Leading the Way: An Efficient Multi-robot Guidance System

Piyush Khandelwal
Dept. of Computer Science
University of Texas at Austin
Austin, TX 78712, USA
piyushk@cs.utexas.edu

Samuel Barrett*
Kiva Systems
North Reading,
MA 01864, USA
basamuel@kivasystems.com

Peter Stone
Dept. of Computer Science
University of Texas at Austin
Austin, TX 78712, USA
pstone@cs.utexas.edu

ABSTRACT

Recent advances in service robotics have made it possible to deploy a large number of mobile robots in indoor environments to perform tasks such as delivery, maintenance and eldercare. If a centrally connected multi-robot system is available, can it be effectively used to aid humans in other on-demand tasks? In this paper, we demonstrate how individual service robots in a multi-robot system can be temporarily reassigned from their original task to help guide a human from one location to another in the environment. We formulate this multi-robot treatment of the human guidance problem as a Markov Decision Process (MDP). Solving the MDP produces a policy to efficiently guide the human, but the state space size makes it infeasible to optimally solve it. Instead, we use the Upper Confidence bound for Trees (UCT) planner to obtain an approximate solution. We show that this solution outperforms an approach that uses a single robot to guide the human from start to finish.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics/Commercial Robots and Applications*

Keywords

Multi-Robot Planning; Human Guidance

1. INTRODUCTION

With recent advances in service robotics, it is becoming increasingly plausible to deploy a large number of mobile robots in indoor settings such as warehouses and hotels [6, 11]. The ubiquitous presence of mobile robots will present interesting challenges for their effective use to aid humans. In this paper, we study the problem of how robots working on ongoing service tasks can be temporarily reassigned to efficiently guide a human to his destination.

We introduce and specify this multi-robot human guidance problem as follows. A human interrupts a robot and uses it as an interface to ask a centralized system for navigation assistance. Given a map of the environment, the location and destination of the human, and the location, destination, and planned path for all the robots in the environment, the system plans how this set of robots can most efficiently assist the human. The system can move robots around in the environment, and use a robot located in the human's vicinity to give directions to the human or lead him to his destination. The objectives of such a system are to help the human reach his goal quickly while minimally disrupting the robots' ongoing tasks. Due to the sequential and stochastic nature of the human's and robots' movement inside the environment, we formally model this human guidance problem as a Markov Decision Process (MDP).

Prior approaches to human-guidance have typically used a single dedicated robot to guide people from start to finish [13, 18], an idea which we term the *single robot approach* to indoor navigation assistance. A single robot may move slower than the people it guides, especially in large or crowded spaces. In these situations, a multi-robot solution may provide a superior solution to human guidance, by allowing the person to move at his own natural speed. Rather than requiring a robot to stay with the person, robots can be proactively sent to key locations in the environment where the person is likely to need help next to get to the goal, and the human is handed off from one robot to the next. In this paper, we will test the hypothesis that a multi-robot solution should always be able to provide an equivalent or quicker solution for navigating the human to his goal when compared to the single robot approach.

In addition, robots in this system are not dedicated to the task of guiding people inside the building, but can be multi-purpose. Our algorithm incorporates each robot's utility of performing its service task to that of getting the human to the goal, and attempts to use robots in a manner such that there is only a minimal overall system utility loss. We will also test the hypothesis that a multi-robot solution to the human guidance problem can reduce the overall loss in utility when compared to the single robot approach.

The three main contributions of this work are as follows. First, we define the multi-robot human guidance problem as an MDP in Section 3. Second, we propose using the Upper Confidence bound in Trees (UCT) planning algorithm to approximately solve this MDP in Section 4. Finally, we demonstrate the efficacy of this approach through experiments that compare it to the single robot approach in Section 5. Before discussing these contributions, we briefly touch upon related work in the following section.

2. RELATED WORK

From a high-level perspective, two independent research components exist in the human guidance problem: (i) human-robot interaction to understand how robots can effectively communicate directions to people (and how people respond to those directions), and (ii) planning and execution to decide along which path to guide the person and, in our case, using which robot. In this paper, we will primarily focus on the planning and execution component of the human guidance problem using multiple robots.

*Samuel performed this work while a student at the Univ. of Texas.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

With regards to the human-robot interaction component, past research has investigated how humans interpret natural language navigation instructions [3] and how natural language instructions can be interpreted by robots in order to navigate to a goal [17]. In this research, we are interested in the complementary problem of how a robot can give directions to humans in indoor settings. One line of previous research has focused on providing navigation instructions using stationary booths and personal handheld devices [1, 2]. Another application other than navigation assistance used ambient displays to influence human routes through a building in order to encourage building inhabitants to make healthier choices such as taking the stairs rather than the elevator [14]. In order to focus on the planning and execution components of the guidance task, we assume that research such as that cited above enables robots to have an effective means of communicating directions to people. Specifically, in our experiments we have the robots display arrows indicating the direction the person should walk, but our approach could use other interfaces such as voice commands.

With regards to the planning and execution component, in preliminary work we explored the initial problem of selecting locations in the environment where robots should be placed, under the simplifying assumption that they can move *instantaneously* to the assigned location [8]. While this assumption simplifies the problem by making the robots' current locations irrelevant, it also renders the solution inapplicable to the real world. This paper therefore introduces a much more complete treatment of the multi-robot guidance problem with a novel MDP formulation which includes a representation of each robot's motion and travel time.

Using a robot to guide a human has been explored over the last two decades, for applications such as leading tours [18] and providing navigation assistance to the elderly or visually impaired [10, 12]. However, to the best of our knowledge, the robot has always been dedicated to this purpose. In contrast, we assume there is a system of multiple robots that are otherwise executing background tasks. We now follow with a formal description of the problem.

3. PROBLEM DESCRIPTION

A multi-robot human guidance task begins when a human approaches a robot and requests assistance reaching a destination. In this research, we do not address how to best communicate this request; it could be for instance via natural language, through a smartphone interface, or typing on a laptop on the robot. We assume that this robot and several others are otherwise engaged in background service tasks, which they will interrupt to help the person. We also assume that there is a centralized system with some representation of the environment, the locations of all robots in the environment, and information about each robots' current service task. The system's goal is to guide the person to the destination efficiently, while limiting the robots' time away from their background tasks.

We formally describe such a multi-robot guidance task as a MDP $M = \langle S, A, P, R \rangle$ where S represents the environment's state space, A_s is the set of actions the system can take at state $s \in S$, P is the transition function that gives the transition probability of reaching a particular next state from a given state-action pair ($P : S \times A \times S \to \mathbb{R}$ such that $\sum_{s'} P(s,a,s') = 1$), and R is the reward received given a transition ($R : S \times A \times S \to \mathbb{R}$). A task inside this MDP is episodic with a designated start state in which the person requests assistance, and termination state in which the person has arrived at his destination. A task solution is represented as a policy $\pi(s) : S \to A_s$ that provides the action the system should take at any state.

In the remainder of this section, we introduce the environmental representation and notation used to define this MDP, and then specify each of its four components in detail.

3.1 Representation & Notation

In principle, the environment can be thought of as a continuous, bounded, multi-planar space, with the humans and robots having positions and orientations that can be represented as (x, y, θ) poses. However, for the purpose of decision-making, we reason about the environment as a topological graph, which provides a compact, discrete representation while still retaining information about all key locations and the connectivity between those locations. We automatically convert a discretized grid representation of the environment into a topological graph as illustrated in Fig. 1a, and documented in prior work [8].

A topological graph $g = \langle N_g, E_g \rangle$ consists of nodes N_g located at each intersection or junction and straight-line edges E_g between neighboring junctions. Given g, the location of a robot or person in the environment can be projected to the closest graph edge e_{uv} from node u to node v to produce a tuple $l_{uvp} = \langle u, v, p \rangle$ where $p \in [0,1]$ represents how far along edge e_{uv} the projection lies. If $p = 0$, then the object projects exactly at node u, and if $p = 1$ then the object projects exactly at node v.

Given this environmental representation, we introduce the following notation and then define the four components of the MDP.

- $\bar{v}_h$ and $\bar{v}_r$ are the estimated average speed of the the human and robot along graph edges, respectively.
- *euclidDist(u,v)* is the euclidean distance between graph nodes u and v in the environment's topological representation.
- *pathDist(u,v)* is the shortest path distance between nodes u and v computed using Dijkstra's algorithm.
- *adjacentNodes(u)* is the set of all nodes adjacent to node u.
- *nodeAngle(u,v)* is the compass direction of the edge from u to v. Formally, $nodeAngle(u,v) = \arctan((v.y - u.y)/(v.x - u.x))$, where $\langle u.x, u.y \rangle$ and $\langle v.x, v.y \rangle$ are the euclidean coordinates of nodes u and v, respectively.

3.2 State Representation

The MDP state is defined so as to represent the factors that affect decision-making, namely the human's location and destination, the robots' current locations, and the service task each robot is performing. In addition, the MDP state also represents the information about whether a specific robot has been diverted to help the human at a given location, and if any robot has assisted the human by leading or directing him to his destination.

Specifically, should R robots be present in the environment, we define a factored representation with the following $8R+4$ factors:

$$\langle loc, loc_{prev}, robot_i.\langle l_u, l_v, l_p, \tau_d, \tau_u, \tau_t, \tau_T, h \rangle, assist.type, assist.loc \rangle,$$

where $robot_i$ represents information about a particular robot, and $i \in \{1, \ldots, R\}$. Factors can be continuous (labeled C) or discrete (labeled D) and are explained in more detail as follows:

- $loc \in N_g$ (D) is the closest graph node to the human's location. N_g is the set of graph nodes in the environment.
- $loc_{prev} \in N_g$ (D) is the previous value contained in loc. At the start of a task, it is initialized to the same value as loc.
- For every robot in the environment, the following information is maintained in the state:
  - l_u, l_v and l_p represents the robot's precise location in the topological representation, as explained in Section. 3.1. l_u (D), l_v (D), and l_p (C) can take the following values:

$$l_u \in N_g, l_v \in N_g, \text{ and } l_p \in [0,1].$$

- $\tau_d \in N_g$ (D) is the location of the service task τ that the robot needs to perform.
- $\tau_u \geq 0$ (C) is the utility of performing service task τ.
- τ_T (C) is the total time the robot needs to spend at τ_d while performing service task τ.
- τ_t (C) is the time the robot has already spent at τ_d performing task τ. It can take any value less than τ_T, and is 0 before the robot arrives at τ_d.
- h (D) represents if the robot has been diverted to help the human, and can take the following values:

$$h \in \{\text{NONE}\} \cup N_g,$$

where NONE represents that the robot has not been assigned to help the human, and $h \in N_g$ represents that the robot should move to h to assist the human.

- *assist.type* (D) represents whether any assistance was provided to the human at his current location *loc* by a colocated robot. It can take values in:

$$assist.type \in \{\text{NONE}, \text{LEAD}, \text{DIRECT}\}$$

- *assist.loc* (D) is a location adjacent to the human's current location *loc* that the system has advised the human to move towards using a colocated robot, and can take values in:

$$\begin{cases} \text{NONE} & assist.type = \text{NONE} \\ v : v \in adjacentNodes(loc) & assist.type \neq \text{NONE} \end{cases}$$

The human's location *loc* is his destination in a terminal state.

3.3 Action Representation

Actions are taken whenever the human completes a transition to a graph node, effectively converting the MDP into a *Semi-Markov Decision Process* [7]. Whenever a human completes a transition to a graph node, the system can execute five types of discrete actions.

- *AssignRobot(r,v)* ($r \in R, v \in N_g$) – Deviate robot r from its current service task and start navigation to v to assist the human in the future. Here R is the set of all robots and N_g is the set of all nodes in the environment.
- *ReleaseRobot(r)* ($r \in R$) – Release robot r from assisting the human, and allow it to return to its service task.
- *DirectHuman(r,v)* – Display a directional arrow on the display interface of a robot r colocated with the human to direct the human towards node v. This action can only be taken if robot r is at the exact same location *loc* as the human:

$$isRobotExactlyAt(r, loc) \iff ((r.l_u = loc \textbf{ and } r.l_p = 0) \textbf{ or } (r.l_v = loc \textbf{ and } r.l_p = 1)),$$

Furthermore, the system can only direct the human to a node adjacent to his current location, i.e. $v \in adjacentNodes(loc)$.

- *LeadHuman(r,v)* – Use robot r colocated with the human to start leading him to an adjacent location v.
- *Wait* – Do nothing and wait for the human to move (or be led) to a location adjacent to his current location.

Wait is the only action that causes the passage of time, and is also the only action with a non-deterministic outcome (explained in Section 3.4). All other actions are decisions made by the system, are deterministic, and do not cause the passage of time. For instance, the system needs to follow *LeadHuman* (i.e. a decision to lead the human) with *Wait* so that time can pass and the system can actually lead the human to the intended destination. Figure 1 illustrates the execution of a simple policy from start to terminal state in a small environment with two robots. The full state representation of the system for each state in Figure 1 is given in Table 1. Furthermore, Table 2 lists the preconditions that need to be met to execute each action, as well as the outcome of all deterministic actions.

3.4 Transition Function

There are three non-deterministic events that happen whenever the *Wait* action is taken:

1. The human decides to move from his current location to an adjacent location in the environment. For instance, if the human is being led by a robot, he will likely follow the robot. If a colocated robot gave directions to the human, then the human will try his best to follow them. Finally, if no assistance was provided to the human, then he will make a decision based on past assistance from the system and search for his goal. The non-determinism behind a human's decision making process is captured by a *Human Decision Model*, and this model needs to account for decisions that real people make in the presence of robots providing navigation assistance.
2. A robot may complete his current service task after spending τ_T time at task location τ_d. Once it completes this task, a new task is assigned to it. A *Task Generation Model* captures the distribution of new tasks that can be assigned to the robot.
3. The robots and human will move around in the environment, and this motion is described by the *Object Motion Model*. This model accounts for environmental factors such as:
   - What path does a robot choose to go to its destination?
   - How do robots and humans avoid obstacles?
   - At what speed do robots and humans navigate different regions in the environment?
   - How much time passes while the human completes the transition to an adjacent node?

In a model-based reinforcement learning setting [16], these models can be learned by the system using real world experience. Once models are available to the system, it can can solve the MDP using optimal or approximate MDP solvers.

3.5 Reward

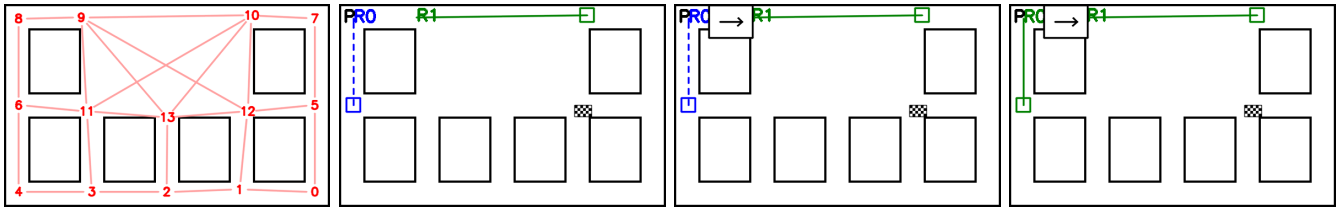
The reward function comprises the following two desiderata for the multi-robot human guidance problem. The solution should minimize (i) the *time taken* by the human to reach the goal as well as (ii) the *utility loss* suffered by the system caused by the time required to temporarily reallocate robots to guide the human. In this section, we construct a reward function that addresses these goals.

Let the amount of time that passes when the human transitions from s to s' be Δt. The utility loss for a robot r deviated from its service task τ with utility τ_u is dependent on the amount of time the robot is delayed in reaching task location τ_d:

$$U^r_{ss'} = \tau_u(timeToDest(r_{s'}, r.\tau_d) + \Delta t - timeToDest(r_s, r.\tau_d)),$$

where $timeToDest(r_s, d)$ is the shortest time robot r needs to reach d from system state s in which robot r is located at $\langle l_u, l_v, l_p \rangle$:

$$timeToDest(r_s, d) = \min(r.l_p \times (euclidDist(r.l_u, r.l_v)) + pathDist(r.l_u, d), (1 - r.l_p) \times (euclidDist(r.l_u, r.l_v)) + pathDist(r.l_v, d)).$$



(a) Topological representation (b) s_1: start state (c) s_2: after *DirectHuman*(0,9) (d) s_3: after *ReleaseRobot*(0)

(e) s_4: after *AssignRobot*(1,9) (f) s_5: after *Wait* (g) s_6: after *LeadHuman*(1,12) (h) s_7: terminal state after *Wait*

Figure 1: The human's location is marked by a black P, and robots as marked as R0 and R1. Robots are colored green when performing a service task and blue when assisting the human (the dashed blue line indicates the shortest path to return to the original service task). In this figure, given the system state depicted in (b), the system uses the robot at node 8 to direct the human to node 9, and assigns the robot at node 9 to eventually lead the human to his goal at node 12.

			$robot_0$								$robot_1$								$assist$	
State	loc	loc_{prev}	l_u	l_v	l_p	τ_d	τ_t	τ_T	τ_u	h	l_u	l_v	l_p	τ_d	τ_t	τ_T	τ_u	h	$type$	loc
s_1	8	8	8	8	0	6	0	5	1	8	9	0	0	10	0	5	1	NONE	NONE	NONE
s_2	8	8	8	8	0	6	0	5	1	8	9	0	0	10	0	5	1	NONE	DIRECT	9
s_3	8	8	8	8	0	6	0	5	1	NONE	9	0	0	10	0	5	1	NONE	DIRECT	9
s_4	8	8	8	8	0	6	0	5	1	NONE	9	0	0	10	0	5	1	9	DIRECT	9
s_5	9	8	8	6	0.36	6	0	5	1	NONE	9	0	0	10	0	5	1	9	NONE	NONE
s_6	9	8	8	6	0.36	6	0	5	1	NONE	9	0	0	10	0	5	1	12	LEAD	12
s_7	12	9	8	9	0.35	9	0	5	1	NONE	9	12	1	10	0	5	1	12	NONE	NONE

Table 1: Complete state representation for all states in Figure 1.

The time loss incurred by the human and the robots' utility loss can be combined to form the final reward function for this MDP:

$$R_{ss'} = -h_u \Delta t - \sum_r U^r_{ss'}$$

where h_u is the utility of helping the human.

Note that when $\tau_u = 0$, the robot is effectively idle as assigning the robot to help the human incurs no utility loss. On the other hand, a service task with $\tau_u >> h_u$ would lead the system to incur an extremely high negative reward if preempted. Also, for any other action other than *Wait*, no time passes and the robots do not move around in the environment and incur no utility loss. Consequently, for actions other than *Wait*, $R_{ss'}$ is always 0.

With the description of the MDP now complete, we next describe how UCT can be used to approximately solve this MDP.

4. UCT-BASED SOLUTION

UCT is an anytime algorithm that continues improving its policy estimation during an episode by performing extra planning in between decisions, making it well suited as a real-time solver for the human guidance task as the policy can be updated whenever the system is waiting for the human to move in the environment.

4.1 Background - UCT

While planning in an MDP using Value Iteration [16] provably converges to the optimal policy, it can be too computationally expensive to calculate in scenarios where the state-action space is too large, as is the case with the multi-robot human guidance MDP. Therefore, it can be advantageous to use planners that restrict search to a specific region of the state-action space. Specifically, in this paper we will use the Upper Confidence bounds for Trees (UCT) [9] planning algorithm. UCT has been shown to perform well on large domains with a large numbers of actions, such as Go [4] and large Partially Observable MDPs (POMDPs) [15].

Planning is performed by simulating a number of state-action trajectories from the current MDP state, i.e. Monte Carlo rollouts. For each state-action pair, UCT stores the number of visits for that pair as well as the long term expected reward of choosing that action at that state in a tree structure. During the planning phase, when the state has been previously visited, the action with the highest upper confidence bound on the estimated value is selected, and actions are chosen randomly otherwise. This approach balances exploring different actions with improving the estimate of promising actions.

To improve performance, this paper uses a modified version of UCT [5]. The original UCT formulation stores the path to a given state (i.e. the actions taken to reach that state) in its representation. However, our formulation drops this extra information to merge more states, which may relax the tree into a graph. In addition, when updating the estimated value of a state-action, the original UCT formulation only uses the reward accumulated from the Monte Carlo rollout. In contrast, we use the eligibility trace update used in Q learning, which combines the Monte Carlo estimate with the current expected value of future states. Let $Q(s,a)$ be the estimated value of taking action a from state s, n_{sa} be the number

Action	Preconditions	Next State $s' = s$ except:
AssignRobot(r,v)	$s.assist.type \neq$ LEAD **and** $s.robot_r.h =$ NONE	$s'.robot_r.h = u$
ReleaseRobot(r)	$s.assist.type \neq$ LEAD **and** $s.robot_r.h \neq$ NONE	$s'.robot_r.h =$ NONE
DirectHuman(r,v)	$s.assist.type \neq$ LEAD **and** *isRobotExactlyAt*$(s.robot_r, s.loc)$ **and** $s.robot_r.h = s.loc$	$s'.assist.\langle type, loc\rangle = \langle$DIRECT$, v\rangle$
LeadHuman(r,v)	$s.assist.type \neq$ LEAD **and** *isRobotExactlyAt*$(s.robot_r, s.loc)$ **and** $s.robot_r.h = s.loc$	$s'.assist.\langle type, loc\rangle = \langle$LEAD$, v\rangle$ **and** $s'.robot_r.h = v$
Wait		$s'.assist.\langle type, loc\rangle = \langle$NONE,NONE$\rangle$ **and** *other non-deterministic changes (Section 3.4).*

Table 2: Preconditions and transition functions for each action in the MDP.

of state-action visits, s' be the resulting state, and $0 \leq \gamma \leq 1$ be the discount factor of future rewards r_i. The new value is given by:

$$\delta_{t+1} = r_0 + \gamma \left[\lambda \left(\sum_{i=1} \gamma^{i-1} r_i \right) + (1-\lambda) \max_{a'} Q_t(s', a') \right] - Q_t(s,a)$$

$$Q_{t+1}(s,a) = Q_t(s,a) + \frac{1}{n_{sa}+1} \delta_{t+1}$$

where λ balances Monte Carlo versus Temporal Difference (TD) updates. $\max_{a'} Q_t(s', a')$ is the maximum value over the experienced actions from the next state including actions experienced in the current rollout. Intuitively, when λ is close to 1, values received from taking exploratory actions at future states strongly influence the estimated value of a state-action pair. Using values close to 0 limits the effect of these exploratory actions. Finally, UCT estimates the upper confidence bound using $Q(s,a) + C_p \sqrt{(\ln n_s)/n_{sa}}$, where n_s is the number of visits of the state and $C_p = \sqrt{2}$ when the MDP rewards are scaled between 0 and 1. C_p can also be tuned empirically to better balance exploration versus exploitation when the number of rollouts is limited due to computational constraints.

In addition, UCT convergence can be further improved if a reasonable yet suboptimal *default policy* is available. Whenever UCT encounters a previously unseen state, it executes this default policy instead of a randomly selected action. Using a default policy allows UCT to converge to better actions faster higher up in the tree.

4.2 Transition Function

Before we can use UCT to solve the multi-robot human guidance MDP, we need to provide UCT with a generative transition model of the environment for drawing samples while planning. This model needs to define distributions for all non-deterministic events described in Section 3.4. While this model can be learned from the environment, we leave learning to future work. Instead, we supply a hand-coded model to UCT. As long as the model does not deviate significantly from its real world counterpart, the policy derived from planning should be successful.

4.2.1 Human Decision Model

The human decision model needs to account for a human's decision to move to an adjacent node given his current location and the assistance that the system has provided to him. We categorize this model by the different types of assistance provided to the human at his current location:

- $s.assist.type =$ NONE: If no assistance is provided to the human at his current location, we assume that the human will continue moving in the same direction as he has been moving in. To compute the distribution of his next location, we first compute an expected direction of motion $expDir \in [0, 2\pi)$:

  $$expDir = nodeAngle(s.loc_{prev}, s.loc).$$

  Using *expDir*, we can calculate the distribution of transition probabilities to adjacent nodes. The hand-coded model simulates the human's uncertainty about which adjacent node $v \in adjacentNodes(s.loc)$ to move to next as:

  $$edgeAngle = nodeAngle(loc, v)$$

  $$P(v) = \frac{1}{c} exp\left(-\frac{absAngleDiff(edgeAngle, expDir)^2}{2\sigma^2} \right) + d$$

  where c is a normalizing constant so that $\sum_v P(v) = 1$ and $\sum_v d = 0.01$. The value d is used to represent the inherent unpredictability of humans. $\sigma^2 = 0.1$ controls the spread of the Gaussian function and represents how sure a human is to move in the the expected direction of motion.

- $s.assist.type =$ DIRECT: If a robot colocated with the human gave directions, these directions will influence the human's decision. We calculate the expected direction of motion as:

  $$expDir = nodeAngle(s.loc, s.assist.loc).$$

  The probability of transitioning to adjacent nodes is calculated as before, except $\sigma^2 = 0.05$, indicating the human is more confident about moving in the expected direction of motion than the case where no assistance was provided.

- $s.assist.type =$ LEAD: If the human is being led by a robot an adjacent node, then there is no non-determinism in the human's next location, and $s'.loc = s.assist.loc$.

4.2.2 Robot Task Generation Model

When a robot completes a service task, it is assigned a new service task τ' with the following specifications:

- τ'_d: It is assumed that each robot has a home base ($r_{hb} \in N_g$), and it performs tasks close to this home base. We first generate a value k from a Poisson distribution with mean 1 that represents how far away from the robot's home base r_{hb} the next task will be located. τ'_d is selected randomly from $\{v : edgeDist(v, r_{hb}) = k\}$ where *edgeDist(u,v)* is the smallest number of graph edges needed to traverse from u to v.

- τ'_u: τ'_u is set to the problem-defined average task utility $\bar{\tau}_u$.

- τ'_T: τ'_T is set to the problem-defined average task time $\bar{\tau}_T$.

4.2.3 Object Motion Model

The object motion model used by the UCT solver is characterized by the following properties:

- Humans and robots move at constant speed along graph edges at $\bar{v}_h$ and $\bar{v}_r$, respectively.

- Collisions between different objects have no effect. Objects can pass each other while traversing the same graph edge.

- Each robot selects the path to its destination using the shortest path, computed using Dijkstra's algorithm.

- The time Δt required for the human to walk to the adjacent node is calculated as:

$$\Delta t = euclidDist(s.loc, s'.loc)/v_h$$

where v_h is the speed of the human during this transition. If the human is being led to $s'.loc$ by a robot, he can become limited by the speed of the robot, and $v_h = min(\bar{v}_h, \bar{v}_r)$. If the human is moving on his own accord, then $v_h = \bar{v}_h$.

4.3 Limiting action branching

Given a graphical representation with 50 nodes and 10 robots in the environment, the number of available actions can be greater than 500. While UCT is an approximate solver commonly used in domains with high action branching, this explosion in the state-action space makes it infeasible to solve a non-deterministic MDP in real-time. In this section, we detail some heuristics that are applied by the UCT solver to reduce the state-action size.

1. In the *AssignRobot(r,v)* action, instead of specifying which robot r is assigned to a location v, select the robot automatically using a heuristic. This heuristic effectively reduces the action to *AssignRobot(v)*, and attempts to minimize the utility loss incurred by the system for selecting a particular robot to service the assignment request. To select which robot r is sent to v in *AssignRobot(v)*, the solver first computes the minimum time that the human needs to reach location v as:

$$t_{min} = pathDist(s.loc, v)/\bar{v}_h$$

Next, the system calculates the set R' of all unassigned robots that can reach the assigned location before the human:

$$R' = \{r : \forall r \in R \mid r.h = \text{NONE} \textbf{ and } timeToDest(r_s, v) < t_{min}\}$$

If $|R'| > 0$ (a robot can reach l in time), the solver selects from R' the robot that has the minimal expected utility loss caused by waiting for the human at v before moving to its original service task destination τ_d:

$$\underset{r \in R'}{\text{argmin}}[t_{min} + pathDist(v, \tau_d)/\bar{v}_r - timeToDest(r_s, \tau_d)]$$

If $|R'| = 0$ (no robots can reach v in time), the solver selects the robot that can reach assigned location v first:

$$\underset{r \in R}{\text{argmin}}[timeToDest(r_s, v)]$$

2. Prespecify the maximum number of simultaneously assigned robots, since it is unlikely that the system will have to assign all robots in the environment to help the human at the same time. Given a user defined parameter *maxAssignedRobots*, the *AssignRobot* action is allowed only if:

$$|\{r : \forall r \mid s.robot_r.\tau_d \neq \text{NONE}\}| < maxAssignedRobots$$

3. Reduce the abstraction introduced by executing different permutations of the same *AssignRobot*, *ReleaseRobot*, *DirectHuman* and *LeadHuman* actions before *Wait* is executed, as these permutations results in the same final state in which *Wait* is executed. Consequently, while executing actions, the following priority order is imposed:

$$DirectHuman < ReleaseRobot < AssignRobot < LeadHuman$$

If an action with a higher priority has been executed, then no action of a lower priority can be executed unless the *Wait* action is executed. To enforce this constraint, the solver keeps track of the action taken at the previous state.

4. Do not assign multiple robots to the same location. If any robot r has been assigned to location v, i.e., $s.robot_r.h = v$, then *AssignRobot(v)* is not a valid action.

5. Automatically release the robot used to direct the human after the *DirectHuman* action is called.

6. Ensure that if a robot is colocated with the human, then it provides assistance to the human before the *Wait* action is called. This constraint is imposed by disallowing the *Wait* action unless $s.assist.type \neq$ NONE.

7. Do not assign a robot to a location from which a robot has just been released, unless the *Wait* action is executed. To enforce this constraint, the solver keeps track of all the nodes from where robots were released since the last *Wait* action.

Of all the heuristics outlined above, the first three have the greatest impact on reducing action branching.

5. EXPERIMENTS

The ultimate test of the solution proposed in Section 4 is whether it can be deployed on physical multi-robot service robots that are continually operational in a way that is useful to people. Since we do not have access to such a system at present, in this paper we validate the algorithm via simulation experiments. We begin with a simplified setting in Section 5.1 to validate that UCT effectively approximates the optimal solution when it is available. In Section 5.2 we then evaluate performance on the complete problem in a way that is meant to capture its future intended use in the real world.

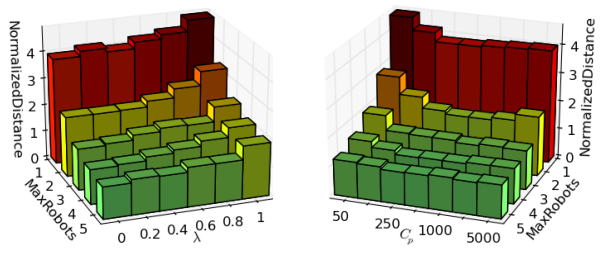
5.1 Instantaneous Robot Motion Domain

In preliminary work, we described a multi-robot human guidance domain with instantaneous robot motion (IRM) [8], which we briefly revisit here. In the IRM domain, robots are instantaneously moved to their destinations, and the utility of background service tasks is not considered. One way to visualize this domain is to imagine an assistance interface (display) installed at every key location in the environment that directs people to the next assistance interface or the goal. The only restriction this domain has is on the number of times directions can be given to the human, i.e. there is a limit on the total number of robots that can be assigned to help the human (*maxRobots*). As in the full version of the domain, there is non-determinism based on the human's stochastic decisions to move to a particular adjacent location. The objective in this domain is to minimize the distance traveled by the human.

Due to the relatively small size of the state-space, this domain can be solved optimally using offline Value Iteration, and then the policy only needs to be looked up as the human is moving around in the environment. In this experiment, we apply UCT to this domain; UCT planning is performed whenever the system is waiting for the human to move around in the environment. In this smaller domain, no action restriction is required (unlike in Section 4.3), and the only algorithmic parameters that need to be tuned are UCT's confidence bound parameter C_p and eligibility trace parameter λ.

Experiments in this section are performed on the topological environment depicted in Figure 3a. All results are averaged over 1,000 trials over randomly generated start and goal locations of the human. Since the shortest distance between the start and goal location is different in each trial, all results are normalized against the shortest distance between the human's start and goal locations for that trial before results are aggregated.

Fig. 2a shows the average normalized distance walked by the person to reach the goal while varying λ and *maxRobots* with $C_p =$



(a) Varying λ (b) Varying C_p

Figure 2: (a) and (b) show the normalized distance at different UCT parameters using a red(worse)-green(better) heatmap.

maxRobots	1	2	3	4	5
UCT	3.8	1.78	1.40	1.19	1.15
VI	**3.46**	**1.70**	**1.24**	1.14	1.12

Table 3: Average normalized distance traveled by the person from start to goal. The VI result is in bold when it is statistically better than the UCT using a two sample t-test with p = 0.95.

500. Irrespective of the value of total robot placements, best performance is achieved when λ is close to 0. Since UCT does not converge in the provided computation time, some exploratory actions may be taken every rollout. In this domain, sub-optimal actions can be extremely costly, such as sending the human in a direction away from the goal. When λ is close to 1, backing up the true value of the rollout up the tree leads to incorrect value estimation, degrading performance. In contrast, when $\lambda = 0$, exploratory actions further down the tree do not affect the value of a state-action pair, and the values are closer to their true values, improving performance.

Fig. 2b shows the performance at different confidence bounds when $\lambda = 0$. The results show that as expected, at low values of C_p, UCT fails to explore sufficiently and performs poorly. The results show the best performance with a confidence bound of $C_p = 500$. Higher values produce a slight degradation in performance due to over-exploration. Table 3 shows the performance of UCT($\lambda = 0$, $C_p = 500$) when compared to the Value Iteration baseline. When UCT is allowed to give directions at least four times, the performance of UCT is statistically indistinguishable from optimal.

These results show that the performance of UCT can be tuned to be comparable to the optimal policy in a smaller and similar domain. In the next section, we examine UCT's performance in the full multi-robot guidance domain described in Section 3, which is too complex for VI. We also use the best UCT parameters identified in this section throughout the rest of the paper.

5.2 Complete Multi-Robot Guidance Domain

In this section, we perform simulated evaluations of the UCT-based solution to the multi-robot guidance problem, and compare how it performs against the single robot approach of leading the person from start to goal. The goal of these evaluations is to determine the environmental conditions in which each solution performs better. In order to perform these evaluations, we need a complete description of the transition function of the environment. As described in Section 4.2, UCT planning uses static models to draw transition samples. We perform one set of experiments with these exact models, but also perform experiments with different models to evaluate planning with inaccurate models.

Similar to the previous set of experiments all results in this section are averaged across 1000 trials with randomly assigned start and goal locations for the human. Since the shortest distance between the start and goal location is different in each trial, all results are normalized against the time required by the human to walk along the shortest path from the start to the goal.

Across all the experiments, the following parameters were kept constant: (i) The utility for guiding the human h_u is 1.0. (ii) The average task time $\bar{\tau}_T$ is 5 seconds. (iii) The maximum number of simultaneously assigned robots *maxAssignedRobots* was set to 1. (iv) UCT parameters λ and C_p were set to 0 and 500, respectively.

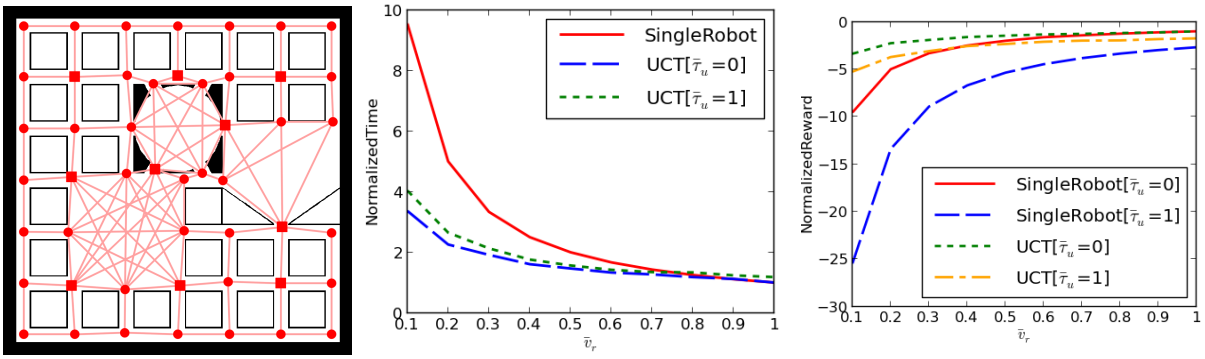
Since we cannot solve for the optimal solution to this MDP in a reasonable amount of time, evaluations have only been performed against the single robot approach to guidance. The single robot approach can be implemented using an MDP policy that uses the robot initially colocated with the human to lead the human all the way to the goal. This policy can be implemented by alternating the *LeadHuman* and the *Wait* action along the shortest path to the goal.

The UCT solution described in Section 4 is implemented as described, and planning is run whenever the *Wait* action is executed and the system is waiting for the human to move to the next goal. The single robot approach described above is used as the *default policy* for UCT whenever a robot is colocated with the human, i.e. when a robot is available to lead the human to the goal. In situations where a robot is not colocated with the human, instead of taking a random action at unseen states during planning, we implement the following heuristic as a default policy that attempts to send a robot to the person to provide assistance. The complete heuristic is described in Algorithm 1. The heuristic iteratively finds the best location in the human's likely path (lines 12-22) where an unassigned robot can reach in time, and sends a robot to that location (line 10). If a robot is attempting to help the human at a location different than the one computed by the heuristic, then it is released (line 6). In all other cases the system waits.

Algorithm 1 Default planning policy in states where no robot is colocated with the human.

```
1:  curNode ← s.loc, curDir ← nodeAngle(s.loc, s.loc_prev).
2:  while true do
3:      if |R'| > 0 in AssignRobot(curNode) then   ▷ (see Sec. 4.3)
4:          if r.h ≠ NONE for any r ∈ s.robots then
5:              if AssignRobot(curNode) will not assign r then
6:                  return ReleaseRobot(r)
7:              end if
8:              return Wait
9:          end if
10:         return AssignRobot(curNode)
11:     end if
12:     A ← adjacentNodes(curNode)
13:     diff ← Map(), dir ← Map()
14:     for all a ∈ A do
15:         dir[a] ← nodeAngle(curNode, a)
16:         diff[a] ← absAngleDiff(curDir, dir[a])
17:     end for
18:     nextNode ← argmin(diff)
19:     if diff[nextNode] > π/4 then
20:         break
21:     end if
22:     curDir ← dir[nextNode], curNode ← nextNode
23: end while
24: return Wait
```

In this set of experiments, we vary the average robot speed $\bar{v}_r$ and the average service task utility $\bar{\tau}_u$ while keeping the average



(a) Environment (b) Normalized Time (c) Normalized Reward

Figure 3: (a) The evaluation environment of size 46.8m × 46.8m. For the service task generation model presented in Section 4.2, the home bases $r_h b$ of the 10 robots are marked using squares. (b) and (c) show the normalized time and reward for different approaches in the multi-robot guidance MDP under varying average robot speed $\bar{v}_r$ and average service task utility $\bar{\tau}_u$.

human speed $\bar{v}_h$ fixed at 1m/s. The evaluation uses the exact transition model used by UCT planning, and the results are presented in Fig. 3. Fig. 3b shows the average normalized time required by the human to reach the destination, and Fig. 3c shows the average normalized system reward. The single robot which leads the person to the goal at the robot's speed requires $\bar{v}_h/\bar{v}_r$ normalized time to lead the person to the goal. In comparison, the multi-robot UCT approach gets the human to his goal faster than the single robot approach for $\bar{v}_r \leq 0.7$ when the average service task utility is 1 (same as the utility for guiding the human h_u), and is always faster or equivalent when the average service task utility is 0.

In terms of the system reward, UCT always performs better than or equivalent to the single robot irrespective of the value of $\bar{\tau}_u$, as depicted in Fig. 3c. It should be noted that as $\bar{\tau}_u$ approaches 0 and $\bar{v}_r$ approaches $\bar{v}_h$, UCT and the single robot approach perform equivalently as the single robot policy is optimal, and indeed UCT executes this policy. In other words, as the service task utility increases or robots are slower than humans, the UCT solutions acts as a seamless transition from a single robot to a multi-robot approach.

In Table 4, we tabulate the normalized time and reward at one set of environmental parameters ($\bar{v}_r = 0.5, \bar{\tau}_u = 1$), but differ the evaluation model from the one used in UCT planning. In the real world, any planning in this domain will always be done with a model that is not completely accurate, and in this set of experiments we verify whether the UCT solution is likely to hold up in the real world. During evaluation, we either change the motion model by changing the human's actual speed during evaluation or change the human decision model by multiplying m_{σ^2} to the variance in the human's decision (explained in Section 4.2). All planning is performed with $\bar{v}_h = 1.0$ and $m_{\sigma^2} = 1$. As Table 4 shows, UCT accrues more reward than the baseline despite planning with inaccurate models.

Params.	Method	Norm. Time	Norm. Reward
$\bar{v}_h = 1$	SingleRobot	2.00±0.00	-5.36±0.03
$m_{\sigma^2} = 1$	UCT	**1.57±0.03**	**-2.34±0.06**
$\bar{v}_h = 0.8$	SingleRobot	1.60±0.00	-4.29±0.03
$m_{\sigma^2} = 1$	UCT	**1.52±0.03**	**-2.25±0.06**
$\bar{v}_h = 0.5$	SingleRobot	**1.00**±0.00	-2.68±0.03
$m_{\sigma^2} = 1$	UCT	1.45±0.04	**-2.09±0.06**
$\bar{v}_h = 1$	SingleRobot	2.00±0.00	-5.36±0.03
$m_{\sigma^2} = 0.5$	UCT	**1.44±0.03**	**-2.13±0.06**
$\bar{v}_h = 1$	SingleRobot	2.00±0.00	-5.36±0.03
$m_{\sigma^2} = 2.0$	UCT	1.99±0.07	**-3.08±0.06**

Table 4: Normalized time and reward at $\bar{v}_r = 0.5$ and $\bar{\tau}_u = 1$. A statistically better result is in bold (using a t-test with p = 0.95). The first row of parameters corresponds to perfect planning.

6. DISCUSSION AND FUTURE WORK

The main contributions of this paper are formulating the problem of guiding a human unfamiliar with the environment using multiple robots as an MDP, providing one solution to this problem based on UCT, and evaluating this solution compared to a single robot approach. The key research challenges that needs to be answered to construct a system that performs multi-robot guidance in the real world are (i) an effective probabilistic planning algorithm suited to the task; (ii) a multi-robot platform capable of smooth navigation and communication; and (iii) constructing models of real humans in the environment via real human user evaluations. We have addressed the first challenge in this paper.

The formulation described in this paper is designed to be suitable for implementation in the real world, and our experiments show that the system performs reasonably well even in situations where it plans with inaccurate models. However, a number of real world challenges such as robot localization; noise in human perception; inter-robot communication errors; action execution failure; and space management between robots and humans need to be tackled to construct a real multi-robot system. As such, it is necessary to implement this guidance system on a multi-robot platform to validate the applicability of this work in the real world.

This work should be useful in other tasks where multiple service robots need to be interrupted from their service tasks to work on an on-demand task such as an unexpected delivery, as well as tasks where multiple people need to be serviced simultaneously. In addition, this paper focuses on a centralized approach and it may be necessary to look into distributed solutions for scalability. The framework in this paper for interrupting service robots for human-guidance should serve well as a basis for these future applications.

Acknowledgments

Tools developed by the ROS community have been used in this work. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287), ONR (21C184-01), AFOSR (FA8750-14-1-0070, FA9550-14-1-0087) and Yujin Robot.

REFERENCES

[1] J. Baus, A. Krüger, and W. Wahlster. A resource-adaptive mobile navigation system. In *International Conference on Intelligent User Interfaces*, 2002.

[2] A. Butz, J. Baus, A. Krüger, and M. Lohse. A hybrid indoor navigation system. In *International Conference on Intelligent User Interfaces*, 2001.

[3] D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Conference on Artificial Intelligence (AAAI)*, 2011.

[4] S. Gelly and Y. Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *Conference on Neural Information Processing Systems (NIPS)*, 2006.

[5] T. Hester and P. Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 2013.

[6] J. Hightower. Amazon is replacing people with drones and robots. http://flagpole.com/news/news-features/2014/09/24/amazon-is-replacing-people-with-drones-and-robots, 2014. Accessed: 2014-10-21.

[7] R. A. Howard. *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. Courier Dover Publications, 2013.

[8] P. Khandelwal and P. Stone. Multi-robot human guidance using topological graphs. In *AAAI Spring 2014 Symposium on Qualitative Representations for Robots*, March 2014.

[9] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, 2006.

[10] G. Lacey and K. M. Dawson-Howe. The application of robotics to a mobility aid for the elderly blind. *Robotics and Autonomous Systems*, 1998.

[11] J. Markoff. Beep, says the bellhop. http://www.nytimes.com/2014/08/12/technology/hotel-to-begin-testing-botlr-a-robotic-bellhop.html, 2014. Accessed: 2014-10-21.

[12] M. Montemerlo, J. Pineau, N. Roy, et al. Experiences with a mobile robotic guide for the elderly. In *Innovative Applications of Artificial Intelligence (IAAI)*, 2002.

[13] R. Philippsen and R. Siegwart. Smooth and efficient obstacle avoidance for a tour guide robot. In *International Conference on Robotics and Automation (ICRA)*, 2003.

[14] Y. Rogers, W. R. Hazlewood, P. Marshall, et al. Ambient influence: Can twinkly lights lure and abstract representations trigger behavioral change? In *International Conference on Ubiquitous computing (UBICOMP)*, 2010.

[15] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Conference on Neural Information Processing Systems (NIPS)*. 2010.

[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Cambridge University Press, 1998.

[17] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, et al. Understanding natural language commands for robotic navigation and mobile manipulation. In *Conference on Artificial Intelligence (AAAI)*, 2011.

[18] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, et al. MINERVA: A second-generation museum tour-guide robot. In *International Conference on Robotics and Automation (ICRA)*, 1999.