Machine Learning and Artificial Intelligence for Robotics
Assignment: 1

Boston Cleek

October 28, 2019

Problem 1

A matrix was used to represent the grid. Each element in the matrix represents a cell given the cell size. The value of an element is 1 for unoccupied cells and 1000 for occupied cells. The location of a landmark in the world was converted into grid coordinates by determining which cell it should fall into based on the domains of each cell.

Problem 2

The naive and online implementations of A* use the Octile distance heuristic Eq. (1). On a square grid the heuristic considers eight directions of movement [1]. Given the position of the current node (x_n, y_n) and the goal (x_g, y_g) the heuristic is calculated as follows.

$$h(n) = D_1\,(dx + dy) + (D_2 - 2\,D_1)\,min(dx, dy) \tag{1}$$

Where dx and dy are the absolute differences in the horizontal and vertical positions between the node and the goal, $dx = \mid x_n - x_g \mid$ and $dy = \mid y_n - y_g \mid$. D_1 is the distance to an unoccupied node (north, south, east, or west) and D_2 is the diagonal distance to an unoccupied node in this paper ($D_1 = 1$ and $D_2 = \sqrt{2}$).

Let $\gamma = D_2 - 2\,D_1$ and the true cost $\Lambda = D1\,(dx + dy)$, if $\gamma < 0$ the heuristic will not over-estimate the true cost because Λ will be reduced by $\gamma\,min(dx, dy)$. Therefore, the Octile distance heuristic is admissible.

A grid with a starting position of (1,1) and a goal of (3,5), the true cost is 6. Based on Eq. (1) the value of the heuristic is 4.828.

Problem 3

The naive algorithm has knowledge of the obstacles *a priori* and considers adding up to 8 potential neighboring nodes to the open list. A few conditions must be met. First, the neighboring nodes must be within the bounds of the gird. If a node at the same position as the neighbor's position in the grid is already in the open list, the true cost g(n) of both nodes is compared, where $g(n)$ is the cost from the starting node to the neighbor node. If the neighbor's true cost is greater than the node in the open list then the neighbor is not added. If the neighbor's true cost is less than the cost of the node in the list, the cost of the node already in the open list is updated to the neighbor's lower cost. If a node at the same position as the neighbor's position in the grid is already in the closed list, the neighbor's node is not added to the open list. A min heap was implemented for finding the node in the open list with the minimum total cost f(n), where $f(n) = g(n) + h(n)$. Using the naive implementation of A* a path was planned between the start and end goals bellow with a cell size 1x1 m.

Path	Start	Goal
A	[0.5, -1.5]	[0.5, 1.5]
B	[4.5, 3.5]	[4.5, -1.5]
C	[-0.5, 5.5]	[1.5, -3.5]

Table 1: Path configurations for cell size of 1x1 m

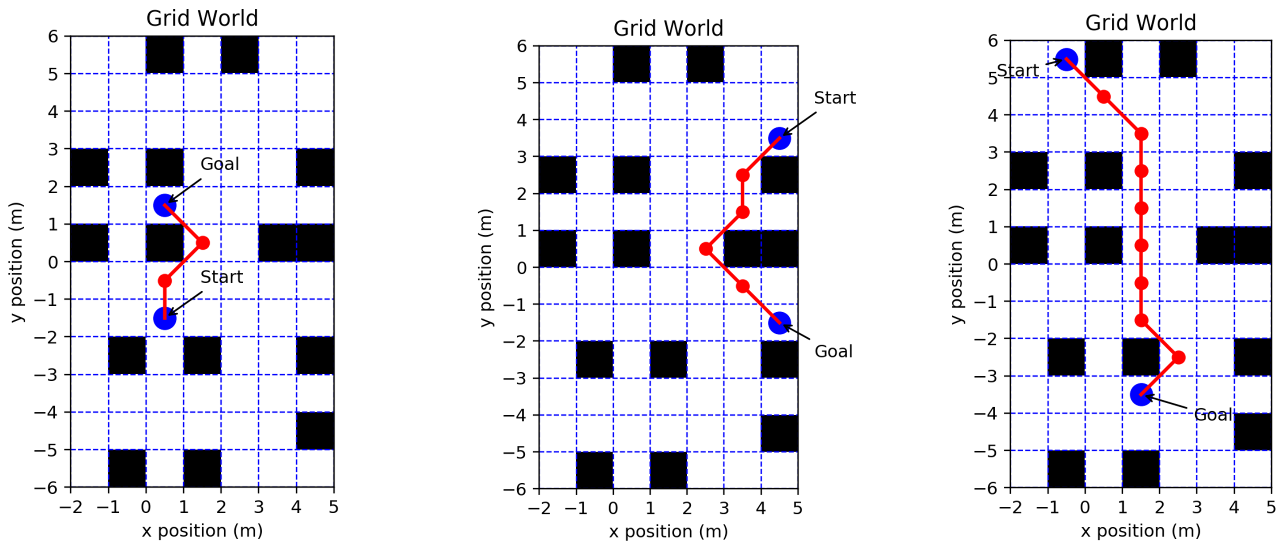

Figure 1: Paths A, B, and C from Table (1) are left, middle, and right respectively where obstacles are black, unoccupied cells are white, the path planned by naive A* is red, and the start/end positions are blue

Problem 4

The online implementation of A* does not have knowledge of the obstacles *a priori* and must re-plan from the current node when an obstacle is encountered. The online algorithm determines the costs of all the neighboring nodes, up to eight neighbors. If a neighboring node is already on the closed list, the neighboring node is ignored. The neighboring node with the smallest total cost f(n) is selected and placed on the open list. The open list will only have one node at a time in contrast to the naive implementation that will likely have an open list with more than one node.

Problem 5

Using the online implementation of A*, a path was planned between the start and end goals bellow with a cell size 1x1 m.

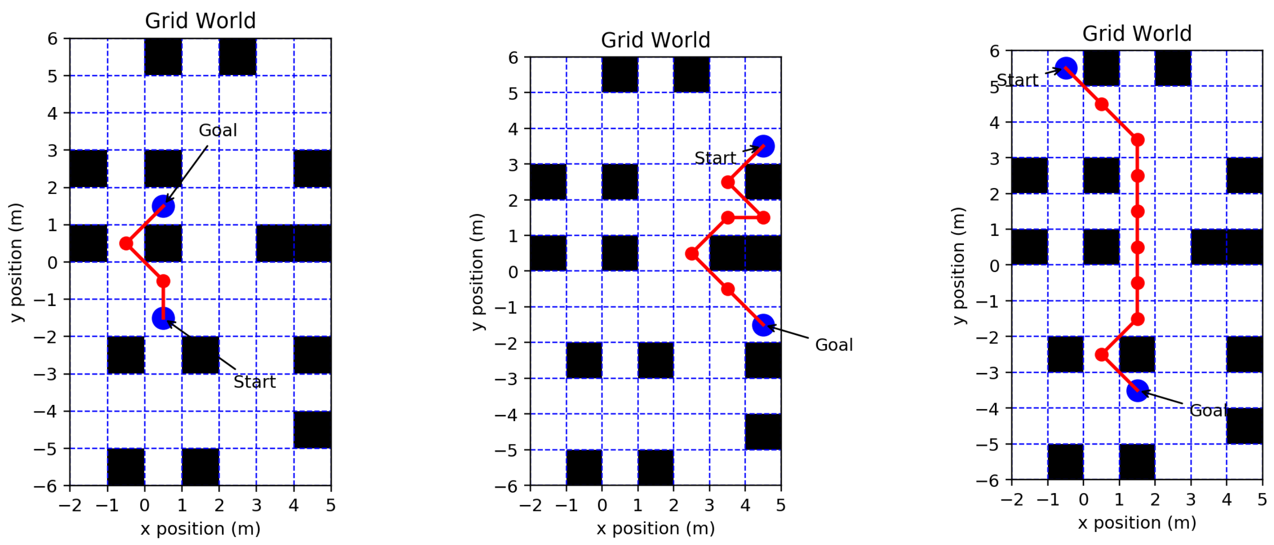

Figure 2: Paths A, B, and C from Table (1) are left, middle, and right respectively where obstacles are black, unoccupied cells are white, the path planned by online A* is red, and the start/end positions are blue

There are slight differences between the paths in Fig. (1) and Fig. (2). For path A, the naive algorithm chose to go to the right of the obstacle before reaching the goal and the online algorithm went to the left. The same outcome occurred for path C. In paths A and C, the grid cells to the left and to the right of the obstacle just before the goal have the same total cost $f(n)$. When two neighboring cells have the same cost the online algorithm may chose a different grid cell than the naive algorithm. This behavior is a result of the order in which the neighboring cells are observed. However, for path B, the online algorithm chose to go to the right, hitting a wall, in Fig (2) compared to traveling directly down in Fig (1). This is a result of the online algorithm planning from the current grid cell because obstacles are not known *a priori*.

Problem 6

For the .1x.1 m cell size, the footprint of each obstacle was increased by 0.3 m in each direction for all landmarks. The corresponding grid coordinates for each landmark were then labeled as occupied.

Problem 7

The following paths use a cell size of .1x.1m and the path is planned using the online implementation of A* described in *Problem 4*. Planning online using a fine grid is fast despite the small grid cell size because the open list contains 1 target cell each iteration. The advantages and disadvantages of fine grids are discussed in further detail in *Problem 11*.

Path	Start	Goal
A	[2.45, -3.55]	[0.95, -1.55]
B	[4.95, -.05]	[2.45, 0.25]
C	[-0.55, 1.45]	[1.95, 3.95]

Table 2: Path configurations with cell size of .1x.1m

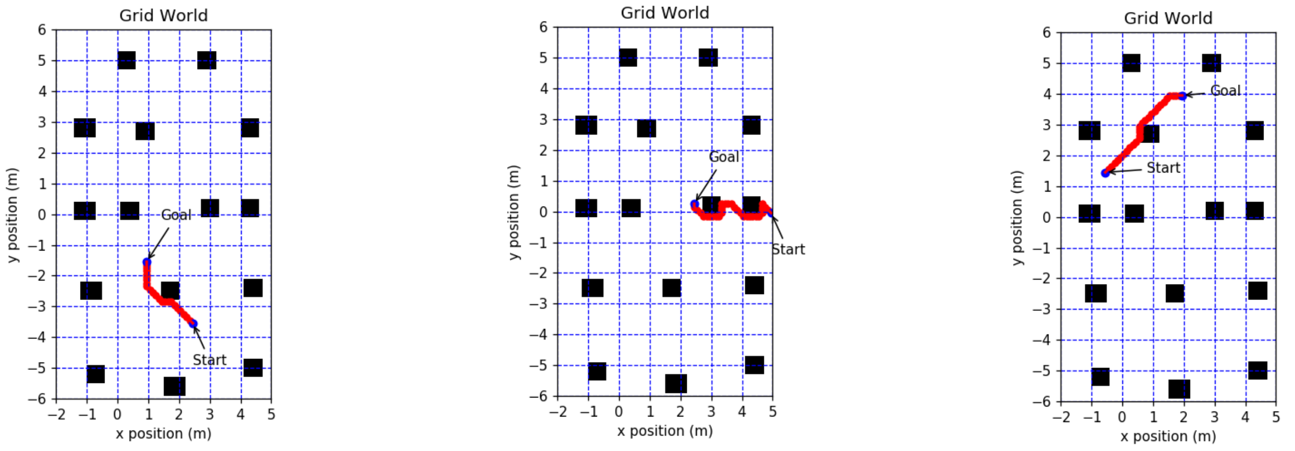

Figure 3: Paths A, B, and C from Table (2) are left, middle, and right respectively where obstacles are black, unoccupied cells are white, the path planned by online A* is red, and the start/end positions are blue

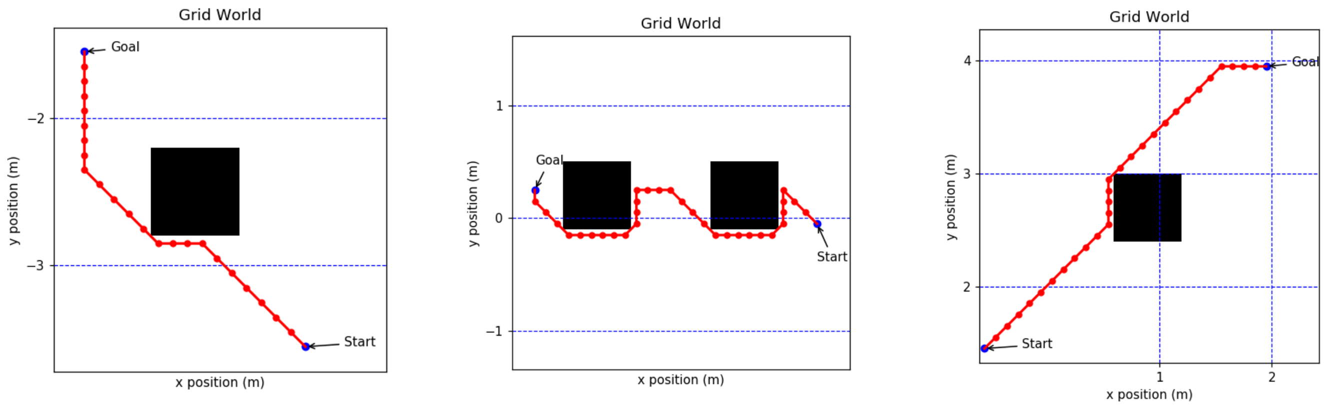

Figure 4: Zoomed in views of paths A, B, and C from Fig. (3) are left, middle, and right respectively where obstacles are black, unoccupied cells are white, the path planned by online A* is red, and the start/end positions are blue

Problem 8

The inverse kinematic controller models the robot as a kinematic unicycle, where u_1 and u_2 are the linear and angular velocity control inputs. In the frame of the robot, the positive x-direction is forward and the positive y-direction is left. Positive angular velocity and angular position is considered counter clockwise. The kinematics are propagated forward using a *fourth order Runge-Kutta* integrator with a time step $dt = 0.1s$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} u_1 cos(\theta) \\ u_1 sin(\theta) \\ u_2 \end{bmatrix} \tag{2}$$

Given the pose of the robot (x, y, θ) and a target position $(x_T,\ y_T)$ the nominal controls u_1 and u_2 are determined using the distance to the target and the error in the robot's heading towards the target. Both controls are issued at the same time allowing the robot to turn and move towards the target simultaneously.

$$u_1 = k_1 \sqrt{(x_T - x)^2 + (y_T - y)^2} + \varepsilon_v \tag{3}$$

$$u_2 = k_2(\arctan 2(\frac{y_T - y}{x_T - x}) - \theta) + \varepsilon_\alpha \tag{4}$$

Where k_1 and k_2 are the feedback gains, ε_v and ε_α are random perturbations in the nominal control inputs. The robot's pose is now probabilistic because of the random perturbations. The random perturbations are drawn from normal Gaussian distributions with means equal to zero, $\varepsilon_v \sim \mathcal{N}(0,\, \sigma_v^2)$ and $\varepsilon_\alpha \sim \mathcal{N}(0,\, \sigma_\alpha^2)$. The controller uses the following parameters ($k_1 = 0.5$, $k_2 = 2$, $\sigma_v = 0.01$, $\sigma_\alpha = 0.0875$).

The values for the parameters σ_v and σ_α assume the robot's linear and angular velocity can be modulated within a standard deviation of $1\,\frac{cm}{s}$ and $0.0875\,\frac{rad}{s} 5\,deg$. These values are realistic for a differential drive mobile robot such as the Jackal or Husky from Clearpath Robotics.

The values for the feedback gains were determined experimentally. If the gains are too high, the robot will over shoot the desired target and if they are too low, it will take a longer time to reach the target cell. The gain k_2 must be greater than k_1. If the robot's linear velocity is too high, the robot will have difficulty turning towards the target. Initially, the gains were set low and then incrementally increased until many oscillations in the robot's pose were observed. The gains were then slowly decreased until the oscillations decreased and the robot could achieve the target cell.

The robot has the following acceleration limits $\dot{v}_{max} = 0.228\,\frac{m}{s^2}$ and $\dot{\omega}_{max} = 5.579\,\frac{rad}{s}$. To ensure the robot does not exceed the acceleration limits, the nominal controls calculated in Eq. (3) and Eq. (4) are not issued yet. Based on the kinematics Eq. (2) the linear velocity of the robot is the control u_1 and the angular velocity is the the control u_2. The linear and angular accelerations of the robot are then computed using *forward finite differences.*

$$a_l = \frac{u_{1_t} - u_{1_{t-1}}}{dt} \tag{5}$$

$$a_\alpha = \frac{u_{2_t} - u_{2_{t-1}}}{dt} \tag{6}$$

Where a_l and a_α are the tangential and angular accelerations, the robot will experience these accelerations if the nominal controls are issued. The variables u_{1_t} and u_{2_t} are the current nominal linear and angular controls, $u_{1_{t-1}}$ and $u_{2_{t-1}}$ were the controls issued at the previous time step. If the accelerations calculated in Eq. (5) and Eq. (6) are greater than the acceleration limits, the controls issued are given bellow.

$$u_1 = u_{1_{t-1}} + \dot{v}_{max}\, dt \tag{7}$$

$$u_2 = u_{2_{t-1}} + \dot{\omega}_{max}\, dt \tag{8}$$

The control update in Eq. (7) and Eq. (8) ensures the acceleration limits will not be exceeded because the max accelerations are considered during the update. The direction of acceleration needs to be considered in Eq. (7) and Eq. (8). If an acceleration from either Eq. (5) or Eq. (6) is negative, the corresponding limit used either $\dot{v}_{max}$ or $\dot{\omega}_{max}$ should be negative in either Eq. (7) or Eq. (8). Random perturbations are not applied in Eq. (7) and Eq. (8) because there is no guarantee that the acceleration limits will not be exceeded.

Problem 9

All visualization in *Problems 9, 10, and 11* use the following color scheme: the start/end configurations are blue, the planned path using A* is red, the position of the robot is purple, the heading of the robot is depicted by a yellow arrow, all obstacles are black, and all unoccupied cells are white.

The paths planned by online A* using the path configurations in Table (2) were driven using the kinematic controller. The robot always starts with the speeds ($v = 0\, \frac{m}{s}$ and $= 0\, \frac{rad}{s}$) at the position of the start with a heading of $\theta = -pi/2$. At each pose update from the *Runge-Kutta 4* integrator, noise was added to each of the states.

$$\begin{aligned}
x_t &= x_t + \varepsilon_x, & \varepsilon_x &\sim \mathcal{N}(0,\, \sigma_x^2)\\
y_t &= y_t + \varepsilon_y, & \varepsilon_y &\sim \mathcal{N}(0,\, \sigma_y^2)\\
\theta_t &= \theta_t + \varepsilon_\theta, & \varepsilon_\theta &\sim \mathcal{N}(0,\, \sigma_\theta^2)
\end{aligned}$$

The noise was drawn from a normal distribution with a mean of zero and the following standard deviations ($\sigma_x = 0.02$, $\sigma_y = 0.02$, and $\sigma_\theta = 0.0875$). The standard deviations for x and y positions correspond to 2 cm and the standard deviation for angular position is $\approx 5\, deg$. The values are realistic for the motion model because smaller values are unlikely to capture stochasticity in the environment.

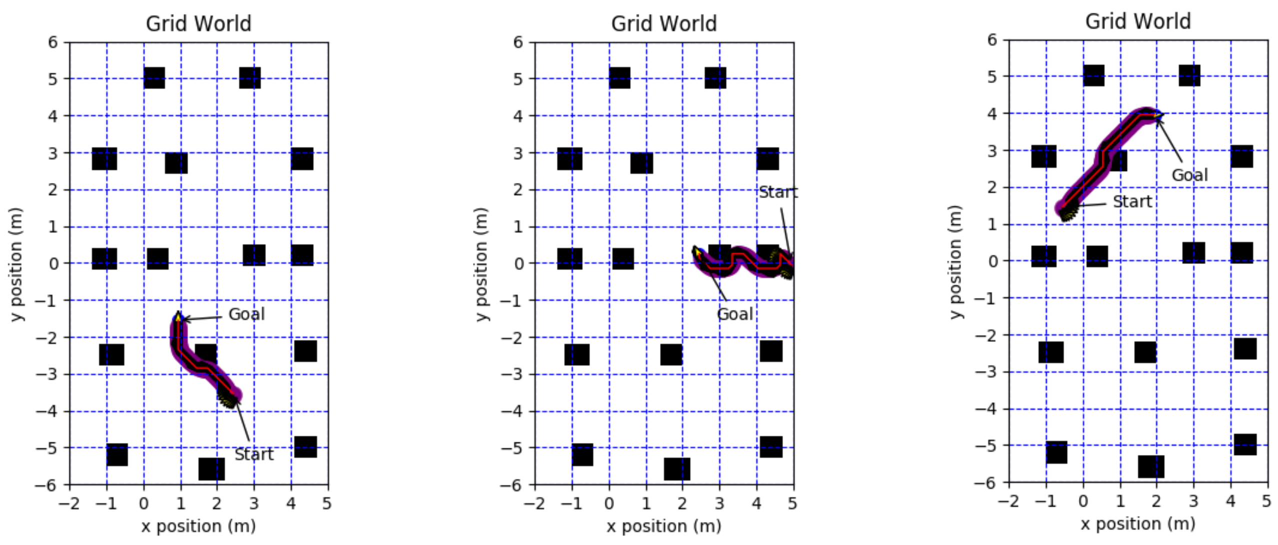

Figure 5: Driving pre-planned paths A, B, and C from Table (2) are left, middle, and right respectively

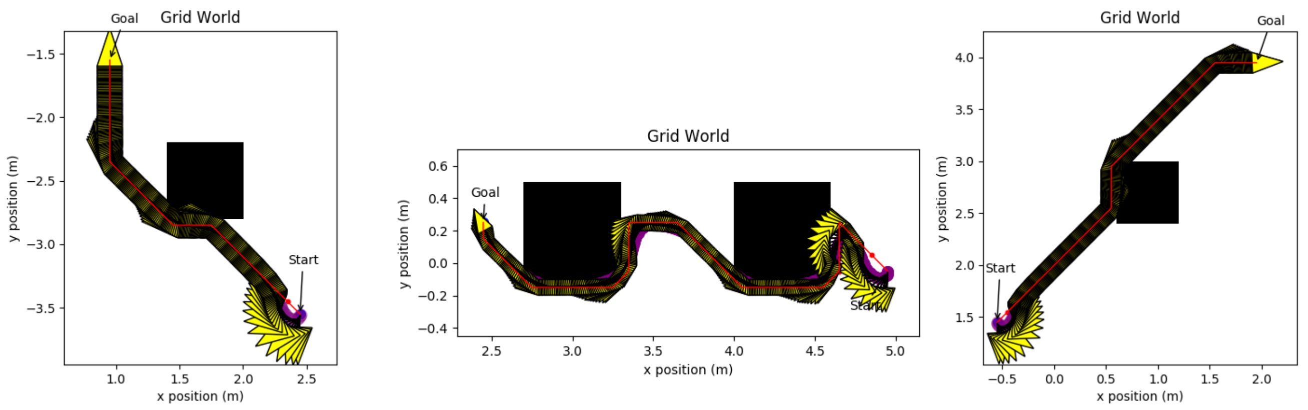

Figure 6: Zoomed in views of paths A, B, and C from Fig. (5) are left, middle, and right respectively

It may look as though the robot is driving through obstacles. The overlap is due to the size of the markers for the robot. When the robot gets within a distance of 0.15 m of the target cell, it then attempts to drive to the next target cell. The kinematic controller described in *Problem 8* consistently drives the robot from start to goal for all path configurations. Issues were encountered when the feedback gains became too high or when the linear velocity gain k_1 became large compared to the angular velocity gain K_2. The high gains caused the robot to overshoot target cells resulting in oscillations. Even when the noise added to the controls and to the pose was increased, the robot successfully traversed the paths. Issues were encountered when attempting to avoid the first obstacle in path B. In the presence of high noise ($\sigma_x = 0.2$, $\sigma_y = 0.2$, and $\sigma_\theta = 0.175$) the robot would collide with the first obstacle and then oscillate while trying to navigate around it.

Problem 10

When the robot is planning while driving, all planning is performed from the robot's current position. If the robot ends up in a cell other than the target cell, A* must select a new target cell. The robot's kinematics and noise added in the controller influence the path planning component.

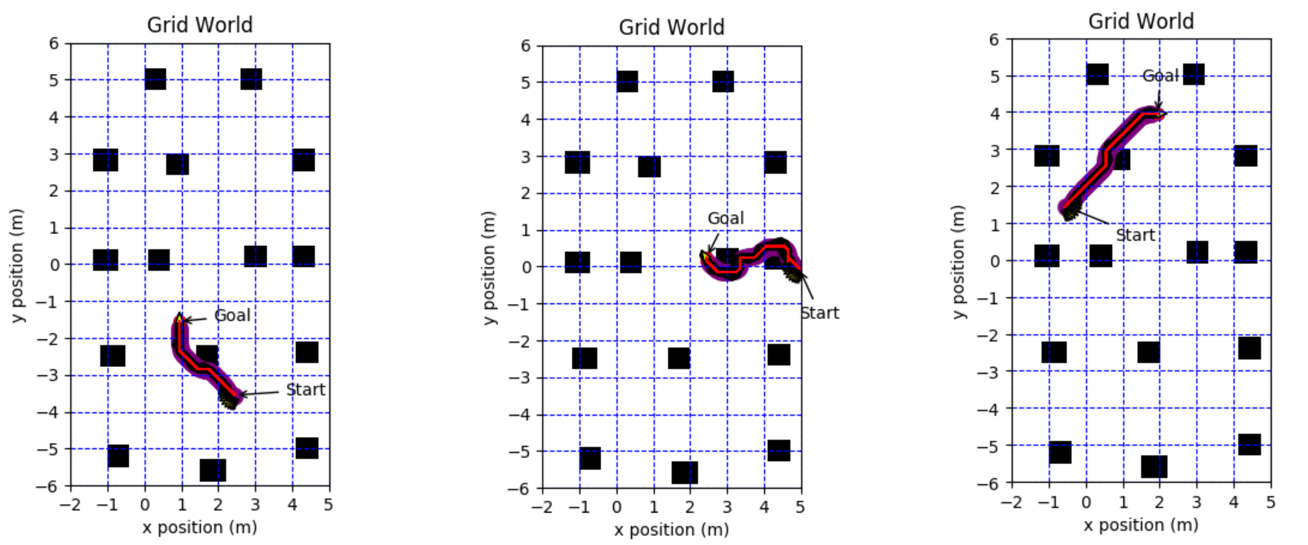

Figure 7: Planning while driving paths A, B, and C from Table (2) are left, middle, and right respectively

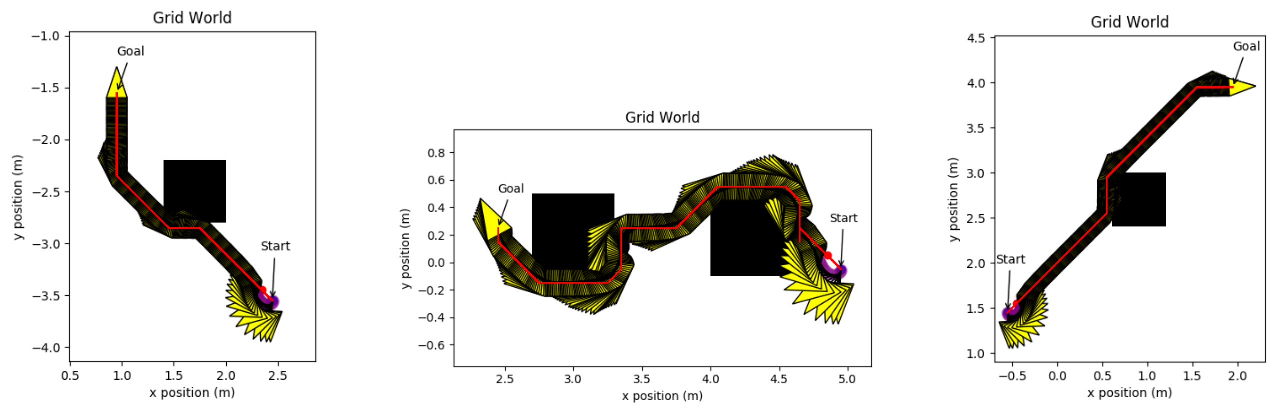

Figure 8: Zoomed in views of paths A, B, and C from Fig. (7) are left, middle, and right respectively

The paths in Fig. (7) appear similar to those in Fig. (5). The differences are seen in the zoomed in view in Fig. (8). When the robot is planning from its location, it experiences a divergence from the expected path when negotiating obstacles. The most apparent difference is in path B in Fig. (8) compare to Fig. (6). The robot struggled to circumvent the first obstacle, throwing it off from the expected path in Fig. (5). When the noise was increased ($\sigma_x = 0.2$, $\sigma_y = 0.2$, and $\sigma_\theta = 0.175$) the robot took significantly longer to make it past the first obstacle. When the noise was lowered ($\sigma_x = 0.005$, $\sigma_y = 0.005$, and $\sigma_\theta = 0.00175$) path B in Fig (8) was similar to Fig. (6).

Problem 11

Planning and driving on a coarse grid takes less time then the fine grid. Short simulation time is especially advantageous if the goal is to simulate many planned paths and observe whether the robot is successful. Such a setting could be in a stocahstic planner where thousands of attempts are made to plan a path and drive to a goal [2]. The trajectories can be scored based on how much they deviate from the goal defined by a cost function. The controls resulting in a successful trajectory are then issued to the robot. A stochastic planner effectively allows a robot to simulate thousands of paths before attempting to navigate in the world. The stochatic planner can be implemented in real time when the planning component is computation inexpensive (i.e on coarse grids), and when kinematics are simple. A stochastic planner implemented on a real robot was the Georgia Tech AutoRally car [2].

However, coarse grids fail to accurately capture the physical geometry of the world, which may be crucial to the path planning situation. A surgical robot attempting to navigate through tissue without damaging nerves or vessels will need a fine grid size. If a robot does not reach a target cell and needs to re-plan from its position, it will have more target cell options in a fine grid. For example, if the robot ends up in a corner, there is a greater chance A* can select a new target cell that is not on the closed list. This prevents the robot from getting stuck or backtracking to previously visited cells.

The vehicle dynamics are important when selecting a grid size. The trajectories in the coarse grid are not as smooth as the fine grid. The robot is able to accomplish trajectories with sharp turns because it uses a differential drive. If a robot has ackerman steering geometry, it will not be able to make sharp turns because it will require forward momentum to turn. A few examples are a semi-truck navigating streets or a passenger car that should maintain a smooth ride. The planning algorithm in these examples should take advantage of the smoother trajectories that result from a fine grid.

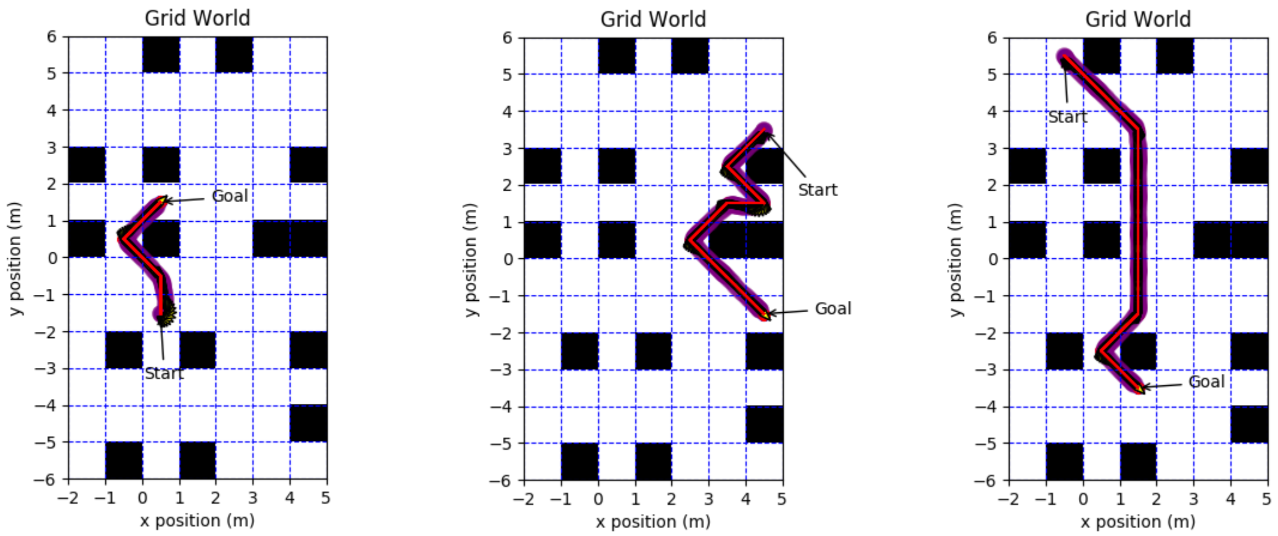

Figure 9: Planning while driving paths A, B, and C from Table. (1) are left, middle, and right respectively with grid size 1x1 m

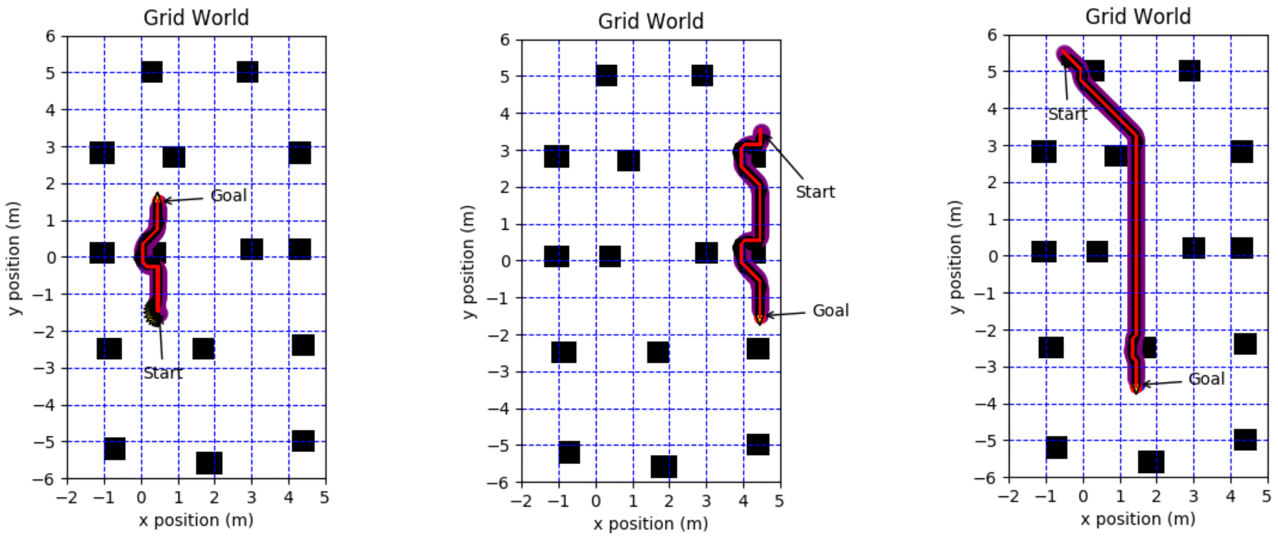

Figure 10: Planning while driving paths A, B, and C from Table. (1) are left, middle, and right respectively with grid size .1x.1 m

Problem 12

If a robot is to navigate the physical world with high fidelity, the motion model must capture dynamics. The lateral and longitudinal frictional forces on the wheels need to be considered. These forces are the result of tire deformation during acceleration. Models such as the linear tire model [3] or the brush tire model [4] aim to estimate these forces by modeling properties of the tires such as the longitudinal and lateral tire stiffness. The suspension model will influence the drive performance. Suspension models typically include a combination of springs, dampers, and masses with properties such as the spring or damping constant.

The kinematic model assumes the robot is a point mass and does not consider the geometry or the inertia of the robot. The robot has inertia and this should be included in the dynamics model along with the inertia for important components such as the wheels and tires. The size of the robot needs to be inflated to account for the physical footprint just as the landmark's size was inflated for the fine grid. The robot may not encounter an obstacle head on. However, when the robot is in a cell next to the obstacle, collision checks ensuring parts of the robot do not clip the obstacle should be implemented.

The vehicle may need to traverse terrain that includes a variety of sizes of obstacle and hills. The planning and control algorithms need to be extended to capture 3 physical dimensions. Building 3D grids is possible using Octomap (probabilistic 3D mapping using octrees) but will likely require stereo cameras for modeling purposes and will be computationally more expensive than 2D mapping. The tire dynamics are especially important in the 3D setting because they will determine if the robot can roll over an obstacle. The type of terrain the robot is driving on also needs to be known. Driving in sand is much different than grass or on the street. The type of terrain will influence the potential paths as well as how aggressively or smoothly the robot can drive.

Optimal controllers are necessary for navigation in the world because of their ability to reject disturbances. The implementation of a model predictive controller with trajectory optimization between grid cells will improve the fidelity of the controller [5]. The starting pose and the target cell are the boundary conditions to a set of differential equations (the vehicle dynamics). The trajectory between the two points is solved by modeling the boundary value problem as a nonlinear optimization problem. Optimization techniques such as direct collocation can solve for optimal trajectories. Direct collocation uses polynomials or bezier curves to approximate a smooth transition between states and controls. A smooth trajectory between grid cells provides a robust reference for a model predictive controller to sample. The control signal from a model predictive controller is likely to have greater success driving the robot to a target than a kinematic controller. Model predictive controllers can exploit the vehicle dynamics producing a control signal that results in the vehicle drifting and jumping if desired.

References

[1] Theory.stanford.edu, "Heuristics," 2019. [Online]. Available: http://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html

[2] G. Williams, A. Aldrich, and E. Theodorou, "Model predictive path integral control using covariance variable importance sampling," *arXiv preprint arXiv:1509.01149*, 2015.

[3] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 1094–1099.

[4] J. Svendenius and B. Wittenmark, "Brush tire model with increased flexibility," in *2003 European Control Conference (ECC)*. IEEE, 2003, pp. 1863–1868.

[5] S. Aghli and C. Heckman, "Terrain aware model predictive controller for autonomous ground vehicles."