STOCHASTIC PATH PLANNING FOR UNDERWATER ROBOT

Laukika Sankpal
Arizona State University
lsankpal@asu.edu

***Abstract*—Path planning for an underwater robot is currently a topic of interest for many pieces of research due to the challenges faced underwater. The environment underwater is uncertain since we do not know the intensity and direction of the water currents. We have used Gazebo in ROS for setting up the underwater environment of a 10x10 grid with predefined vortices across the map as the disturbance. Further, we implemented a stochastic path planning algorithm partially observable Markov decision process (POMDP), to take the robot from starting point o goal point by successfully avoiding the disturbances.**

***Keywords*— Underwater robot, Robot Operating System, Gazebo, POMDP, Monte-Carlo tree search**

I. INTRODUCTION

Path planning is a crucial aspect of the functioning of any mobile robot with some degree of freedom. A path planning algorithm helps in the movement of a robot from a current location to a goal location in a given environment. It also ensures obstacle avoidance by the robot. Additionally, it should return an optimal trajectory from one point to another with minimum cost. Underwater path planning is more challenging as compared to terrestrial path planning. Some of the challenges are localization, underwater vision, and resistance due to water current. Global Positioning System (GPS) is affective on land, but that is not the case underwater since there is no propagation of GPS signals underwater. Visuals and object detection underwater is also difficult. The movement of the underwater robot also has to take into consideration the resistance of unpredictable water currents. We have implemented the POMDP algorithm taking into consideration the uncertainty of the robot's location.

II. EXPLANATION OF THE SOLUTION

For localization underwater, we made use of the IMU sensor with some predefined noise for the uncertainty. The underwater robot has some degree of uncertainty regarding its current location at any given time. Due to this, upon taking any action, the robot might not end up in the desired location. To calculate this error, we used the PID controller. The input to the PID controller is the desired location that the robot should end up at, and its output is the actual location that the robot ends up at. With the help of the feedback mechanism of the PID controller, we calculate the accumulated error after multiple estimations.

The next step is to find an optimal policy that can guide the agent from the initial position to the goal destination while compensating the uncertainty due to sensor noise. For this, we use POMDP (Partially Observable Markov Decision Process). POMDP can deal with uncertainty in robot's understanding and optimally trade between the actions that gather information and actions that lead to the desired goal. The real-world problems of MDP are required to consider an exhaustive large set of parameters that are difficult to specify from domain knowledge alone.

In this work, we have considered a 10 x 10 grid, and the robot can face in any of the four directions, north, south, east, and west. The actions that the robot can take are, move north, move south, move east, and move west. We gave probability distribution to all the locations in the state space. The robot is not able to observe its state in the MDP. It can only make an observation, which is a function of the state based on which a belief is formed of where the robot is in the state space.

2.1 The POMDP model

The POMDP here consists of the n-tuple {S, A, O, T, Ω, R,γ}. S, A, and O are sets of states, actions, and observations. The transition function T (s′|s, a) is a distribution over the states the agent may transition to after taking action a from state s. The observation function Ω(o|s, a) is a distribution over observations o that may occur in state s after taking action a. The reward function R(s, a) specifies the immediate reward for each state-action pair. The factor $\gamma \in [0, 1)$ weighs the importance of current and future rewards. In the POMDP model, the agent must choose actions based on past observations; the true state is hidden. The belief, a probability distribution over states, is a sufficient statistic for a history of actions and observations. The belief at time t + 1 can be computed from the previous belief, bt, the last action a, and observation o, by applying Bayes rule.

$$b_{t+1}^{a,o}(s) = \Omega(o|s,a) \sum_{s' \in S} T(s|s',a) b_t(s') / Pr(o|b,a), \quad (1)$$

Where,

$$Pr(o|b,a) = \sum_{s' \in S} \Omega(o|s',a) \sum_{s \in S} T(s'|s,a) b_t(s)$$

If the goal is to maximize the expected discounted reward, then the optimal policy is given by:

$$V_t(b) = \max_{a \in A} Q_t(b,a), \quad (2)$$

$$Q_t(b,a) = R(b,a) + \gamma \sum_{o \in O} Pr(o|b,a) V_t(b^{a,o}), \quad (3)$$

Where, the value function V (b) is the expected discounted reward that an agent will receive if its current belief is b and Q(b, a) is the value of taking action a in belief b [1].

2.2 Monte-Carlo tree search

With the help of Monte-Carlo simulation, the Monte-Carlo tree search [2] evaluates the nodes of a search tree in a sequentially best-fit order. For each state s and action a pair, there is a node with value Q(s, a) in the tree and a counter variable N(s, a) for each action a. The overall count N(s, a) = ∑a N(s, a). At the beginning of the process, each of the nodes is initialized to Q(s, a) = 0, N(s, a) = 0. The mean return from state s of all simulations where the action a was selected from state s. Every simulation starts with the current state st and is

divided into two stages: In the first stage, a tree policy is used within the search tree, and in the second stage, a rollout policy [3] is used once simulations are out of the scope of the search tree. In the most basic version of the MCTS, a greedy tree policy is used in the first stage. It selects the action a with the highest value. In the second stage, it selects a uniform rollout policy. At the end of each simulation, a new node is appended to the search tree containing the first state visited in the second stage.

2.3 Monte-Carlo planning in POMDP

In Partially Observable Monte-Carlo Planning POMCP, the UCT (Upper Confidence bound applied to Trees) search that picks the actions at each stage, and the particle filter updates the belief state of the agent.

2.3.1 Partially ObservableUCT (PO-UCT)

We use the search tree of histories instead of states by applying the UCT algorithm to a partially observable environment. This search tree has a node T(h) =(N(h), V(h)) for each represented history. N (h) keeps a count of the occasions that history h has been visited. V (h) is the estimation of history h, assessed by the mean return of all simulations beginning with h. New nodes are initialized to (Vinit(h), Ninit(h)) if domain information is available, and to h0, 0i in the other case. We assume that the belief state B(s, h) is known precisely. Every simulation begins in an initial state that is sampled from B (·, h_t). These simulations are separated in two stages in the fully observable environment of the algorithm. In the primary phase of the simulation, when the child nodes exist for all the children, actions are chosen by UCB1. Then the actions are selected to maximize the augmented value. In the second stage of simulation, actions are selected by rollout policy πrollout (h, a) based on history. After every simulation, precisely one node is added to the tree. This corresponds to the first new history encountered during that simulation [3]

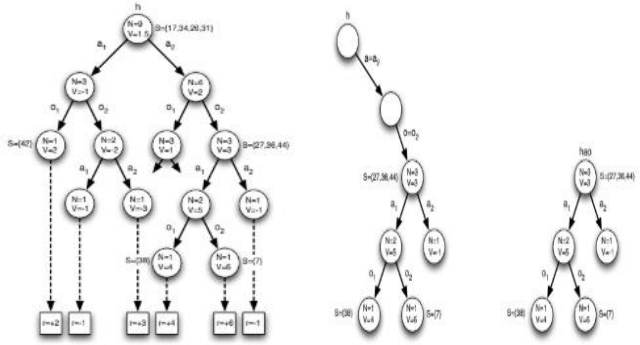

Figure 1.

Figure 1 gives us the idea about POMCP with 2 observations, 2 actions, 50 states, and no intermediate rewards in the environment. Multiple simulations are taken into consideration and used to construct a search tree by the agent, and each history is evaluated by its mean return (left). This search tree is then used to select a real action a and observe a real observation o (middle). The next thing the agent needs to do is to prune the tree and begin a new search from the updated history hao (right).

2.3.2 Monte-Carlo belief state updates

The majority of POMDP planning methods operate by updating the belief states by Bayes' theorem [4]. However, even a single update in the vast state spaces might be computationally infeasible, and a compact representation of the transition probabilities or the observation probabilities may not be available. We use unweighted particle filtering to approximate the belief state in large POMDP's. Furthermore, we use the Monte-Carlo method to update the particles based on the sample observations, rewards, and state transitions. We can implement the unweighted particle filtering efficiently with the blackbox simulator, without even requiring an explicit POMDP model. It provides excellent scalability to more significant problems [3].

2.3.3 Partially observable Monte-Carlo planning

POMCP is a combination of Monte-Carlo belief state updates and PO-UCT. It shares the same simulations for both Monte-Carlo procedures. Every single node in the search tree T(h) = (N(h), V (h), B(h)),contains a set of particles B(h) along with N(h) and V(h). This search procedure is started from the current history h_t. Each of the simulations starts from a specific start state that is sampled from the belief state B(h_t).

Algorithm 1 Partially Observable Monte-Carlo Planning

```
procedure SEARCH(h)
    repeat
        if h = empty then
            s ~ I
        else
            s ~ B(h)
        end if
        SIMULATE(s, h, 0)
    until TIMEOUT()
    return argmax_b V(hb)
end procedure

procedure ROLLOUT(s, h, depth)
    if γ^depth < ε then
        return 0
    end if
    a ~ π_rollout(h, ·)
    (s', o, r) ~ G(s, a)
    return r + γ.ROLLOUT(s', hao, depth+1)
end procedure

procedure SIMULATE(s, h, depth)
    if γ^depth < ε then
        return 0
    end if
    if h ∉ T then
        for all a ∈ A do
            T(ha) ← (N_init(ha), V_init(ha), ∅)
        end for
        return ROLLOUT(s, h, depth)
    end if
    a ← argmax_b V(hb) + c sqrt(log N(h) / N(hb))
    (s', o, r) ~ G(s, a)
    R ← r + γ.SIMULATE(s', hao, depth + 1)
    B(h) ← B(h) ∪ {s}
    N(h) ← N(h) + 1
    N(ha) ← N(ha) + 1
    V(ha) ← V(ha) + (R − V(ha)) / N(ha)
    return R
end procedure
```

Figure 2.

As described in the algorithm, the simulations are performed using the partially observable UCT algorithm. The belief state B(h) is updated to included to include the simulation state for every history h encountered during the simulation. At the end of the search, the action a_t that has the greatest value and receives a real observation o_t from the world is selected by the agent. Now the new root of the search tree is the node T($h_t a_t o_t$), and the belief state B($h_t$ao) is the agent's new belief state. All other histories are now impossible as the rest of the tree is pruned. Figure 1. And Figure 2 explains the entire POMCP algorithm [3].

2.3.4 PyPOMDP

PyPOMDP is a repository that has the python implementation of POMDP solver. The input to the POMDP solver must be a file that specifies the following inputs. (1) Discount factor (2) Values: reward/cost (3) States (4) Actions

(5) Observations (6) Transition Probabilities P(s' | s, a) (7) Observation Probabilities P(e | s, a) (8) Rewards R(s, a, s')

```
# Michael's 1D maze

discount: 0.75
values: reward
states: left middle right goal
actions: w0 e0
observations: nothing goal

T: w0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0
0.333333 0.333333 0.333333 0.0

T: e0
0.0 1.0 0.0 0.0
0.0 0.0 0.0 1.0
0.0 0.0 1.0 0.0
0.333333 0.333333 0.333333 0.0

O: *
1.0 0.0
1.0 0.0
1.0 0.0
0.0 1.0

R: * : * : goal : goal 1.0
```

```
# Stochastic version of Michael's 1D maze

discount: 0.75
values: reward
states: left middle right goal
actions: w0 e0
observations: nothing goal

start:
0.333333 0.333333 0.333333 0.0

T: w0
0.9 0.1 0.0 0.0
0.9 0.0 0.0 0.1
0.0 0.0 0.1 0.9
0.333333 0.333333 0.333333 0.0

T: e0
0.1 0.9 0.0 0.0
0.1 0.0 0.0 0.9
0.0 0.0 0.9 0.1
0.333333 0.333333 0.333333 0.0

O: *
1.0 0.0
1.0 0.0
1.0 0.0
0.0 1.0

R: * : * : goal : goal 1.0
```

Figure 3.

The left side of Figure 3 shows Michael's 1D maze input file to the POMDP solver, and the right side shows a stochastic version of it.

III. MY CONTRIBUTION IN THE PROJECT

(1) Problem design
(2) Research work for the algorithm selection of the project.
(3) Design of transition and observation model for PyPOMDP

Due to the uncertainty of the robot's understanding of its current position, upon taking any action, it might end up at a location different than the location it should end up at if it had the best knowledge of its current location.

The probabilities calculated of the locations that the robot might end up at, with respect to the robot's understanding of its current location, are called transition probabilities, and those calculated with respect to the actual current location of the robot are called as observation probabilities.

The transition model for this project is designed as follows.

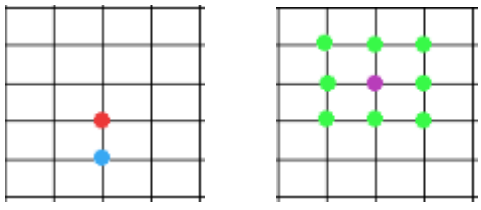

Figure 4.

In the left grid, the first dot represents the actual location of the robot and the dot below that represents the robot's understanding of its current location. The probability distribution for this is shown in the right grid upon taking action 'MOVE NORTH'. The dot in the center which is one location ahead of the actual location of the robot but 2 locations ahead of the robot's understanding of its location, has probability 0.7, and the rest of the surrounding 8 locations have probability 0.3/8 each that the robot might end up at those locations.

Since the grid is a 10x10 map, it was difficult to calculate the probability of each location for every current location on the grid. For that, the next step was to write a code for generating these probabilities and then feed these transition model and observation model to the PyPOMDP solver.

Along with the probabilities, the PyPOMDP also needs the reward values based on the robot's location with respect to the goal location. The reward values for this project are +20 for top right corner, +40 for bottom left corner, +100 for the bottom right corner (goal), and -1 for all other locations.

IV. DESCRIPTION OF THE RESULTS

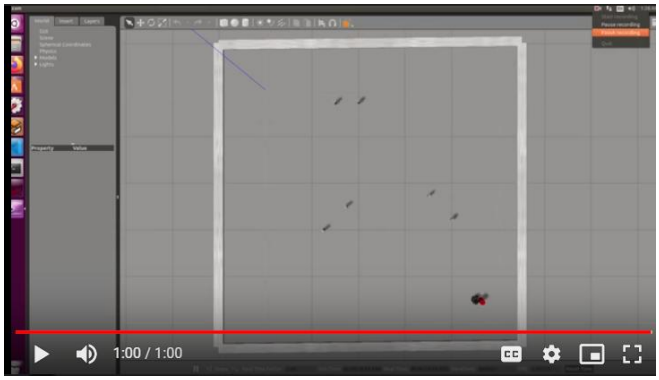

Figure 5.

The environment was set up on a 10x10 grid map in Gazebo World in ROS. The start location of the Gazebolt3 robot with the moving speed of 0.3m/s and rotation speed of π/10 rad/s (20s/r) and 0 rotation radius, is the top left corner. The robot reaches the goal position at the bottom right corner by successfully avoiding the obstacles which, represents the water current.

This is the link to the above video, https://www.youtube.com/watch?v=jRC-E5yia3c&feature=youtu.be

TEAM MEMBERS

- Siyu Liu
- Alena Chang
- Yuhao Jiang

REFERENCES

[1] F. Doshi, J. Pineau, and N. Roy. Reinforcement Learning with Limited Reinforcement: Using Bayes Risk for Active Learning in POMDPs. *Artificial Intelligence*, 2011.

[2] D. Bertsekas and D. Casta˜non. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.

[3] D. Silver and J.Veness. Monte-Carlo Planning in Large POMDPs. *Proc. Neural Inf. Process. Syst.*, pp. 1-9, 2010.

[4] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.