Visual Localization and POMDP for Autonomous Indoor Navigation

Chulhee Yun CHULHEE@STANFORD.EDU
Stanford University, Stanford, CA 94305 USA

Sungjoon Choi CHOIS@STANFORD.EDU
Stanford University, Stanford, CA 94305 USA

Abstract

In this project, we present a system that simulates a fully autonomous robot in an indoor environment. We built a virtual environment and simulated the robot's monocular camera input by rendering the scene in the robot's perspective. Using this simulated camera input, we trained the localization algorithm by uniformly sampling image inputs and collecting a dictionary of ORB descriptors and keypoints. We also discretized the environment and formulated it into two POMDPs with different scenarios, and used QMDP and Fast Informed Bound (FIB) method to calculate approximate solutions to the problems. Through simulation of the policies we could see that FIB with one-step lookahead strategy works the best among tested algorithms. Connection of localization algorithm with POMDP completes the system for fully autonomous robot simulation, and our policy was robust enough to perform trained tasks even with large observation noise.

1. Introduction

Throughout recent years, the technology of autonomous system is getting out of the laboratory and into the real life. The market of unmanned aerial vehicles (UAV, also known as drones) is drastically growing due to its versatility and broad range of application. Google driverless car has achieved its license to drive inside California, and Amazon is currently shipping orders using its robot warehouse.

In these systems, autonomous navigation is the most challenging and important part of their artificial intelligence. The task of autonomous navigation can largely be decomposed into three parts: mapping, localization, and decision making. An autonomous system first should understand its surrounding environment and locate itself in that environment. Then, the system must make a proper decision that maximizes its expected reward or minimizes expected cost.

Our original objective for this project was to combine a Simultaneous Localization and Mapping (SLAM) algorithm with POMDP to implement a fully autonomous robot. However, most of SLAM algorithms calculate the position and pose relative to the starting point instead of the global coordinate of the map, which makes application of our POMDP policy difficult. Moreover, the error of SLAM algorithm accumulates over time, unless there is a loop closure detected during the algorithm. Thus, we concluded that we should separate mapping (training) and localization (testing) for this particular problem. Using images generated from points uniformly sampled in the virtual space, we trained the localization algorithm by constructing a dictionary of ORB (Rublee et al., 2011) descriptors and true 3-D keypoint coordinates. When the test image arrives, the localization algorithm finds matches of descriptors between the test image and dictionary to estimate the robot's position and pose.

This estimate of position and pose can be fed into Partially Observable Markov Decision Process (POMDP) (Astrom, 1965) as some observation values. POMDP is a generalization of Markov Decision Process (MDP), in which agents do not have the ability to be fully aware of their current states. Instead, they only have some observations (e.g. estimated position) that help them guess what states they are actually in. Since the exact state is unobservable, we instead maintain a probability distribution of states (i.e. a vector) referred to as "belief state". The policy of POMDP is a collection of α-vectors, each of which corresponding to a particular action. Inner product of an α-vector with a belief state produces the expected utility of selecting the action corresponding to the α-vector. Thus, the optimal action for a belief state is the action corresponding to the α-vector that maximizes the inner product.

There are a number of algorithms to solve POMDP exactly (Smallwood & Sondik, 1973; Sondik, 1978), but it is known that in general cases, POMDP is intractable to solve exactly. Thus, there have been active research on ap-

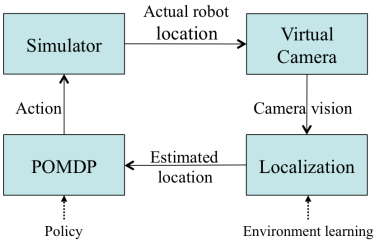

Figure 1. Block diagram of our system.

proximation algorithms for POMDP, in both offline and online settings. Two examples of the simplest and fastest offline approximation methods are QMDP and Fast Informed Bound (FIB) (Hauskrecht, 2000). QMDP creates one α-vector for each action, and iteratively updates the α-vector assuming full observability after the current state. FIB, on the other hand, takes account of the partial observability in α-vector updates. It also creates one α-vector per action, and it is known to produce a tighter upper bound for the exact utility than QMDP. There are also online approximation methods that are executed on run-time. Among them, one-step lookahead strategy combined with approximate value functions is known to upgrade performance compared to strategy without lookahead.

In this project, we combined visual localization algorithm and POMDP to train a robot to autonomously navigate through an environment and achieve given goals. In a 3D virtual living room, a robot observes its environment through its camera. We generated these camera images by rendering an indoor scene with OpenGL and POV-Ray. We trained the localization algorithm using this simulated camera image and later used the algorithm to estimate the position. Besides that, the indoor scene was discretized and formulated into a POMDP problem, and approximate solutions were computed using QMDP and FIB algorithms. Given that we have trained the localization model and have solved POMDP, we can simulate a fully autonomous robot that estimates its position and chooses actions to maximize its utility. The block diagram describing our system is in Figure 1.

The remaining sections of the report are organized as follows. Detailed explanation of our virtual environment and simulated camera input is given in Section 2. The algorithm for environment mapping and robot localization is depicted in Section 3. Section 4 illustrates the approaches taken to formulate the environment into POMDP and to solve the POMDP. The experimental results are presented in Section 5, and we conclude the report in Section 6.

2. Camera Simulation

Visual localization algorithm requires an input from visual sensors which is usually a camera. In real world situation, a robot reads visual information from the camera and localizes its global position. To simulate this behavior, we set up a virtual indoor scene and rendered the scene using a ray tracer. Every time the robot makes movement, the global position of the robot is given to the ray tracer as a viewpoint. A rendered image from the ray tracer is provided to the algorithm. The resolution of the video sequences is 640×480 (VGA) and the field of view is set to 57°, which simulates common consumer cameras for robots.

(a) (b)

Figure 2. High quality synthetic input sequences. Note the realistic characteristics which are challenging for the localization system in the wild. (a) High exposure washes out the chair due to low dynamic range making it hard to detect features. (b) Reflection on the window and the chrome plate changes along with a robots position causing unreliable keypoints.

We found the dataset published by Handa et al. (Handa et al., 2014) is useful for our purpose. Handa released the mesh of a synthetic living room as well as the videos recorded from several handheld trajectories. (The videos are not relevant in our case because we need on demand video sequences, not pre-rendered.) The scene we used in this project is shown in Figure 2.

3. Localization

Every time the robot makes movement, a synthetic color image in an unknown viewpoint is given to the localization pipeline. We begin with learning an indoor scene to estimate the pose of the robot. The information learned from the scene is stored, and then a synthetic image from the robot's vision is used to localize the robot along with this scene information.

3.1. Environment Learning

Environment learning is an one time process required to run before a simulation. First, we uniformly sampled locations of the scene. We sampled 637 locations in total. For each sampled location, we synthesize color inputs using the ray tracer described in Section 2. We also produced corresponding depth maps using OpenGL. Because the ray tracer is not capable of computing 3D coordinates for each pixel, we employed OpenGL and located a depth frame buffer to get 3D coordinates of the scene.

The pairs of color and depth images were then used to ex-

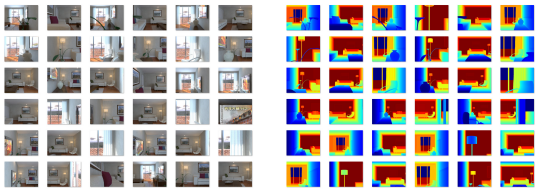

Figure 3. Set of color and depth images. 637 pairs of images were used to learn the environment.

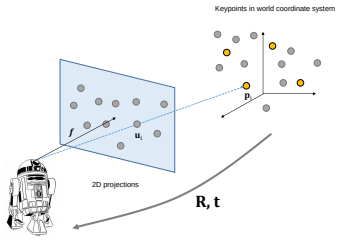

Figure 4. Illustration of the location estimation problem.

tract ORB features. OpenCV detected ORB keypoints from the color images. Again, we compute ORB descriptors for each keypoint. In average, 500 keypoints were detected per color image. The length of descriptor was 32.

Finally, we back-projected the keypoints to 3D coordinates. We can compute the 3D world coordinates because we know the position of the cameras for each synthesized image. These 3D world coordinates of the keypoints were stored along with the corresponding descriptors, resulting in $637 \times 500 \times 32$ vectors.

3.2. Location Estimation

We begin with extracting ORB features given an image like Section 3.1. Note that we have 2D keypoint and 32D descriptor pairs from a query image while there are 3D keypoint and 32D descriptor pairs in the dictionary.

The descriptors from the query image are matched with the descriptors in the dictionary using L2 distance metric for each training location. Given a set of correspondences between 3D points in the world coordinate and 2D keypoints in the query image, we retrieve the pose (R and t) of the robot. This is done by solving the Perspective-n-Point problem which returns R and t. Then, using the following equation, project the 3D points into the image plane to compute projection error and filter out outliers. The threshold we used is 3 px. Finally, our pipeline outputs the pose (R and t) where the number of inliers are maximized.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

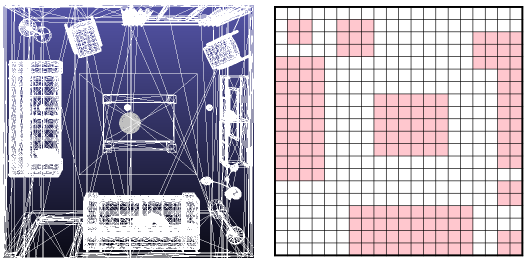

Figure 5. Discretization of virtual living room into 20×20 grid. The grid at upper left corner is the origin $(0, 0)$, the lower right corner being $(19, 19)$.

4. Approximate POMDP Solution and Policy Execution

4.1. POMDP Formulation

We will first clarify our notation related to POMDP. A POMDP problem can be formulated with a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P_T, P_O, R)$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ are sets of states, actions, and observations, respectively. $P_T(s' \mid s, a)$ and $P_O(o \mid s')$ are probability models of state transition and observation, $R(s, a)$ is the reward function.

Discretization of the state and observation space is illustrated in Figure 5. We divided the x, y coordinates into 20×20 grid, and marked the regions that the robot cannot pass into red. In this way, the state space became a space with 241 discretized states. In case of observation, we counted all 400 grids into possible observation because the localization algorithm may mistakenly output locations that the robot cannot pass. We also discretized the action space into eight actions $\mathcal{A} = \{N, W, E, S, NW, NE, SW, SE\}$. The names of the action represent the direction the robot moves inside the grid.

In case of transition model $P_T(s' \mid s, a)$, we assumed that there is a 80% chance of moving to the right direction, and there are 10% chances each of deviating 45°from intended direction of action. This is to take into account errors in pose estimation. If the resulting state is unable to pass or is outside the state space, the robot is blocked by the object and stays in its original state. After investigation about the performance of localization algorithm, we assumed that the observation model follows Gaussian distribution with mean being the true state and standard deviation being the half of grid size. We cropped the Gaussian distribution so that $P_O(o \mid s')$ has nonzero values only at 3×3 neighborhood of s'.

We solved the POMDP algorithm in two different scenarios, an easy one first and a relatively difficult one next.

1. The robot needs to visit the grid $(19, 0)$, the upper right corner. When the robot reaches the corner, it receives a reward of 100, and then the scenario ends.

2. In this scenario, the robot needs to circulate around the grids $(17, 19)$, $(19, 0)$, $(0, 0)$, and $(0, 19)$. The order of visit should be correct. Each time the robot visits the right grid, it receives a reward of 100. The scenario does not end, and repeats until the end of simulation. In this case, $|\mathcal{S}| = 241 \times 4$ because the state needs to carry information about the four different goal states.

On top of that, the robot receives a small negative reward -0.01 unless specified in the scenario. This is to encourage the robot to move faster to the goal.

4.2. Offline Learning Algorithms

Having set up the problem, we used QMDP and FIB algorithm to solve the problem. Both QMDP and FIB algorithm creates one α-vector per action, and they are initialized to zero. The update rule of QMDP is

$$\alpha_a^{(k+1)}(s) = R(s,a) + \gamma \sum_{s'} P_T(s' \mid s, a) \max_{a'} \alpha_{a'}^{(k)}(s'), \quad (2)$$

and the update rule of FIB is

$$\alpha_a^{(k+1)}(s) = R(s,a) + \gamma \sum_{o} \max_{a'} \sum_{s'} P_O(o \mid s', a) P_T(s' \mid s, a) \alpha_{a'}^{(k)}(s'). \quad (3)$$

4.3. Policy Execution

In policy execution, we used an online method called lookahead strategy. The standard decision strategy for the action was $\pi(b) = \text{argmax}_a \, \alpha_a^T b$, which considers only current time step. The lookahead strategy considers not only the current but also the next step to determine the action.

$$\pi(b) = \operatorname*{argmax}_a [R(b,a) + \gamma \sum_{o} P(o \mid b, a) U(\textsc{UpdateBelief}(b, a, o))], \quad (4)$$

where $R(b,a) = \sum_s R(s,a) b(s)$ and $P(o \mid b, a) = \sum_{s'} P_O(o \mid s', a) \sum_s P_T(s' \mid s, a) b(s)$.

4.4. Simulating in Continuous Space

Although we discretized the state and observation space, we can still run the simulation in continuous space using the policies obtained. Given the true (continuous) state values (x, y, θ) and observation values $(\hat{x}, \hat{y}, \hat{\theta})$, we can calculate which discrete state and observation these belong to, and calculate actions and update beliefs in the same way as discrete case. Instead of sampling s' from $P_T(s' \mid s, a)$, we update the state values in continuous space, by executing the selected action with some Gaussian noise.

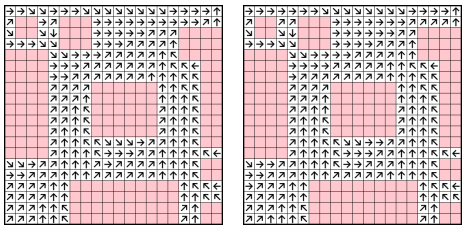

Figure 6. Illustration of policies for scenario 1 learned from QMDP and FIB algorithms.

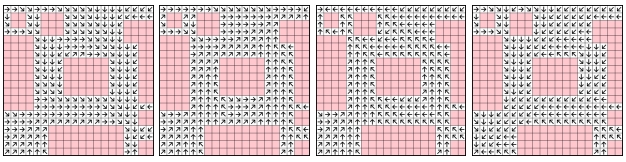

Figure 7. Illustration of policies for scenario 2 learned from FIB algorithms

5. Experimental Results

5.1. Policies Learned

Figure 6 shows a brief sketch of the policy obtained using QMDP and FIB algorithms. The arrow assigned to each state indicates the resulting policy when we believe we are in that state with probability 1. We can see that the two algorithms produce largely the same policy; however, performance resulting from these policies may significantly differ depending on the problem, because of different values of α-vectors. Figure 7 illustrates the policies for scenario 2, clustered according to the goal position. We can note that the robot tends to move to direction closer to the goals.

5.2. Simulation without Localization Algorithm

To verify the validity of our obtained policy, we performed simulations on our scenarios using the same observation model that we used to solve the POMDP. For each scenario, we tested four pairs of policy-strategy: (QMDP, Standard), (QMDP, Lookahead), (FIB, Standard), and (FIB, Lookahead).

All the simulation cycle started at state $(0, 19)$. For the first scenario, when the robot reaches the goal state the simulation was reset and initialized to the starting state. We performed the simulation for 1,000 steps and accumulated the rewards for evaluation. Since the second scenario did not have terminal condition, ten simulations were executed for 500 time steps each. The pair that has the biggest summed reward can be considered to be the best approach.

The results are summarized in Table 1. In the first scenario, we can notice that there is no large difference between the four methods because the scenario is too simple to solve. The minimum steps taken to finish scenario 1 is 25, which

Table 1. Total reward without localization algorithm.

Policy-Strategy	Scenario 1	Scenario 2
QMDP+Standard	3390.34	11651.17
QMDP+Lookahead	3490.35	12751.28
FIB+Standard	3490.35	13551.36
FIB+Lookahead	3490.35	14351.44

Table 2. (Estimated) total reward with localization algorithm.

Policy-Strategy	Scenario 1	Scenario 2
QMDP+Standard	2090.21	99.01
QMDP+Lookahead	2290.23	99.01
FIB+Standard	3390.34	199.02
FIB+Lookahead	2290.23	99.01

is hard to attain under partial observability and stochastic actions. From the accumulated rewards we can conclude that the algorithms took approximately 28.5 steps in average to finish scenario 1, which is near optimal.

As problem gets bigger and more difficult, however, it is demonstrated that algorithm with tighter utility upper bound is more likely to do better; FIB outperforms QMDP in scenario 2. Also, we can notice that adopting lookahead strategy improves the performance of the policy. The ideal utility that a robot can get in this situation is 24952.5. Considering this, the performance of the algorithm degraded due to partial observability and stochastic actions.

5.3. Simulation with Localization Algorithm

Finally, we performed simulation of our policies using the output of localization algorithm as observations. In reality, output values of localization algorithm contains many outliers. Especially when the camera is close to walls or surfaces, the algorithm barely finds features to match with trained dictionary, producing unrealistic estimates of the position and pose. So we could see that the performance of the POMDP policy simulation got worse when coupled with localization algorithm. Despite frequent outliers in observation, however, we could observe that the robot eventually achieves the goal, although more time steps were required.

Due to huge running time of the localization algorithm, we could not have enough time and resources to run the simulation with as many time steps as in Section 5.2. For scenario 1, we measured the performance by sampling the number of time steps taken for the robot to reach the goal and converting it to expected accumulated rewards in 1,000 time steps. In scenario 2, we simulated the policy and accumulated the reward given in 100 time steps. The results are summarized in Table 2. Even though (FIB, Standard) seems to be the best option to opt for, we have to note that this is only a sample from limited number of time steps.

6. Conclusion

In this project, we constructed a system that can simulate a fully autonomous robot inside an indoor environment. Using an existing mesh of indoor scene, we constructed a virtual environment where the robot lives. The camera input to the robot can be simulated by rendering the scene from the robot's view. The images generated from random points of the environment were used to train the localization algorithm. On the other hand, the indoor environment was formulated into two discretized POMDPs according to different scenarios and approximate algorithms such as QMDP and FIB were used to solve the POMDPs. Given the policy, we could simulate the robot's behavior in continuous state space. Simulation using the observation model verified that our solution works reasonably well, and that FIB with lookahead policy execution strategy performs the best among tested methods. After plugging the output of localization algorithm into the observation of POMDP simulator, we saw that although the performance degrades due to frequent outliers in observation, the robot robustly reaches the goal after some increased number of time steps. If we had more time, we would be willing to improve the localization algorithm and test variety of interesting scenarios that can be formulated in POMDPs.

References

Astrom, Karl J. Optimal control of markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

Handa, Ankur, Whelan, Thomas, McDonald, John, and Davison, Andrew J. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *ICRA*, 2014.

Hauskrecht, Milos. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13(1):33–94, 2000.

Rublee, Ethan, Rabaud, Vincent, Konolige, Kurt, and Bradski, Gary. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564–2571. IEEE, 2011.

Smallwood, Richard D and Sondik, Edward J. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

Sondik, Edward J. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.