Multi-agent Path Planning using Markov Decision Processes

Devon Webb

I. Introduction

The final project I worked on for this class has two main elements. The first was modifying the simulator to work with three agents instead of one. The second element was to create a planning algorithm that uses markov decisions processes(MDP's) to calculate the ideal path for each agent given a grid with reward and penalty zones. This also included visualizing the reward and penalty zones in the simulator. In this report I will discuss the steps that were required to complete both elements the project and future additions that could be built upon from this project.

II. Related Works

The motivation for this project comes research shown in [1]. In the work the author describes many of the advantages to using an MDP to describe the actions of a robot as opposed to other methods like state machines. When robots are interacting with other robots or people in an environment, the effect the agents actions have a probabilistic nature that is dependent on it's environments and other agents. MDP's also provide a clear way of describing an environment where many agents strive to achieve a similar goal and where actions of the said agent have a probility of achieving a variety of effects. These attributes are useful when considering multi-agent, small, fixed wing vehicles as they may easily be blown off course by strong gusts of wind.

[1] also shows how MDP's can be used to maximize the goals of a group of collaborating or competing robots. This can be useful for groups of fixed wing vehicles that are surveying an area or simply trying to get agents to certain goal locations as quickly as possible. Processing the MPD's can also be done very efficiently using one of many open source probabilistic model checking software. One such example is PRISM[2] which is also used by [1].

III. Simulating Multiple Agents

Adding two additional agents to the simulator required going back to each special project in the class and modifying the visualizer files to handle multiple agents. The DrawMav function had to be adjusted to include a color preference to be able to distinguish between each agent, the effect of which is shown in Fig 1. In my implementation, I passed an int to the DrawMav function that specifies which MAV I'm referencing. The int is then used to select a color for that MAV.

The algorithm functions we made to calculate dynamics, controls, autopilot commands, way points, paths, and path following commands were all built in a way that allowed for individual instances of each algorithm for each agent, as seen in Fig 2, and then allowed for updates for each agents path, autopilot, and dynamics functions.

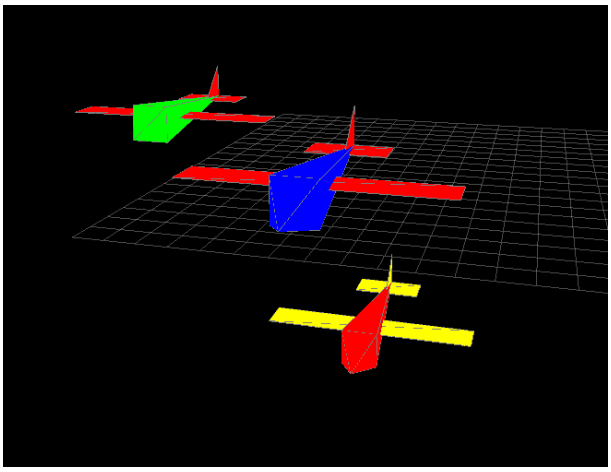

Fig. 1. Visual of three agents as shown in simulator

```
path_follower = PathFollower()
path_manager = PathManager()
autopilot = Autopilot(SIM.ts_simulation)
observer = Observer(SIM.ts_simulation, initial_state)

mav2 = MavDynamics(SIM.ts_simulation, initN=(init2[0]*300+150), initE=(init2[1]*300+150+offset))
autopilot2 = Autopilot(SIM.ts_simulation)
path_follower2 = PathFollower()
path_manager2 = PathManager()
observer2 = Observer(SIM.ts_simulation, initial_state)

mav3 = MavDynamics(SIM.ts_simulation, initN=(init3[0]*300+150), initE=(init3[1]*300+150), alpha=np.pi/2
autopilot3 = Autopilot(SIM.ts_simulation)
path_follower3 = PathFollower()
path_manager3 = PathManager()
observer3 = Observer(SIM.ts_simulation, initial_state)
```

Fig. 2. Code showing individual instances of autopilot, path planning, path following, way point, observer, and dynamics elements

Each agents state, path, and way points were then passed to a modified way point viewer. This viewer was used with special project 11 and the output is shown in Fig 3.

IV. Markov Decision Process for Path Planning

An example of how MDP's can be used for path planning is shown in [1] as a simple example of a robot navigating a grid. This is shown again in Fig 4. The graph from [1] can be navigated by calculating the utility of each square in the grid using the following equation.

$$U(s) = R(s) + \gamma \max_{a \epsilon A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

The equation for Utility *U(s)* combines the reward of the current state of an agent (in the grid case, each grid is considered the state of the agent) with the utility of the next possible state that has the highest utility and the largest probability of being achieved given an action a. The second half of the equation is multiplied by the coefficient γ which

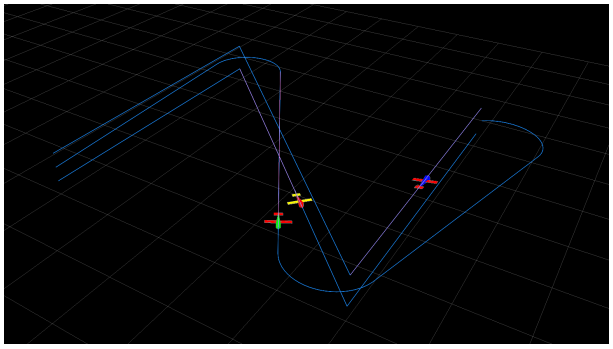

Fig. 3. Three agents visualized using special project 11. Agent 1 is following it's path using a fillet algorithm, agent 2 uses the dubins algorithm, and agent 3 uses a straight line algorithm.

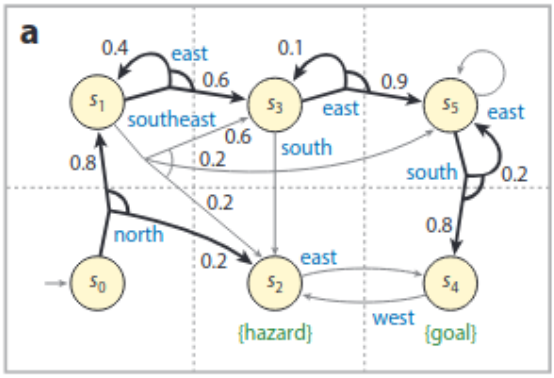

Fig. 4. Grid example of a MDP shown in [1] in figure 1.A

has a value of $0 < \gamma < 1$ which is chosen by the user and specifies how much emphasis to put on the next states utility.

In can also seen the utility equation is recursive. As such, one approach to calculating the utility of the grid is to find the utility of each state, assuming that each other state starts with a utility of 0, and continuing to recalculate the utility of each state until values converge.

Once the utility of each state is known, a path can be planned given an initial location by choosing the action that leads to the next state with the highest utility. In the grid example, this means moving the agent to an adjacent square with the highest utility of the surrounding grid points. this is done until a reward zone is achieved.

V. MDP Implementation in a Simulator

To implement an MDP path planning algorithm in the simulator, I first defined a grid with reward and penalty locations as well as starting points for three agents as shown in Fig 5. The actions available in each grid location are to move to any of the connecting squares with a 0.9% chance of success and a 0.1% chance of remaining in the same grid location. These percentage values were picked to be simple and to demonstrate the path planning algorithm, but could be changed in the future to account for factors such as wind.

I then calculated the utility of each zone as shown in Fig 6 and found a path for one of the agents using the

-1	-1	-1	-100	-1	-1
-1	-1	-1	100	-1	-1
-1	-1	100	-1	-1	-1
-1	100	-1	-1	-1	-1
-1	-1	-100	-1	-100	-1
-1	-1	-1	-1	-1	-1

Fig. 5. Grid example used in simulator with three reward zones and three penalty zones. Every other zone is give an reward value of -1 to encourage the path planning algorithm to move towards the reward zones.

-1 -10.1	-1 -2.111	-1 -1.111	-100 -90	-1 -0.111	-1 -1.01
-1 -10.1	-1 -0.100	-1 9.00	100 100	-1 9.00	-1 -1.111
-1 -0.100	-1 9.00	100 100	-1 9.00	-1 -2.1	-1 -1.110
-1 9.000	100 100	-1 9.00	-1 -0.100	-1 -10.0	-1 -1.111
-1 -0.100	-1 9.000	-100 -99	-1 -1.000	-100 -100.1	-1 -1.111
-1 -1.111	-1 -1.111	-1 -1.111	-1 -1.111	-1 -1.111	-1 -1.111

Fig. 6. Fig 5 with utility values of each square and a path for the bottom right agent.

utilities. Once a path was found for one agent, that agent and the achieved goal location were removed from the grid and the utilities of each state were recalculated. Another path was then found using the new utilities and this process was repeated until no agents remained.

The initial grid with all reward zones were then visualized within the simulator and the paths were given to the agents with the instructions to circle the last way point. This final solution is given in Fig 7

VI. Conclusion

This project successfully implemented and visualized a three agent simulation that used a MDP to plan a path through a series of grid locations. Implementing this MDP planner helped me gain a better understating of how they can be used for multi-agent planning and implementing 3 agents in the simulator improved my understanding for how each element of the simulator functions.

If I had more time to work on this project there are several additions I would make. These include adding random gusts of wind and changing the probabilities of movement from square to square in the MDP to account for that. I would also use an open-source model checker to find the best paths for all agents. I wasn't able to get PRISM working for this project, but adding it's functionality would allow for more efficient path calculations instead of finding the utility of every square for each agent. I would also change the code I used to visualize the paths and way points to allow for any number of agents instead of being set to three.

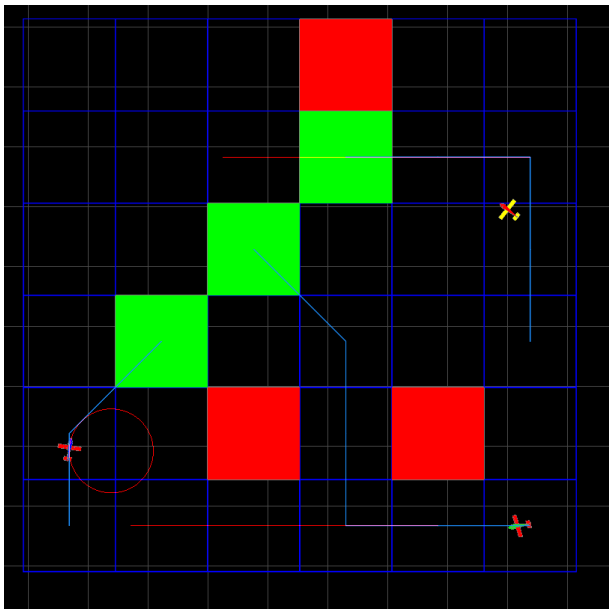

Fig. 7. Grid, agents, and path visualized within the simulator

VII. Project Repository

the github Repo for this project, which includes all code and the slides for the Final Presentation, can be found at https://github.com/dwebb9/ECEN674$_{M}ulti Agent_{M}PD$

References

[1] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic model checking and autonomy," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, 2022.

[2] ——, "Prism 4.0: Verification of probabilistic real-time systems," in *International conference on computer aided verification*. Springer, 2011, pp. 585–591.