re-computation took approximately 4s. For experiment 2, there were 3 state changes with a prediction time of 53.47s. Finally, for experiment 3, the system averaged a prediction time of 30.3s and an average re-computation of 4.8s. Interestingly, the system recomputed destination locations for experiments 1 and 3, there were no re-computations required for experiment 2. We believe this is due to a higher uncertainty in the position estimation: as there are multiple routes possible to destination 1, there is a higher degree of uncertainty in the position estimate. Additional analysis shows the position with the highest mean error for the different runs to lie in the section of the corridor that constitutes a major part of paths for experiments 1 and 3, while this section of the corridor constitutes only a small portion of experiment 2 path. Despite the added complexity of predicting and planning, our re-planning algorithm is able to significantly reduce re-planning time, highlighting the effectiveness of our map-representation and path-planning, as shown in Table 2. We can see that the localization algorithm, on average, is able to converge to within 1 meter of the actual starting location for all three experiment types. Paths 1 and 3 are approximately 15m long, while in experiment 2 the user travels a total of 32m to destination. More-

Table 1: Map loading and path-planning times

| With localization and prediction | Process Time (s) | Load Time (s) | Grid Planning (ms) | Grid Re-planning (ms) |
|----------------------------------|------------------|---------------|--------------------|-----------------------|
| Yes | 46.90 | 3.58 | 749 | 22.8 |
| No | 46.90 | 3.44 | 408 | 23 |



Figure 7: Indoor Test Environment I with Selected Destinations and User Routes

Table 2: Experimental Results

| Exp | Path (m) | Start ϵ (m) | Final ϵ (m) | Mean ϵ of most prob. loc. (m) | Prediction Time (s) |
|-----|----------|----------------------|----------------------|--|---------------------|
| 1 | 16.5 | 0.71 | 3.42 | 2.20 | 43 |
| 2 | 31.67 | 0.94 | 4.26 | 2.49 | 53.47 |
| 3 | 15.46 | 0.90 | 3.34 | 2.71 | 30.324 |

over, the mean error for the localization over the course of the different experiments is approximately $3m$ and in most cases the system is also able to converge to a final destination position within an error bound of approximately $5m$. Finally, we note that the estimate of final position is of a value greater than the calculated mean error, this is likely to be because the selected destination locations lie outside the boundaries of generated robot map. Consequently, there exists a smaller sample of data points correlating to the destination locations in our RSSI database and the system is not able to converge to the final location with a higher degree of certainty.

5. CONCLUSION AND FUTURE WORK

In this paper we have developed an unique architecture for GPS-free indoor navigation and path prediction that uses a combination of multiple sensor data and probabilistic planning models to provide seamless navigation aid. To explore the feasibility of our approach we implemented a prototype solution on commercial mobile phones. The developed application was tested in indoor environments and was found to accurately predict intended destination and to effectively navigate the user to the identified destination with minimal user interaction. To our knowledge, this is the first architecture that attempts to integrate the core navigation components of path-planning and localization with intent prediction on a commercial smart-phone.

Although the current instantiation of our architecture has yielded promising results, there are a number of extensions that could be incorporated into the implementation. To enhance the indoor localization component, we intend to look at methods to combine the robot map with the building floor plan to improve the robustness and to pro-actively account for magnetic distortions. To address degradation of the built RSSI database, we will collect signal samples over multiple runs, locations, and during different times of day, and analyze it to identify the rate of decay so that it can improve the quality of the location estimate.

6. ADDITIONAL AUTHORS

Additional authors: Chet Gnegy (University of Pittsburgh, email: cng11@pitt.edu), Evan Glasgow (University of Texas,

email: eglasgow@utdallas.edu); and Piotr Yordanov (American University of Beirut, email: pay03@aub.edu.lb)

7. REFERENCES

- [1] M. G. Armentano and A. Amandi. Plan recognition for interface agents. *Artificial Intelligence Review*, 28(2):131–162, 2007.
- [2] R. E. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.
- [3] A. Bernardos, J. Casar, and P. Tarrío. Real time calibration for rss indoor positioning systems. In *Indoor Positioning and Indoor Navigation*, pages 1–7, sept. 2010.
- [4] D. Cagigas. Hierarchical algorithm with materialization of costs for robot path planning. *Robotics and Autonomous Systems*, 52(2-3):190–208, 2005.
- [5] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.
- [6] J. Hoffmann. Towards efficient belief update for planning-based web service composition. In *Proc. 18th European Conference on Artificial Intelligence*, pages 558–562, 2008.
- [7] Y. Jin, H.-S. Toh, W.-S. Soh, and W.-C. Wong. A robust dead-reckoning pedestrian tracking system with low cost sensors. In *Proc. 6th IEEE Int. Conf. on Pervasive Computing and Communications*, 2011.
- [8] G. Korsah, A. Stentz, and M. Dias. Dd* lite: Efficient incremental search with state dominance. Technical Report CMU-RI-TR-07-12, Robotics Institute, Pittsburgh, PA, May 2007.
- [9] N. Kothari, B. Kannan, E. D. Glasgow, and M. B. Dias. Robust indoor localization on a commercial smart-phone. In *Procedia Computer Science*, pages 1114–1120. Elsevier, 2012.
- [10] X. Luo, W. J. O’Brien, and C. L. Julien. Comparative evaluation of received signal strength index (rssi) based indoor localization techniques for construction jobsites. *Advanced Engineering Informatics*, 25(2):355–363, 2011.
- [11] J. Oh, F. Meneguzzi, and K. Sycara. Antipa: an agent architecture for intelligent information assistance. In *Proc. 19th European Conference on Artificial Intelligence*, pages 1055–1056, 2010.
- [12] V. Otsason, A. Varshavsky, A. LaMarca, and E. de Lara. Accurate gsm indoor localization. In *Ubiquitous Computing*, volume 3660 of *LNCS*, pages 903–903, 2005.
- [13] H. Wang, H. Lenz, A. Szabo, J. Bamberger, and U. Hanebeck. Wlan-based pedestrian tracking using particle filters and low-cost mems sensors. In *4th Workshop on Positioning, Navigation and Communication*, pages 1–7, 2007.
- [14] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proc. 23rd National Conference on Artificial Intelligence*, pages 1433–1438. AAAI Press, 2008.