Predictive Indoor Navigation using Commercial Smart-phones

Felipe Meneguzzi*, Balajee Kannan *, Katia Sycara*, Chet Gnegy†, Evan Glasgow‡, Piotr Yordanov§ and M. Bernardine Dias *
*Carnegie Mellon University, Pittsburgh, USA
Email: {meneguzz,bkannan,katia,mbdias}@cs.cmu.com
†University of Pittsburgh, Pittsburgh, USA
Email: cng11@pitt.edu
‡University of Texas, Dallas, USA
Email: eglasgow@utdallas.edu
§American University of Beirut
Beirut, Lebanon, Email: pay03@aub.edu.lb

***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 indoor navigation include robust localization in the absence of GPS, intuitive recognition of user navigation goals, and efficient route-planning and re-planning techniques. In this paper, we present an architecture for indoor navigation that integrates observed behavior for recognizing user navigation goals and estimating future paths without direct input from the user. Our architecture comprises of three core components: *effective localization*, *map representation and route planning*, and *plan recognition*. The outlined architecture is unique in its integration of the core navigation and prediction components. To evaluate the feasibility of the proposed architecture, we develop a prototype application on a commercial smart-phone. The developed application was tested in an indoor environment and was found to accurately predict intended destination and to provide effective navigation guidance to the user.**

***Keywords*-GPS-free localization; resource-constrained path-planning; MDP-based intention prediction; hierarchical map representation; mobile computing.**

I. Introduction

Current mobile phone technology has evolved to a point where affordable smart-phones with a variety of sensors are readily available to the public. The capability of the on-board sensing make them ideal for developing navigation applications, whereby relevant information is communicated to users based on their actively or passively determined location. While outdoor navigation is a well established technological field and available for most modern smart-phones, analogous techniques for indoor environments are still in their infancy.

In this paper, we describe a unique architecture for indoor navigation that integrates behavior recognition, multi-sensor indoor localization, and path-planning in order to pro-actively provide directions without direct input from users. To our knowledge, this is the first architecture that attempts to integrate the core navigation components of path-planning and localization with intent prediction towards a more refined navigation solution. The system comprises of three core components: *effective localization*, *map representation and route planning*, and *plan recognition*. To achieve effective localization in the absence of GPS, we combine complementary localization algorithms 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 D*lite [1] path-planning algorithm for providing fast and efficient user specific routes. The final component of the system is the plan recognizer, which 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. To explore the feasibility of our approach we implemented a prototype solution on commercial mobile phones. The developed application was tested in an indoor environment and was found to accurately predict intended destination and to effectively navigate the user to the identified destination with minimal user interaction.

The rest of the paper is organized as follows. In Section II we briefly review research efforts related to this paper. In Section III we describe the application architecture and detail its individual components, subsequently describing the results from the experiments conducted using the application in Section IV. Finally, in Section V, we conclude the paper with a summary of the results and outline future work directions.

II. Related Work

A. GPS-free Localization

Modern smart-phones have an extensive array of available sensors, many of which are relevant to the localization problem. These include GSM and Wifi radios, Bluetooth,

Near Field Communication (an extension of RFID), accelerometer, magnetometer, and gyroscope sensors. Such sensors can be used to estimate various parameters of a mobile user in a GPS-free environment. For example, RSSI fingerprinting for pose estimation is an accepted and popular technique for indoor pose estimation, GSM signal strengths have been shown to accurately determine which floor of a building the user is in [2]. Wifi-based RSSI fingerprinting is another popular indoor localization technique [3; 4] with an accuracy range of 3-10 meters. 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 re-calibration. Unlike RSSI methods, a dead-reckoning system comprising of accelerometer, magnetometer, and gyroscope sensors can provide fast and accurate estimation of local pose. Jin et. al., [5] propose a robust dead reckoning tracking system using a set of commercial sensors carried by the same walking pedestrian. 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. As such, the sensory error accumulation is unbounded over time and distance.

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. Our work addresses some of the above shortcomings including the use of a magnetometer, which allows the users orientation to be derived in a global frame. We also use a robotic system to collect signal strength calibration data that can significantly reduce the overhead in the creation of the fingerprint database.

B. Intention Recognition

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 [6]. Our system uses a model of plan recognition based on recent work on predicting human planning under the assumption of near-optimal rationality [7] 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 [8] 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. We discuss the details of the MDP model in more detail in Section III-C.

C. Map representation and path-planning

Hierarchical maps are a popular technique to represent environments in domains such as robotics [9] and transit planning networks [10]. 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, it gets more focused on only the sub-graph of interest to the route between the start and goal nodes. At each level, one chooses the nodes of interest to the search and discards the others with their whole sub-graphs and related arcs. Accordingly, it reduces the computational cost from exponential to linear in the best case [11]. The hierarchical graph allows for a compressed representation of large map details and is flexible to allow dynamic changes.

For route planning applications on a smart-phone, the speed and efficiency of the algorithms can be restricted by the on-board computation capabilities. Therefore, an effective route planner should have both high throughput and low delays in terms of query processing, and should be capable of dynamic re-planning. Prior work [12] 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 enabling rapid re-planning if changes in travel costs are discovered. Daniel and Cagigas [9] 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 (materialization of path costs) are used to 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. The approach is described in more detail in Section III-B.

III. Predictive Indoor Navigation System

Our integrated predictive indoor navigation architecture consists of three major components: indoor localization; path-planning and map management; and user prediction, as illustrated in Figure 1. 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

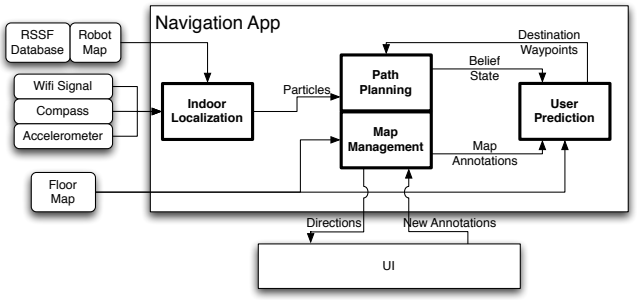


Figure 1: System Architecture

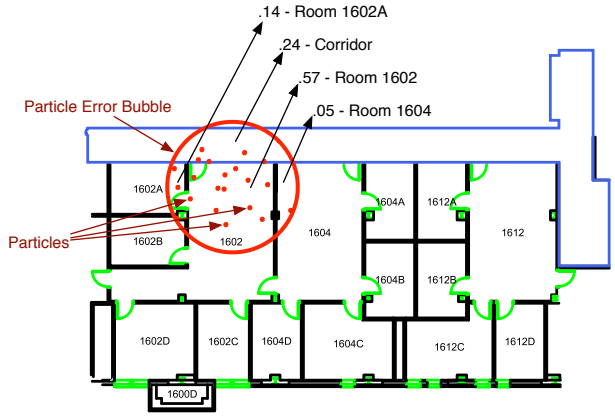


Figure 2: Particle Dispersion and Belief State.

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 to the user a visual representation of the collated information regarding current position, predicted destination, and the generated route overlaid in a map of the floor.

A. Indoor Localization

After considering the benefits and drawbacks of the various sensors, we selected the Wifi radio combined with dead reckoning using accelerometer, magnetometer, and gyroscope sensors to provide indoor localization fixes. A

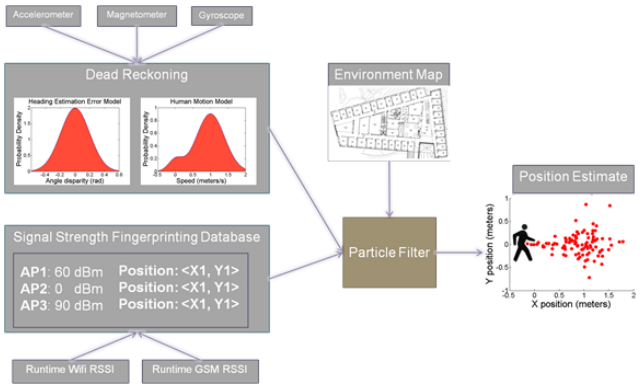


Figure 3: Localization System Architecture

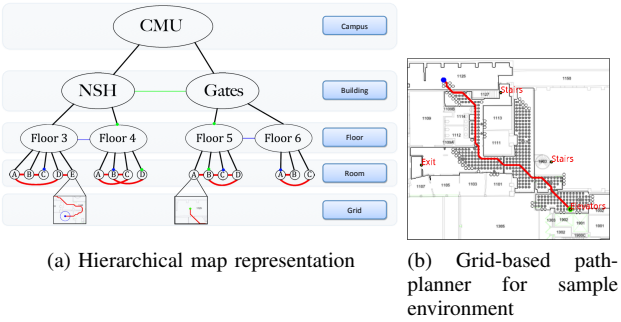
gait-based motion model combined with a heading estimator provides a pre-filtered dead-reckoning sensor estimate to the particle filter (PF) (see Figure 3). Simultaneously, pose is estimated based on fingerprinting between observed Wifi signal strength readings and pre-collected database of RSSI estimates. 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. To minimize the computational requirements of the solution, it is desirable to keep the dimensionality of the particle filter as low as possible. For this reason, we do not track heading within the filter, but estimate only the linear position. 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 [13].

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

(a) Pioneer Robot for Data Collection

(b) Overlaid Data Points on Robot Generated Map

Figure 4: RSSI Database Generation



(a) Hierarchical map representation

(b) Grid-based path-planner for sample environment

Figure 5: Hierarchal path-planning

our comparison database during execution. Figures 4a and 4b show the robot setup and a sample RSSI database.

B. Hierarchical path-planning

In our approach, we use a hierarchal graph to represent the environment at varying granularities over the different levels of the graph. For this application, we consider a two-layer hierarchy (see Figure 5a. Low-level maps are used to represent individual rooms in a large building with great spatial detail, without having to represent the spatial relationships to locations in other rooms. On the other hand, high-level maps are used to represent larger areas of a building without representing the exact spatial relationship of individual locations inside rooms and corridors. Consequently 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 essential for precise navigation within rooms and hallways. This multi-level approach allows for multi-floor planning and can be extended to planning between buildings.

Topologically, in our system the high-level maps are represented as a graph and the low-level maps as a grid structure connected to the leaf nodes of the graph. Since most path-planning algorithms treat the grid-like map as a connection of nodes and edges, a planning approach like the D^* can be leveraged for our domain. D^*lite is a dynamic path-planner capable of handling changing environments in an efficient, optimal, and complete manner. The D*lite planning algorithm searches for an optimal path from the destination to the start node. This improves efficiency as the destination node is fixed, but the start node varies with the change in user's location.

For our system, we developed two separate and iterative planners, a high-level planner and a grid planner. The high-level planner handles building-wide connections and provides a restricted path for the grid planner. The grid planner operates at the individual floor level and is used to find a fine-grained path to either the destination (should it be on the same floor) or an exit (if the destination is on another floor). Specific to our system, at the lowest level of the hierarchy there are rooms which correspond to vertices used in the D*lite algorithm. These 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: The first step in the path-planning process is to run the 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. Once 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. 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 1 details the node selection criteria.

We 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.

Consider the following illustrative example. If the high-level planner determines that the user is on floor 1 of a

Algorithm 1 Path-Planning Algorithm Pseudo-code

```
1: function PLANPATH(Start, Goal)
2:     OPEN ← Goal                          ▷ If Goal is initially empty, exit
3:     while ( OPEN is not empty ) do
4:         Node N ← 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 ← K
9:                 cost(K) ← cost(N) + distance(N,K)
10:                return true                          ▷ Path Found
11:            else if K is not in OPEN or CLOSED then
12:                N.predecessor ← K
13:                OPEN ← K
14:                cost(K) ← cost(N) + distance(N,K)
15:            end if
16:        end for
17:    end while
18:    return false                                     ▷ No Path
19: end function
```

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.

C. 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* [14]. 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 states[1]. 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 $a = \pi(s)$):

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

The *optimal* policy $\pi^*(s)$ is then 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* [15], where V^* and Q^* are the equations for the maximum reward of an action and a state.

$$Q^*(s, a) = R(s) + V^*(P(s, a))$$

$$V^*(s) = \max_a Q^*(s, a)$$

The results of these equations are calculated iteratively until they converge, and an optimal policy is one where:

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a)$$

Here, we adapt previous work [7; 8] to generate a a *stochastic policy* $\pi^{\approx} : S \times A \rightarrow \mathbb{R}$ that returns the *probability* with which an imperfectly rational agent selects an action. 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. Using the values obtained by value iteration the stochastic policy is calculated as follows:

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

Using the values obtained by value iteration the stochastic policy is calculated as follows:

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

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 Algorithm 2. The algorithm takes as input a belief state B, a stochastic policy $\pi^{\approx}$, 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, given the MDP transition function, 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 in a

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

Algorithm 2 Predicting future user paths.

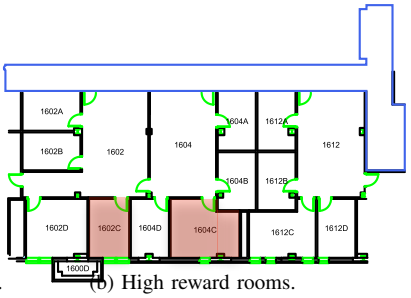
function PREDICTFUTUREPATHS($B, \pi^{\approx}, thr, maxD$)
  $n \leftarrow newTreeNode(\text{nil}, \text{nil}, 1, 0)$
  for all s such that $B(s) > thr$ **do**
    $n' \leftarrow newTreeNode(nil, s, B(s), 0)$
    CREATEPLANTREE($n', \pi^{\approx}, s, thr, 1, maxD, \emptyset$)
  end for
  return n
end function
procedure CREATEPLANTREE(n, $\pi^{\approx}$, s, thr, d, $maxD$, $visited$)
  if $d \leq maxD$ **then**
    for all $a \in A$ such that $\pi^{\approx}(s, a) \geq thr$ **do**
      $p \leftarrow \pi^{\approx}(s, a)$
      node $n' \leftarrow newTreeNode(a, s, p, d)$
      n.ADDCHILD(n')
      $v' \leftarrow visited \cup s$
      for all s' such that $(P_a(s, s') > 0) \wedge (s'.getValue > s.getValue) \wedge (s' \notin visited)$ **do**
        CREATEPLANTREE($n', \pi^{\approx}, s', thr, d + 1, maxD, v'$)
      end for
    end for
  end if
end procedure

Algorithm 3 Predicted user's destination.

function COMPUTEDESTINATION($PlanTree$, $V*$)
  $maxR \leftarrow 0$
  for all path $P = [n_0, ..., n_m] \in PlanTree$ **do**
    $r \leftarrow n_0.p * \sum_{n_1}^{n_m} V^*(n.s)$
    if $r > maxR$ **then**
      $maxR \leftarrow r$
      $bestPath \leftarrow P$
    end if
  end for
  return $n_m \in bestPath$
end function

...	
R1600	0.136531
...	
R1602	0.332103
R1602A	0.00369
R1602B	0.03321
R1602C	0.05166
R1602D	0.03321
...	
R1604	0.217712

(a) Example belief state.



(b) High reward rooms.

Figure 6: Example inputs.

way that assumes 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 3, consists of selecting the non-cyclic path with the largest long term reward. The first level of the tree generated from Algorithm 2 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 (see Section II-B). 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 state stored in each node at level 1 in the tree.Once the highest reward path is selected, its the leaf node 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 6b have a reward of 10, and all 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 states/rooms as highlighted in Figure 6a. 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, considering all actions that a user might choose to take him/her to other rooms results in in the tree of Figure 7. The high reward paths are highlighted in green in the figure.

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 6a, 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.

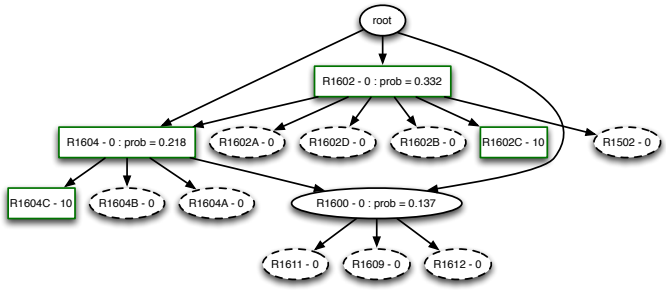


Figure 7: Predicted plan tree, given belief state of Figure 6a.

Figure 8: Indoor Test Environment with Selected Destinations and User Routes

IV. Implementation and Experiments

A. Implementation on a mobile platform

We developed a stand-alone application implementing the outlined system for an Android-based smart-phone, the Google Nexus S. The primary issue encountered during development was load balancing between the the different system components. Consequently, the resource intensive components of the system were developed as background services integrated using the 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 to the mobile application, value iteration is run in the server, speeding up prediction.

B. Automated Map Translation

In order to model the indoor environment and extract a logical layout of the building, we utilize a vector graphic representation, the Scalable Vector Graphics (SVG) format [16], of the floor plans of the building. SVG is an XML-based format that can be processed by a multitude of standard tools in order to extract data from an image. Using the defined formats, an interconnected network of rooms is established based on their adjacency information. This extracted information is useful to both the path-planner and the destination prediction modules. As we have seen in Section III-C, an MDP problem is composed of a set of states, a set of actions, a transition function and a reward function. Thus, using the map representation, we generate the set of states for the MDP using the resulting set of rooms from the vector map representation. Moreover, the connections between the rooms is used to create the set of actions within the MDP, 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 $1 - \epsilon$.

C. Experimental Platform

We tested the system on a single floor of an indoor environment (see Figure 8) to analyze the feasibility of the implemented solution. Three sets of experiments involving navigation from a defined start to multiple destination locations were performed. The user held the smart-phone pointing forward and walked at a normal pace. Each set

of experiments was repeated multiple times for consistency, for a total of 20 runs. There were approximately 20 Wifi access points (Figure 6b) 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 (see Figure 4b), along with a laser map of the environment. In all experiments the user starts from a pre-defined starting location and has two potential destination locations to head towards. In Experiment 1, the user heads towards destination 1 (room 1602 in Figure 8), while in experiment 2, the user heads towards destination 2 (room 1513). Finally, in experiment 3, the user starts heading towards destination 2, but changes intent and heads to destination 1 during the course of the experiment. Once the prediction module identifies the user destination intent, it is communicated to the path-planning module. The routes and destinations are illustrated in Figure 8.

Table I: Map loading and path-planning times

With localization and prediction	Process Time (s)	Load Time (s)	Grid Planning (ms)	Grid Re-planning (ms)	High-Level Planning (ms)
Yes	46.90	3.58	749	22.8	20
No	46.90	3.44	408	23	20

Table I 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. After the initial processing, future load times and subsequent re-planning drop dramatically, as once a connected graph is built, we make the assumption that the base topography (as indicated by the floor plan) remains the same. Consequently, instead of processing the map in its entirety we load only the connectivity graph, thereby reducing overall load time. Finally, the increase in computational times when other system modules are running is a clear indication of the constraints of the mobile platform. Despite the added complexity, our re-planning algorithm is able to significantly reduce re-planning time, highlighting the effectiveness of our map-representation and path-planning.

We tabulate the results from our experiments in table II. The system is not initialized with a starting location and attempts to converge based on the measured signal strength. From the table we can see that for 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

Table II: Experimental Results

Exp.	Path-length (m)	Start pose ϵ (m)	Final pose ϵ (m)	Mean ϵ of most prob. loc. (m)	Time to predict (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

user travels a total of $32m$ to destination. Moreover, we can see that 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 $5m$. It is interesting to observe that the estimate of final position is of by a value greater than the calculated mean error. We believe this is due to the fact that 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. As a result, the system is not able to converge to the final location with a higher degree of certainty.

During the course of experimentation, we encountered runs with significant localization drift due to magnetic interference in the environment as well as outdated RSSI database. In order to focus on analyzing the architecture, data from runs suffering from significant magnetic distortion was disregarded. Finally, looking at the prediction component of the system we can see that system accurately estimates destination for each of the runs. 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, averaged over the different runs 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 hypothesize that multiple predictions in experiments 1 and 3 occur whenever there is 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.

V. Conclusion

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. This architecture has been adapted for deployment within the computational restrictions of smart-phones and subsequently tested for effectiveness via a series of experiments involving human users. Experimental results have shown the effectiveness of the predictive navigation in minimizing the need for direct user input in the device. To our knowledge, this is the first architecture that atempts 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. Regarding the path prediction component, we intend to use the remote server to perform more complex (and thus computationally more expensive), user prediction based on the work of [7; 8].

Acknowledgment

This work is sponsored in part by the Google Core AI gift from Google Inc. The content of this paper does not necessarily reflect the position or policy of the sponsors and no official endorsement should be inferred. The authors would like to thank members of the rCommerce Laboratory at Carnegie Mellon University for their valuable contribution during development and testing. We would like to thank the RI Summer Scholars program for making the author's collaboration possible.

References

[1] S. Koenig and M. Likhachev, "D*lite," in *AAAI/IAAI*, 2002, pp. 476–483.

[2] V. Otsason, A. Varshavsky, A. LaMarca, and E. de Lara, "Accurate gsm indoor localization," in *UbiComp 2005: Ubiquitous Computing*, ser. LNCS, vol. 3660, 2005, pp. 903–903.

[3] X. Luo, W. J. OBrien, and C. L. Julien, "Comparative evaluation of received signal-strength index (rssi) based indoor localization techniques for construction jobsites," vol. 25, no. 2, 2011, pp. 355 – 363.

[4] A. Bernardos, J. Casar, and P. Tarrio, "Real time calibration for rss indoor positioning systems," in *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, sept. 2010, pp. 1 –7.

[5] Y. Jin, H.-S. Toh, W.-S. Soh, and W.-C. Wong, "A robust dead-reckoning pedestrian tracking system with low cost sensors," in *Pervasive Computing and Communications (PerCom), 2011 IEEE International Conference on*, march 2011.

[6] M. G. Armentano and A. Amandi, "Plan recognition for interface agents," *Artificial Intelligence Review*, vol. 28, no. 2, pp. 131–162, 2007.

[7] J. Oh, F. Meneguzzi, and K. Sycara, "Antipa: an agent architecture for intelligent information assistance," in *Proceedings of the Nineteenth European Conference on Artificial Intelligence*, 2010, p. (to appear).

[8] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd National Conference on Artificial Intelligence*. AAAI Press, 2008, pp. 1433–1438.

[9] D. Cagigas, "Hierarchical algorithm with materialization of costs for robot path planning," *Robotics and Autonomous Systems*, vol. 52, no. 2-3, pp. 190 – 208, 2005.

[10] J. Hoffmann, "Towards efficient belief update for planning-based web service composition," in *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, Amsterdam, The Netherlands, 2008, pp. 558–562.

[11] J.-A. Fernández-Madrigal and J. González, "Multihierarchical graph search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 103–113, January 2002.

[12] G. Korsah, A. Stentz, and M. Dias, "Dd* lite: Efficient incremental search with state dominance," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-12, May 2007.

[13] N. Kothari, B. Kannan, and M. Dias, "Robust indoor localization on a commercial smart-phone," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-11-27, August 2011.

[14] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.

[15] R. E. Bellman, *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[16] W3C, World Wide Web Consortium, "Scalable Vector Graphics (SVG) 1.2," W3C Working Draft, May 2004, extracted from http://www.w3.org/TR/SVG12/.