Hide & Seek

Exploring an Unknown World via POMDP

Josh Villbrandt, Divya Sharma, Prithvi Balaram

villbran@usc.edu, divyasha@usc.edu, balaram@usc.edu

Abstract

We explore the idea of making a robot play hide and seek, a two-player game in which one person hides and the other is charged with the task of finding the person in hiding. We focus on the "seeking" part of the game in our project. Specially, we are building a global navigation planner to efficiently guide a robot to a person of unknown location in an unknown world. The problem is modeled as a partially observable Markov decision problem (POMDP.) A person detection program is also created to support the planner. Mapping, localization, and local navigation programs already implemented in ROS are used.

Introduction

The Robotic Operating System (ROS) is used as the core backbone of this project. It provides many powerful tools to speed development including a method of automatically converting between coordinate systems, a system for handling messages between nodes, a simulated robot and world and a powerful 3D visualizer. ROS also includes programs such as a SLAM gmapping implementation and a local navigation planner, which our project builds on.

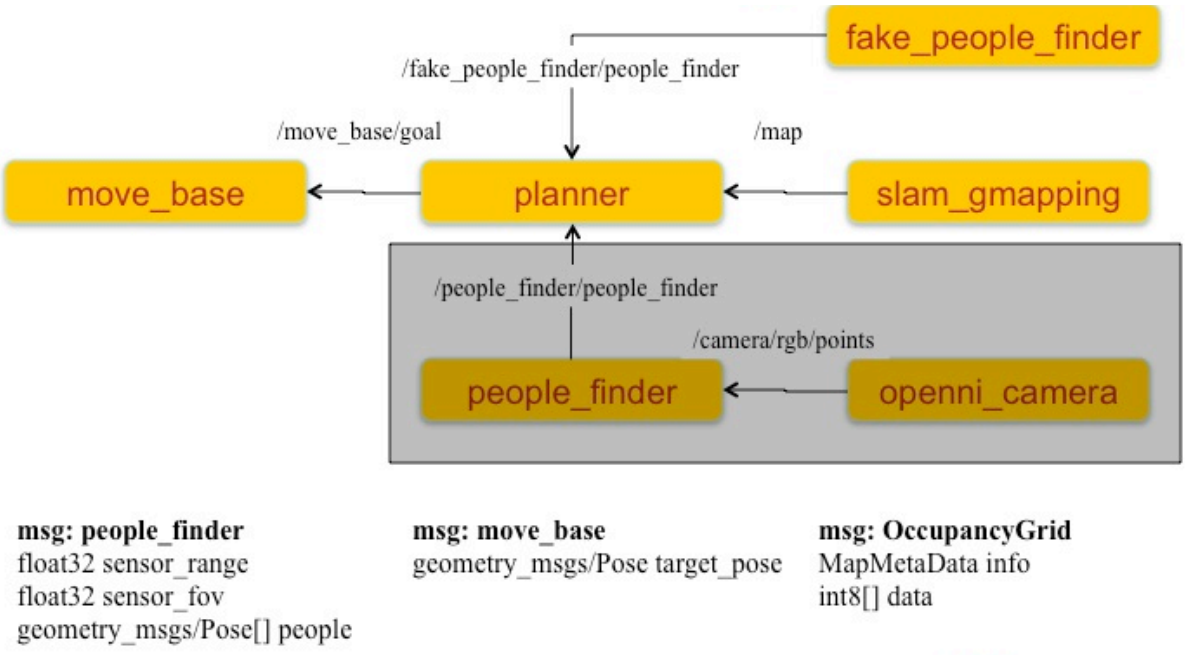

Figure 1: The interactions between relevant nodes and message contents. The greyed out area is how a real people finder would replace the fake people finder.

Our project incorporates two new programs – hide_n_seek/planner and hide_n_seek/fake_person_finder. The planner application chooses navigation goals based on the incoming map of the world and current position from slam_gmapping and the output of the fake_person_finder. The fake_person_finder (so called because it is intended for use only in simulated worlds) continuously monitors the view in front of the robot and reports any people that can be seen. Figure 1 is an interaction diagram, which shows the messages that flow between nodes.

Simulated World and Navigation

The Gazebo physics simulator was used to play out the program in a simulated environment. The custom map Gazebo created for this project is shown as Figure 2. The slam_gmapping node builds up a map of this environment as the robot moves around. The planner uses this map when deciding where to go and sends move_base messages. The move base application implements a local planner that takes into account the robot's physical existence and avoids obstacles when possible.

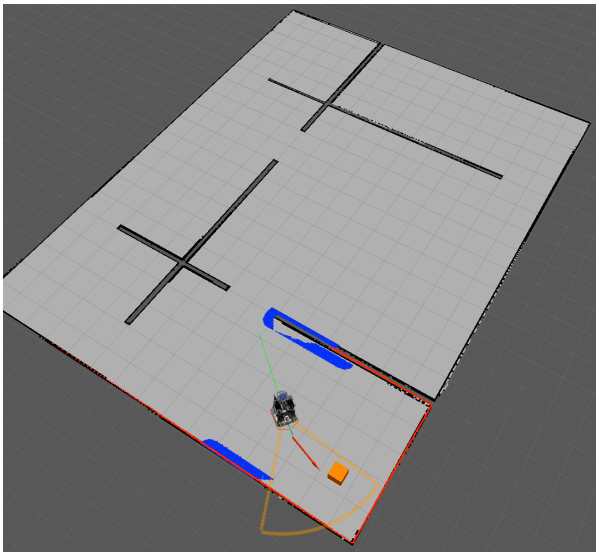

Figure 2: Custom map made for Hide n Seek simulation.

People Finder

To enable the robot to find people in a simulated world, a fake sensor was created as fake_people_finder. This program works with the interactive markers introduced in ROS Electric. The user can drag the interactive marker (orange box) around in Rviz, and then the fake_people_finder recognizes the new position as the location of a person. This is then reported as a person whenever the robot is in range. In an effort to simulate how real-world sensing would work with a Kinect, the field of view of this fake sensor has been chosen to match that of the connect. The interactive marker and sensor field of view (orange semi-circular outline) can be seen in Figure 3.

In addition to mimicking a Kinect's field of view, the fake sensor also incorporates noise. Currently, false positives are reported 5% of the time. The sensor also fails to report a person 10% of the time when one actually exists. This program was fully written with the intent of being replaced with a node actually running off of Kinect data.

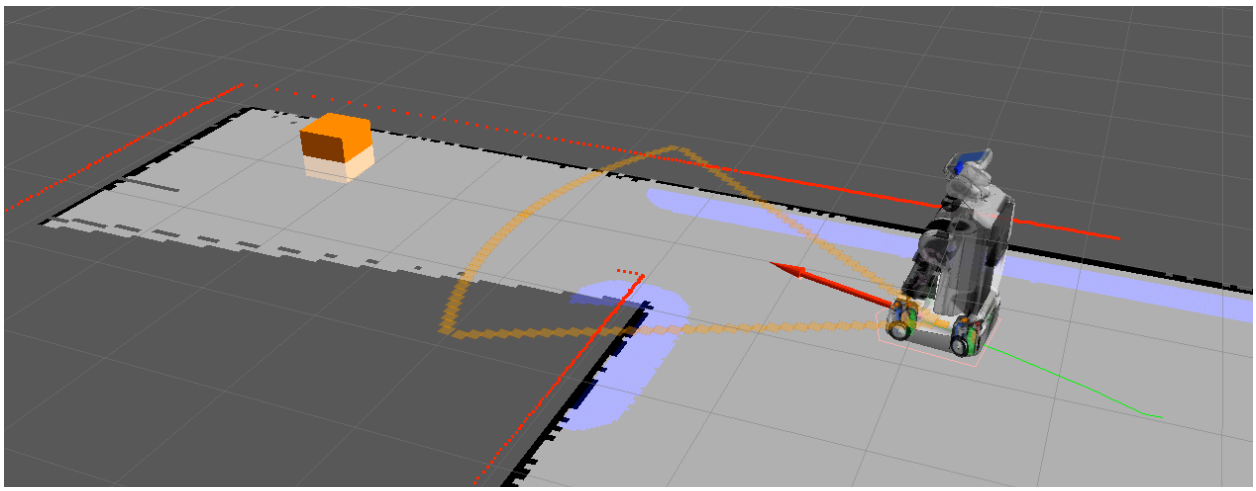

Figure 3: The fake_people_finder uses an interactive marker to change the position of the person while the simulation is running. The orange outline shows the simulated sensor's field of view.

Planner

In order to compute an effective policy for the robot to actuate, the planner must solve the underlying problems of maintaining an accurate internal model of the robot's state and utilizing that model to locate the next-state of highest probable reward. Upon doing so, it may then send the goal to the navigation stack to be evaluated, and subsequently repeat the computation.

We chose to use a POMDP model to effectively track the robot's state throughout its operation, updating the model according to information received from that navigation stack as well as the people finder. We implement the planner as a service loop that runs according to a preset loop rate (0.5Hz), calculating the optimal policy at every iteration and subsequently commanding the robot to execute the policy, waiting until a confirmation of success or failure is returned by the navigation stack before recomputing the optimal policy and repeating.

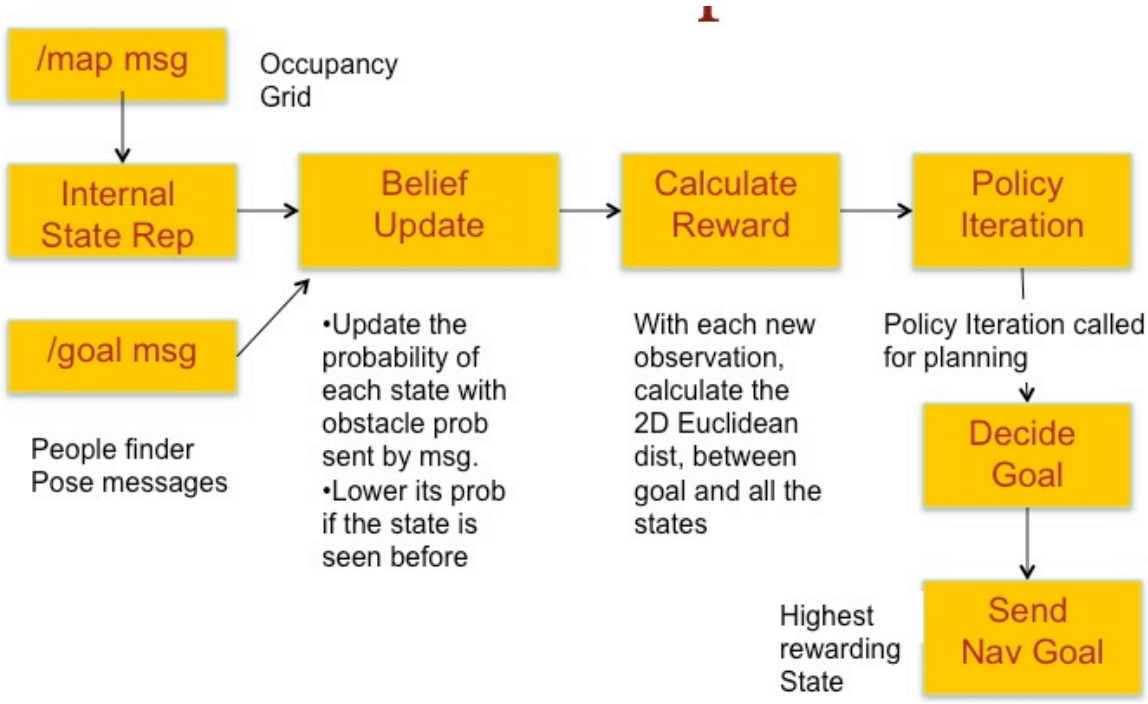

Figure 4: The planner calculates the next best policy using a multi-step process, with its internal data delivered asynchronously by the map and fake people finder.

The POMDP model of the robot's world state is comprised of a vector of known possible states in the map, with each state retaining information denoting its real-space position, its state-space position, and a heuristic value representing the reward of travelling to the state. The different processes that the planner utilizes to manipulate the POMDP model are outlined in Figure 4.

Whenever the planner receives an update to the world map from the navigation stack, it updates the heuristic value of each state according to the probability of an obstacle existing within the state that is returned from the map, decreasing the value of the state's reward by a function of the obstacle probability displayed at that location in the map.

With every update from the people finder, the planner updates the heuristic value of the states containing each of the observed people, increasing the reward of the state by a function of the observation probability introduced by sensor noise. In this manner, the planner is able to keep a continuously updated distribution of belief values over every state in the known map,

providing the information necessary for computing the optimal goal state as well as the policy required to move to that location.

In order to compute the optimal goal state for which to move the robot to, the planner selects the state with the highest likelihood of containing a hidden person and computes a policy for reaching them according to a value iteration algorithm, calculating the highest reward possible at each state for a preset depth (*num_lookahead*) before returning the highest reward next state to travel to from the current state.

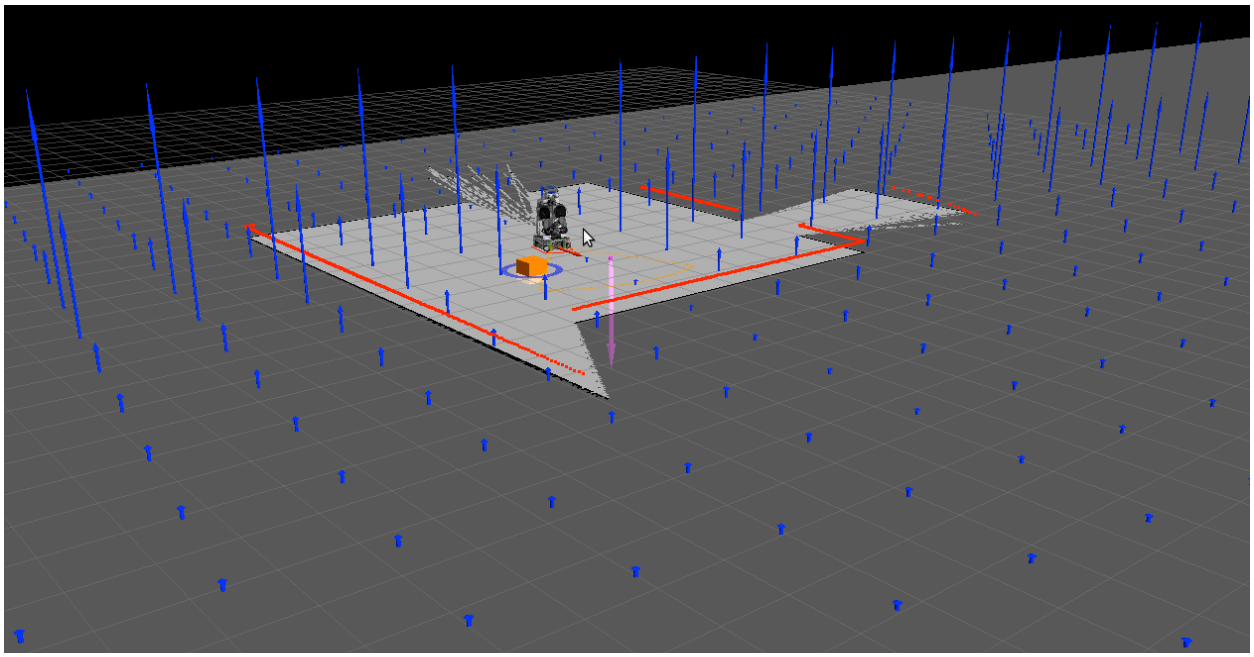

Figure 5: The planner displays the distribution of rewards over the states utilizing blue arrow markers at each state location, with the marker length varying according to the reward of the state.

We implement the value iteration algorithm by iterating over all of the states in the POMDP model and calculating the maximum reward of each state within the model for a depth of *num_lookahead* and returning the highest reward state as the next goal state for the robot. The calculation is implemented via a recursive function that takes a source state and a lookahead amount as input parameters, returning the maximum reward for the source state based upon the maximum reward of all states with respect to the source state as well as the

probability of the source state being reachable and the distance between the source state and the goal.

If the number of lookahead passed into the function is one, the maximum reward of each of the states in the iteration is returned as the default of one divided by the total number of states (1/pomdp.num_states) -- if the lookahead is greater than one, the maximum reward is recursively computed. When the full tree has been explored, we're left with the maximum reward next state from the robot's current state with respect to the goal. Figure 5 displays the reward distribution across all states during a run of the planner's computation.

The best next state is then sent to the navigation stack through a *MoveBaseClient*, provided by the *actionlib* package for sending simple navigation goals to the robot. We synchronously send a request for the robot to move to the goal state afforded by our planning algorithm, and update the value of the just-reached state according to whether or not a person was actually found at that location -- if a person was found, we deem our operation completed, otherwise we return to the main service loop to continue the planning process.

Conclusions

We were able to simulate a robot capable of building a map of an unknown environment using SLAM, detecting simulated people via a fake people finder, and making plans to detect people at their most probable locations. It proved quite difficult to successfully implement the POMDP planning algorithms, due to a few reasons. These included issues in down-sampling the map space into a tractable state space as well as implementing an exploration algorithm that could handle noisy sensors. We have not accounted for noise in the sensor data in our current solution, and leave the topic for future study.