Path Planning with Dynamic Obstacle Avoidance for a Jumping-Enabled Robot

Peggy (Yuchun) Wang* and Caitlin Hogan†
Department of Computer Science, Stanford University

Robotic navigation is an important concern for any autonomous project, and for robots with specialized modes of movement, it can be especially difficult to conduct path planning. In this work, the authors evaluate a path planning model for the Stanford Doggo, a quadruped robot unique in its compliant gait and ability to jump variable heights. Implementation of multiple common noisy sensors used in robotics, as well as particle filtering, allows for a somewhat realistic model of navigation. The difficulty of path planning in three dimensions can be avoided for this robot via use of a "jump-capability" overlay, indicating whether the robot can traverse to a tile at a certain height. While the model presented is grid-based, its insights into autonomous motion can be applied to this robot.

I. Introduction

A. Background

Path planning is integral to any autonomous robotic navigation system. The Stanford Robotics Extreme Mobility team has recently developed a jumping-capable four legged robot (nicknamed "Doggo") which can be controlled manually (Figure 1). In order to autonomously navigate the terrain, it requires a variety of sensors and a path planning system. Inspired by this problem, we focus on developing a platform to test a path planning algorithm for this specific robot. By incorporating the constraints of its jumping capabilities, a simulation model can be used to test methods of dynamic obstacle avoidance, with the eventual goal of implementing the algorithm on the robot.

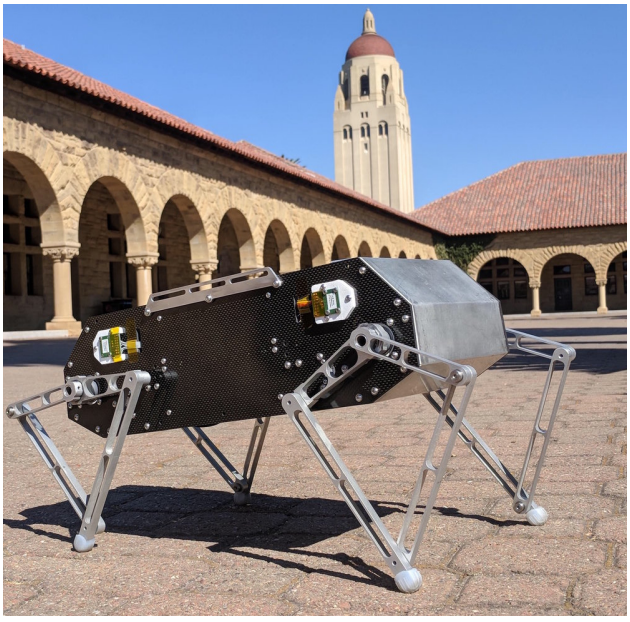

Fig. 1 Stanford Extreme Mobility Four Legged Jumping Robot aka "Doggo"

*peggy.yuchun.wang@cs.stanford.edu
†cahogan@stanford.edu

Planning in continuous spaces is usually computationally intensive and may be intractable. Discretizing the continuous two-dimensional world into a grid speeds up computation and is better able to represent obstacle beliefs in different states [1]. Some 2D path planning algorithms which work well with grid-based worlds include the Dynamic Window Approach [2], Dijkstra's Algorithm, and A* [3]. Recently, variations of A* were implemented successfully for path planning for autonomous vehicles in the DARPA Urban Challenge [4]. A* works well with planning around stationary obstacles [5]. Since we have dynamic obstacles in addition to static obstacles, we can re-plan with A* at every timestep. We assume that at every node, if the dynamic obstacle probabilities are above a certain threshold, there is a static obstacle in that space. We then formulate the environment as a graph, with adjacent nodes being connected in a 2D space.

B. Sources of Uncertainty

One of the biggest sources of uncertainty in autonomous navigation for a physical robot is noisy sensor data [6]. This sensor uncertainty leads to observation uncertainty, especially as to whether there is an obstacle nearby, which then impacts path planning decisions. One way to represent observation uncertainty is through a Hidden Markov Model (HMM) [7]. A common way to represent obstacle uncertainty is through belief vectors for every state based on sensor observations [8]. Each sensor contributes some information about the state, with some level of confidence. Based on the belief vectors and an online planning algorithm, we are able to compute the probability of obstacles being in the next states and plan our path accordingly.

II. Methods and Approach

A. Framework

In order to develop a model for a three-dimensional grid world, in which an agent (representing the robot) makes use of realistically noisy sensor data to navigate an unknown environment, we will be extending the Driverless Car simulation developed by Chris Piech for the Stanford CS221 "driverless car" assignment.

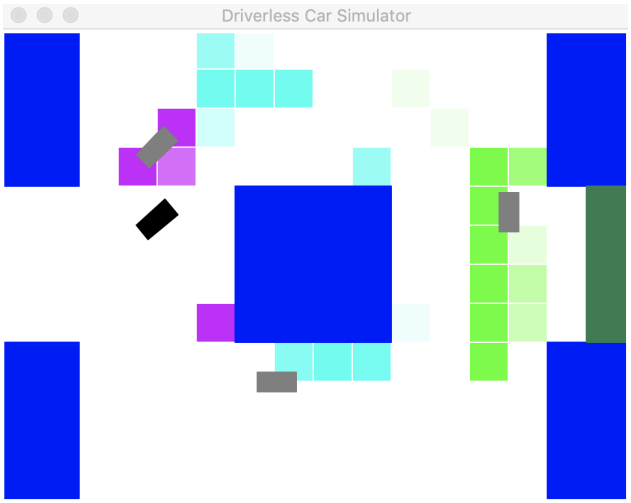

Fig. 2 Small Environment in Autonomous Driving Simulation. The agent vehicle is in black. Stationary Obstacles are in blue, true positions of dynamic obstacles in light gray, belief states of dynamic obstacles in green, purple, and cyan. The deeper the color of the belief state the higher the probability of there being an obstacle in that state. The goal is in dark green.

Specifically, we modified the 2D simulation to include the height of obstacles, included state uncertainty due to sensor noise, and implemented collision avoidance with dynamic obstacles alongside online path planning, taking partial observability into account. We also implemented the contribution of several additional noisy sensors - specifically radar and Lidar. The world is modeled as a Hidden Markov Model, with each tile having a belief probability of having a dynamic obstacle, such as another vehicle. We decided to make use of the existing, robust codebase of this simulation, since it already shared several commonalities with our goal. Additionally, it had already included a form of graphical interface, which is useful for both debugging and illustration of our work. We particularly liked the visual representation of the agent's belief about the location of dynamic obstacles.

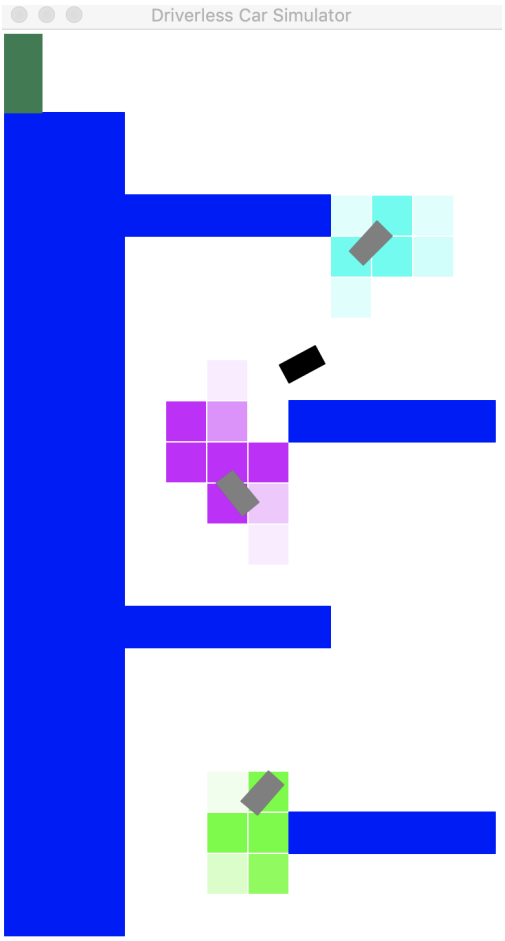

Fig. 3 Lombard Simulation Environment

B. Implementation

The existing codebase provides us with the ground truth of the positions of the agent, dynamic obstacles, and goal positions in a two-dimensional grid world. We first simulated observation of the noisy sensor data, and then created a belief probability of dynamic obstacles based on the sensor data fusion. Next, we updated the beliefs for all the states using particle filtering, and used a path planning algorithm with heuristics inspired by A*. We performed online planning from the current position, taking into account the probability of there being an obstacle in adjacent states.

1. Sensor Simulation and Fusion

There are many techniques available for sensor fusion in simulation environments [9] . Since we are given the ground truth of the position of the obstacles through the simulation codebase, we implemented simulations of radar and Lidar sensors by providing to the robot a randomly generated sample from a Gaussian distribution (corresponding to sensor noise) over the true distance to the obstacles. We then performed data fusion for both the radar and Lidar sensors by combining the probabilities of the noisy sensor data using Bayes' Rule [10]. Values for sensor noise were taken from leading real-world sensors, then slightly exaggerated to ensure that the agent would err on the side of low confidence. Absolute knowledge of the agent's motion relative to its past location was also provided to the agent. This is physically realistic, as the Doggo makes use of quadrature encoders and can use the offset alongside knowledge of motor ability to compute the physical distance traversed. When the radar and Lidar data were combined on the agent, the agent's confidence was very high, usually localizing other obstacles to within two tiles on the simulation.

2. Particle Filtering

We built up a distribution of beliefs about the probability of obstacles nearby the robot using particle filtering [11]. Rather than keeping a complete and exhaustive list of the belief state for every obstacle and every tile in the environment, particle filtering allows us to only keep a constant number of beliefs and updates them based on the importance of knowing the state of a certain tile. We first initialize a random number of particles over all the states, and then randomly update which states the particles will be assigned to based on a weighted probability built from observations [12]. We then use the number of particles in each state to create a belief distribution of whether there are any obstacles in any particular state, to be used for the next iteration of the algorithm. Changing the number of particles helps to identify a balance between knowledge and computational difficulty; with an increasingly high number of sensors, a lower number of particles was needed, as the locations of obstacles was localized with higher certainty.

3. Path Planning with Dynamic Obstacle Avoidance

We model the grid world as a graph, with each grid state as a node, and adjacent grids being connected with edges. First, we considered the A* algorithm for path planning. Initial attempts to use this algorithm resulted in the robot consistently undervaluing the importance of avoiding static obstacles; avoidance of dynamic obstacles was considered far more important. Since maintaining and update belief states for all static obstacles in reality is often computationally intractable, we changed focus to a path planning algorithm which takes advantage of more local data and belief states. Then, for every time step, we use the planning algorithm with a heuristic of "node distance to goal" in order to perform online path planning. The heuristic formula is based on the Manhattan distance to the goal node, and the weighted probability of there being an obstacle in the node. After receiving the belief vector of the obstacle probability for adjacent states, we avoid areas above a certain threshold by using the next best node. Best in this case would mean one that has the shortest distance to agent, and a belief below the obstacle probability threshold. If none of the adjacent nodes have a belief below the threshold, we choose the adjacent node with the lowest obstacle probability and re-plan from there for the next time step. Pseudocode for the path planning algorithm is given in Algorithm 1 (shown below).

Algorithm 1 Online Path Planning Algorithm

```
procedure OnlinePlanning(currentNode, currentBeliefs, graph)
    nextNodes = graph.getAdjacentNodes(currentNode)
    initialize pq = []                          ▷ Priority Queue, where lower values get higher priority
    for node ∈ nextNodes do
        heuristicVal = calculateHeuristic(n)
        pq.add((node, heuristicVal))
    end for
    return pq.pop()                             ▷ Returns the node with the highest priority
end procedure
```

4. 3D Jump-Capable Overlay

After testing the addition of a complete z-axis to the simulation, the results were not what the authors had hoped; the extra information and sensor noise "confused" the agent, rendering its path-planning subpar (nearly nonexistent) and its runtime extremely slow, due to the addition of another dimension to the search space. Upon further testing, it became apparent that the most important feature of this additional axis was simply whether it encoded a height the robot could not reach, or one it could. To incorporate this information into the model, an additional property was added to each tile instead, tracking whether the robot could traverse to that tile (and by which adjacent tiles) or not. The schematic is shown below in Figure 4.

This overlay proved to be far more computationally tractable than the addition of a complete z-axis, and was also more compatible with the noisy sensor data. A high threshold of confidence was required to set a given tile as "jumpable," which we believe reflects the real-world reluctance to damage expensive hardware. One limitation of the overlay is that it cannot be used to constrain or otherwise govern the motion of the dynamic obstacles, which is not reflective of reality. All obstacles were treated like infinitely tall columns, capable of crashing into the robot at any height.

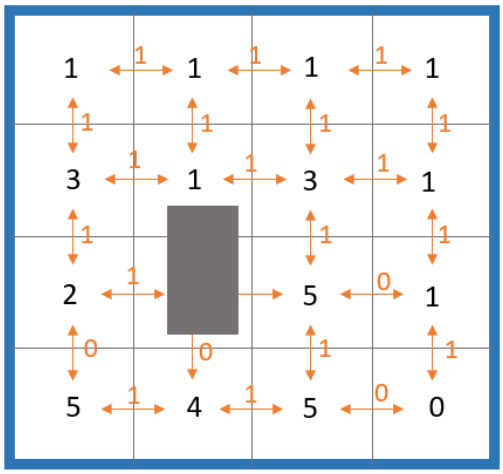

Fig. 4 Jump-Capable Overlay Schematic. In this image, the black numbers represent the ground-truth height of different areas, and the orange numbers represent the jump-capable overlay, where 0 indicates that the robot cannot traverse from one tile to another, and 1 indicates that it can. The jumping ability of the robot has been set to two for this example.

C. Codebase

Our Python code implementation can be found in the Github repository: www.github.com/PeggyYuchunWang/RobotPathPlanningProj .

III. Results

We compared our path planning algorithm to a baseline algorithm in two different simulation environments: the Small Environment (Figure 2) and the Lombard Environment (Figure 3). Our baseline algorithm consisted of a choosing a random next node and stopping at the current node if the belief of there being an obstacle in the next tile is above a certain threshold. As seen in Table 1, our path planning algorithm is better able to avoid the obstacles in the Lombard simulation environment and also is able to find a path to the goal faster than the baseline. The raw data for the results may be found in the Appendix section.

Name	Success Rate	Average Number of Successful Iterations
Small Baseline	60%	48.50
Small Planning Algorithm	60%	32.33
Lombard Baseline	40%	169.0
Lombard Planning Algorithm	50%	70.60

Table 1 Results. "Name" refers to the simulation environment (Small and Lombard) as well as the algorithm used (Baseline or our improved planning algorithm). "Success Rate" refers to the number of times the simulated vehicle is successfully able to reach the goal without colliding with obstacles. "Average Number of Successful Iterations" refers to the average number of timesteps needed to reach the goal if the vehicle was successful.

IV. Future Work

The most significant improvement that we would like to make to our simulation would be to incorporate state uncertainty into our model. Currently, the robot decides deterministically which tile it is in by calculating which tile the majority of its "body" is within. Incorporating state uncertainty would necessitate granting the robot a belief distribution of its location, either by using additional sensor data, such as GPS, or by intrinsically keeping track of actuated motion from a known starting position. Adding state uncertainty would help to account for the real-world lack of absolute

position knowledge, and make any path-planning paradigm a better reflection of the needs of the real robot. We also intend to make the simulation environment more realistic, by incorporating specific constraints into our model such as the presence of the ground, the robot's turning radius, and impediments to lateral motion. Then, we would like to extend our model to use additional sensors and compensate for uncertainties in the local terrain.

Another major area for future development is transferring algorithms designed and evaluated in simulation to the actual robot. This is an ongoing area of research, made more complicated by the variation between any two different robots, and the high cost of failure: if a robot uses an inadvertently unsafe algorithm, delicate and expensive hardware could be damaged by misactuation and crashes. Implementing an algorithm on the Stanford Doggo would require careful and rigorous testing of simple "sim2real" test cases, to assure all researchers that the final autonomous path planning algorithm would adequately tranfer to safe torques and modes of motion on the real robot.

V. Conclusion

This grid-based model is a useful tool in developing and testing autonomous modes of motion for jumping-enabled robots such as the Stanford Doggo quadruped. The model is moderately reflective of the robot's physical and navigational properties, although the true robot currently lacks most of the sensors currently used for path planning in this model (as the robot is not autonomous). The way which sensor fusion was implemented for the model supports the modular addition or removal of any other sensors, which is an important feature for any testing environment to have. The computational difficulty of path planning in three dimensions was largely avoided for this robot due to the "jump-capability" overlay; however, the limitations of only indicating whether the robot can traverse to a tile at a certain height are seen primarily in the descriptiveness of the model for the robot's environment. This could potentially limit any learned policy, as possibly-important height features of obstacles are not represented. Computational models such as this one can be an effective means of generating autonomous robotic navigation, especially for robots with unusual means of movement.

VI. Contributions

A. Peggy

- path planning, results, particle filtering, literature review

B. Caitlin

- sensor simulation and fusion, particle filtering, 3D jump-capable overlay

VII. Appendix

A. Small Baseline

Simulation Number	Result (Success/Fail)	Number of Iterations
1	Success	36
2	Success	88
3	Fail	35
4	Fail	19
5	Fail	27
6	Success	42
7	Success	49
8	Fail	190
9	Success	35
10	Success	41

B. Small Planning

Simulation Number	Result (Success/Fail)	Number of Iterations
1	Success	34
2	Fail	4
3	Success	33
4	Success	31
5	Fail	4
6	Fail	5
7	Success	30
8	Success	33
9	Fail	4
10	Success	33

C. Lombard Baseline

Simulation Number	Result (Success/Fail)	Number of Iterations
1	Fail	41
2	Fail	58
3	Fail	19
4	Success	91
5	Success	84
6	Fail	24
7	Fail	27
8	Fail	24
9	Success	109
10	Success	392

D. Lombard Planning

Simulation Number	Result (Success/Fail)	Number of Iterations
1	Fail	32
2	Fail	10
3	Success	67
4	Success	65
5	Success	67
6	Success	70
7	Fail	32
8	Fail	37
9	Success	84
10	Fail	11

References

[1] Blackmore, L., Li, H., and Williams, B., "A probabilistic approach to optimal robust path planning with obstacles," *2006 American Control Conference*, 2006, pp. 7 pp.–. doi:10.1109/ACC.2006.1656653.

[2] Fox, D., Burgard, W., and Thrun, S., "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, Vol. 4, No. 1, 1997, pp. 23–33.

[3] Sakai, A., Ingram, D., Dinius, J., Chawla, K., Raffin, A., and Paques, A., "PythonRobotics: a Python code collection of robotics algorithms," , 2018.

[4] Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J., "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments," *The International Journal of Robotics Research*, Vol. 29, No. 5, 2010, pp. 485–501. doi: 10.1177/0278364909359210, URL https://doi.org/10.1177/0278364909359210.

[5] Larson, J., Bruch, M., and Ebken, J., "Autonomous navigation and obstacle avoidance for unmanned surface vehicles," *Unmanned Systems Technology VIII*, Vol. 6230, International Society for Optics and Photonics, 2006, p. 623007.

[6] Thrun, S., Burgard, W., and Fox, D., *Probabilistic robotics*, MIT press, 2005.

[7] Zhu, Q., "Hidden Markov model for dynamic obstacle avoidance of mobile robot navigation," *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 3, 1991, pp. 390–397. doi:10.1109/70.88149.

[8] Kochenderfer, M. J., Amato, C., Chowdhary, G., How, J. P., Reynolds, H. J. D., Thornton, J. R., Torres-Carrasquillo, P. A., Üre, N. K., and Vian, J., *Decision Making Under Uncertainty: Theory and Application*, 1st ed., The MIT Press, 2015.

[9] Pohanka, P., Hrabovskỳ, J., and Fiedler, M., "Sensors simulation environment for sensor data fusion," *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, IEEE, 2011, pp. 1–8.

[10] Kubrak, D., Le Gland, F., He, L., and Oster, Y., "Multi—sensor fusion for localization. Concept and simulation results," *Proceedings of the 2009 ION Conference on Global Navigation Satellite Systems, Savannah 2009*, 2009, pp. 767–777.

[11] Thrun, S., "Particle filters in robotics," *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 2002, pp. 511–518.

[12] Fearnhead, P., Papaspiliopoulos, O., Roberts, G. O., and Stuart, A., "Random-weight particle filtering of continuous time processes," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, Vol. 72, No. 4, 2010, pp. 497–512.