ML and AI for Robotics - Assignment:2

Senthil Palanisamy

October 28, 2019

A.1, A.2: Implementation of Astar Algorithm

The heuristic function that was used for the offline version of the algorithm is given below

$$\bar{h}(n) = \min(X_{diff}, Y_{diff}) + \mid X_{diff} - Y_{diff} \mid$$

X_{diff} is the absolute difference between the x coordinates of node and the goal. $X_{diff} = \mid x_n - x_g \mid$, where x_n and x_g refer to the x coordinates of the node n and goal respectively
Y_{diff} is the absolute difference between the y coordinates of node and the goal. $Y_{diff} = \mid y_n - y_g \mid$ where y_n and y_g refer to the y coordinates of the node n and goal respectively
Before presenting the proof of admissibility for this cost function, a short intuition behind the formulation of this cost function is presented. Heuristic functions can derived by relaxing some of the actual constraints in the problem and finding an expression for the cost of the goal from a node in this relaxed setting. To come up with such a relaxed setting, lets assume that our grid would has no obstacles. In such a scenario, the term $\min(X_{diff}, Y_{diff}) + \mid X_{diff} - Y_{diff} \mid$ represents the actual cost of reaching the goal from a given node. This can be understood by considering the figure below

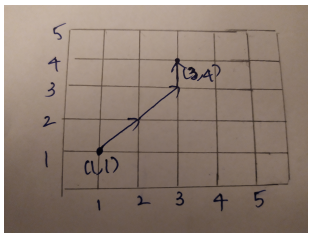


Figure 1: A simple grid world with no obstacles

In accordance with the problem definition of equal costs for the 8-connected neighbors, the cheapest cost from start point (1,1) to the goal point (3,4) is 3. It can be inferred that in finding the cheapest path, we try to maximize the number of diagonal connections and then traverse vertically or horizontally to reach the remaining distance. The term $\min(X_{diff}, Y_{diff})$ represents the maximum number of diagonal entries that can be traversed between the current node and goal node while the term $\mid X_{diff} - Y_{diff} \mid$ represents the remaining distance to be traveled after traversing all diagonal connections. Thus the function $\bar{h}(n) = \min(X_{diff}, Y_{diff}) + \mid X_{diff} - Y_{diff} \mid$ represents the cost of the path to the goal from a given node in a hypothetical no obstacle world. It is easy to prove that this heuristic is admissible id we look at the following inequality

$$\bar{h}(n) = \bar{C}(n) <= C(n)$$

where $\bar{C}(n)$ is the cost of the path to the goal from the node when there are no obstacles and C(n) represents the true cost. This inequality follows from the basic logic that placing an obstacles in the middle of a cheapest cost path can at best leave the cost unaltered or increase its overall cost. This is

purely due to the optimal substructure property of the cheapest path i.e., if a subpath lies on the cheapest path, then the subpath is by itself the cheapest path between those two nodes.

$$\implies \bar{h}(n) <= C(n)$$

Therefore, this heuristic is admissible. It is worth noting that the efficiency of any heuristic lies in how closely it lies to the true cost. In accordance with this fact, it can be seen that

$$SLD(n) <= \bar{h}(n) <= C(n)$$

$$Chebyshev(n) <= \bar{h}(n) <= C(n)$$

It can be verified that the cost returned by this heuristic is always higher than other commonly used heuristics like Short Line Distance (SLD) and Chebyshev distance. Since this heuristic has higher cost than other commonly used heuristic function, it should be more efficient and should lead to faster path finding than the other heuristics presented above. It can also be emphasized that unless some prior knowledge on obstacles is assumed, there cannot be a better heuristic for offline path planning than the one presented here since this heuristic function equals the cost of the path to the goal from a node in an obstacle free world.

The A star algorithm uses the heuristic along with the node transition cost to evaluate the cost of each neighboring node. All the neighboring nodes of the current node are added to the open list and the node that the robot has currently visited is added to the closed list. In each step of the algorithm, the cheapest node from the open list is picked up and all its neighbors are added to the open list, after ensuring that those neighbors have not been added to the closed list. The termination point of the algorithm is when the goal node is added to the closed list. This A star algorithm is both complete and optimal because it will always find a path to the goal (if it exists) and it finds the cheapest path that is available.

A.3: Results of applying A Star on graph

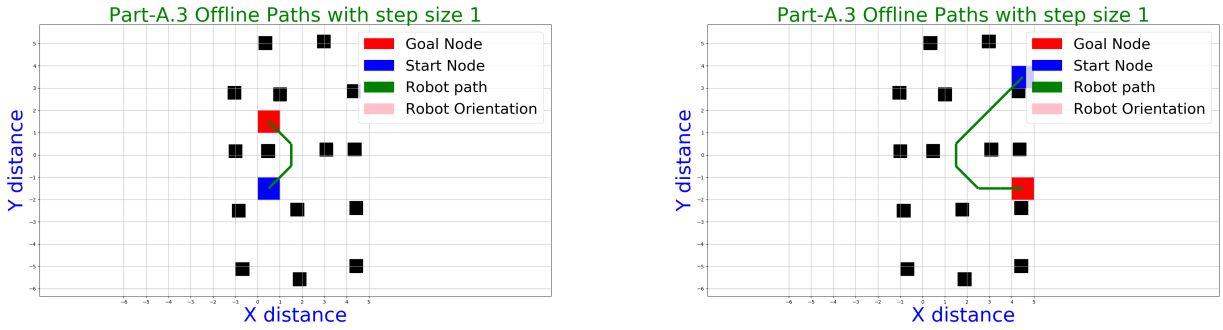


Figure 2: A Star results

It can be seen that the algorithm finds the most optimal path. One implication of the cost function that we have introduced is that the algorithm always prefers the diagonal paths more than vertical or horizontal paths since the heurisitic value associated with some diagonal paths is lower than vertical or horizontal paths. One of the core requirements that make this possible is that we have the view of the whole graph before hand and that the graph is static and does not change with time. Therefore, we can find the best path before making any physical movement with the robot and then follow the path blindly. One of the majors downsides of using AStar is the time and space complexity of the algorithm. The time and space complexity of plain AStar is $O(b^d)$, where b is average branching factor and d is the depth of the tree. The depth of the tree is dependent on the actual distance between the start node and end node, whereas

the branching factor is determined by the nature of the problem constraints and huerisitic chosen. This exponential complexity of Astar makes it impractical for real time application. To explain the nature of how unrealistic the time bound jumps up, experiments were run using different step sizes. From the table we can see that step sizes below 0.5 can be unrealistic for real time computation.

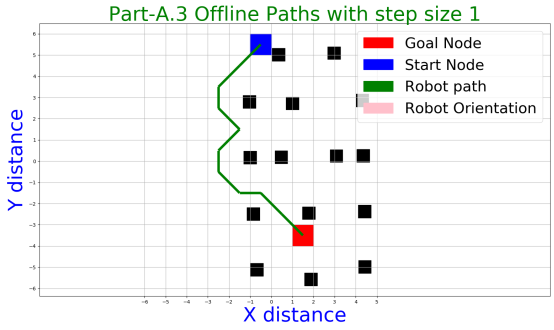


Table 1: AStar timing analysis

S.No	Step Size m	Time taken s
1	1	0.02
2	0.75	0.05
3	0.5	0.07
4	0.25	0.98
5	0.1	44

A.4,5: Online AStar and its results

The modification that has to made to make plain A star algorithm online were as follows
1. Open list was modified so as to include only immediate neighbors and not all neighbors of every node visited, thereby completely eliminating backtracking.
2. There was an interesting problem with the heuristic, which was rectified through a slight modification Point 1 is a direct consequence of the fact that backtracking is expensive in case of online algorithms since the robot has to physically move to the previous location and continue from there on. Due to this, the optimality and completeness property of AStar is destroyed but we make this conscious trade off to meet the real time requirements. Its easy to see how optimality is destroyed but to understand why the algorithm becomes incomplete,lets consider the example shown in Figure 3. It can be seen any reasonable heuristic chosen will guide the robot to make left and up movements. But when the robot makes the first three steps, we can see that both the open set and the closed set become empty sets at which point the algorithm terminates believing that there exists no path where as in reality there existed a path.

We can overcome the incomplete limitation of this algorithm by building a map of the world as we move along and allowing backtracking only in such exceptional scenarios but this has not been implemented in this work. Therefore, it is expected that the algorithm will falsely terminate without finding a path under some specific scenarios.

The heuristic $\bar{h}(n)$ is very nice for offline processing of paths but it encounters a small problem when we calculate paths online as shown Figure 3. It can be seen from the figure that when the robot is at node (1.5, 0.5) (shown in the figure in sky blue), both the nodes (1.5, -0.5) and (1.5, 1.5) (shown in the figure in yellow) have the same cost 3 as per $\bar{h}(n)$. Since in online mode, backtracking is avoided, once the robot arbitrarily breaks the tie by moving into the node (1.5, 1.5), it loses access to the cheapest path. In order to overcome this problem, a new huerisitic function was formulated.

$$h_1(n) = \sqrt{SLD * (\min(X_{diff}, Y_{diff}) + \mid X_{diff} - Y_{diff} \mid)}$$

where SLD is the short line distance between the node n and the goal position

The logic behind this heuristic can be understood by considering the same example. It can be seen that the problem discussed above can be avoided, if, for example we would have chosen (1.5, -0.5) instead of (1.5, 1.5). Intuitively, this can be summarized as moving towards the node that has a lower short line distance in case a tie occurs in $\bar{h}(n)$. Though, we could use a special function as a tie breaker for such scenarios, it is more desirable to have a heuristic function that handles this more generally. Such a function can be brought about by multiplying $\bar{h}(n)$ with SLD (Short Line Distance). To make it a admissible heuristic,

the positive square root of this product is used as opposed to the product itself, which is not admissible. It is easy to prove that this function is admissible. It has already been shown that $\bar{h}(n) <= C(n)$, where C(n) is the true cost. It can be inferred that $SLD <= \bar{h}(n) <= C(n)$, The SLD distance is equal to $\bar{h}(n)$ in the case of a purely diagonal navigation but smaller otherwise

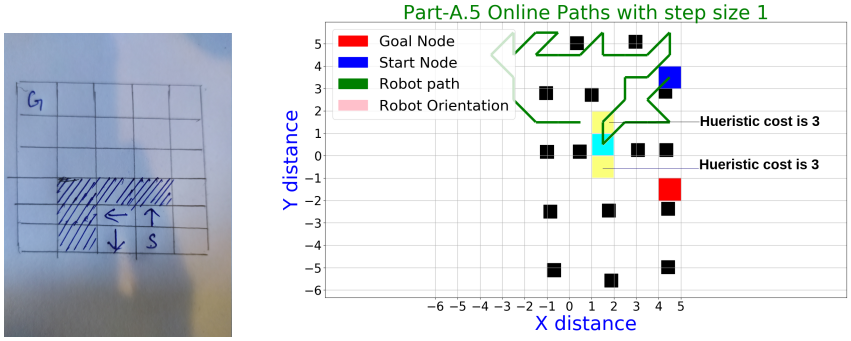


Figure 3: Problems encountered in online version of Astar algorithm

$$\implies \bar{h}(n) * SLD <= C(n) * C(n)$$

Taking Positive square root on both sides

$$\sqrt{\bar{h}(n) * SLD} <= C(n)$$

Hence, it is proved that this is an admissible heuristic. This function performs better for online version and is more efficient than the previous one (due to a better preference in the selection of nodes than $\bar{h}(n)$, which wasn't selecting right nodes when costs were tied)

Before closing this discussion on heuristic, it can be pointed out that a better huerisitic would have been $h_2(n) = \sqrt[3]{\bar{h}(n)^2 * SLD}$, since it can be seen that $h_1(n) <= h_2(n) <= C(n)$. Extending this logic, one step further we can discover a whole family of heuristic functions

$$h_k(n) = \sqrt[k+1]{\bar{h}(n)^k * SLD}$$

where it can be seen that $h_1(n) <= h_2(n) <= h_k(n) <= C(n)$ and it is pretty intuitive that as $k \to \infty$, it can seen that $h_k(n) \to \bar{h}(n) = \bar{C}(n)$, but with increased computation cost and improved performance since the closer the heuristic function is to the true cost, the smaller its branching factor and hence, better is its performance. In this assignment work, only $h_1(n)$ was used but it is expected that any other function with an increased k should yield better performance. The results of the online A star algorithm are shown in the Figure 4 and Figure 5

A.6, A.7: Online Astar with smaller grids

As we can see from Figure 4 and Figure 5 , the robot reaches the goal in all cases. As explained earlier, the path chosen is not the optimal one. Decreasing the grid size increases the resolution of robotic navigation but there is a limit to which this resolution can be enhanced depending on the physical constraints of the robot. For example, for a robot with 0.5 m/s of minimum linear speed and a command sampling rate of 0.5 s, the minimum distance that the robot can travel with a single command will be 0.25 m, which way past the graph resolution. Hence, the robot will consistently overshoot its target node when executing the navigation. Another constraint for example could be the turning radius of the robot. In this case for

example, when a robot is asked to move towards its diagonal neighbors, the implicit assumption is that the robot can turn 45 degrees within a distance of 0.1 meters. This may not be satisfied due to the physical design of the robot and hence the robot could consistently overshoot from its planned trajectory. Thus the optimum grid size is a clear trade off between movement resolution and physical constraints of the robot.

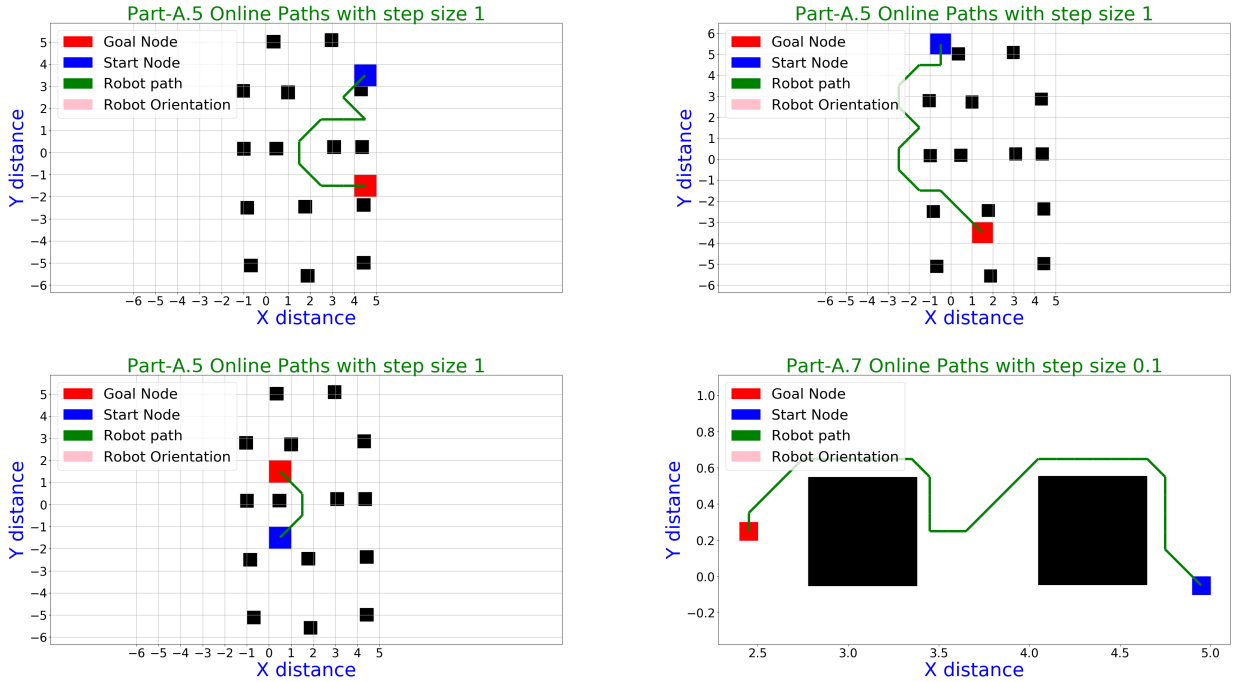


Figure 4: Results of Online Astar

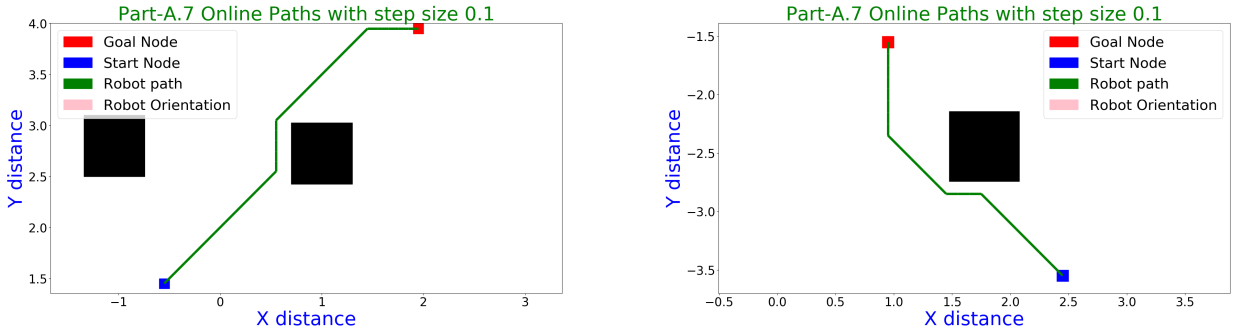


Figure 5: Results of Online Astar

It is also clear that small grid sizes restrict the acceleration of the robot. For example, if the robot were forced to follow a sequence of way points each 0.1m apart, it restricts the acceleration of the robot since it has to make a sequence of small moves. This can be clearly shown by one of the empirical observations when the robot took 91 steps to reach a node with 0.1 step size, which the robot was able to reach within 12 moves with a step size of 1.0. In the case of 1.0 step size, the robot has longer distance to travel between each commands and hence can utilize its acceleration fully and increase it velocity to achieve faster navigation whereas the robot will move at a very slow velocity if a very small step size like 0.1m is used. Therefore, the correct grid size will take into account the physical constraints of the robot, the acceleration limits and the navigation resolution needed.

A hybrid and more superior approach would be to use both coarse grids and fine grids: coarse grids

when we encounter large unoccupied spaces and switching to finer grids once we get close to the goal destination or when we encounter obstacles.

A.8: Controller Design

The motion model that has been assumed for this robot is the unicycle model. Its dynamics model is given by

$$\dot{x} = vcos(\theta)$$
$$\dot{y} = vsin(\theta)$$
$$\dot{\theta} = \omega$$

where $\dot{x}$ is the x velocity, $\dot{y}$ is the y velocity and $\dot{\theta}$ is the rate of rotation, which is the same as the angular velocity command given to the robot, v is the linear velocity command given to the robot and θ is the current orientation of the robot (here it refers to the yaw angle since the robot is restricted to moving in a planar motion)

A PID controller was used to control the heading angle of the robot while a P controller was used to control the robot's velocity. The equation of a PID controller is given as

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)d\tau + k_D \frac{de(t)}{dt}$$

Where k_P is the proportional term and adjusts the output signal proportional to its error
K_I is the integral term which accumulates error over time and adjust the output signal. This means that even if there is a very small error over a long period of time, this term will respond to that and correct our values to achieve perfect tracking. Since, the proportional term cannot get us precisely to the target value, the integral term is included so that we can get more precise. It must be noted that very high k_I values will can lead to oscillation in the system
k_D is the differential term and corrects the velocity values in response to sudden changes within the system. It must be noted that very high K_D values will make the system very sensitive to noise.

In our case, since the pid controller is used to control the robot's heading, the error term $e(t) = \theta_g - \theta(t)$, where θ_g is the goal orientation we wish to reach and $\theta(t)$ is the current orientation of the robot. Since we are dealing with a discrete case here, the intergral term $\int_0^t e(\tau)d\tau$ just reduces to the term $\sum_{i=0}^{i=t} e_i(t)\Delta_t$, where $e_i(t)$ refers to the error term at the time time step i and Δ_t refers to the time interval between successive velocity commands and the differential term $de(t)/dt$ just reduces to $\frac{e(t-1)-e(t)}{\Delta_t}$. Thus the expression for $\omega(t)$ can be written as

$$\overline{\omega}(t) = k_{wp}e(t) + k_{wi}\sum_{i=0}^{i=t} e_i(t)\Delta_t + k_{wd}\frac{e(t-1)-e(t)}{\Delta_t}$$

where k_{wp}, k_{wi}, k_{wd} are constants and their values for this application were tuned to be $k_{wp} = 0.75, k_{wi} = 0.75$ and $k_{wd} = 0.3$. It can be noted here that high values for k_{wi} makes the robot go in loops where as higher values for k_{wd} makes the robot very sensitive to noise and overcompensates for the noise in the system by adjusting $\theta(t)$ heavily. This is clearly illustrated in the figure given below.

The expression for linear velocity is given by, v=$\sqrt{\dot{x}^2 + \dot{y}^2}$. In discrete case, this can be rewritten as v=$\sqrt{(\frac{\Delta_x}{\Delta_t})^2 + (\frac{\Delta_y}{\Delta_t})^2}$ We use a proportional controller for our linear velocity, therefore the expression for the linear velocity term becomes

$$\overline{v}(t) = K_{vp} * \sqrt{(\frac{\Delta_x}{\Delta_t})^2 + (\frac{\Delta_y}{\Delta_t})^2}$$

The value of K_{vp} used in this implementation is 0.7

The equations look fine but the one important thing that hasn't been taken into consideration is the acceleration limits of the robots. The given acceleration limits are $a_v = 0.288m/s^2$ and $a_w = 5.579rad/s^2$. This implies that we cannot increase or decrease the linear velocity of the robot by more than $a_v\Delta_t$ and angular velocity by more than $a_w\Delta_t$ at any moment. Therefore, taking this into consideration

$$\omega_c = \min(|\ \overline{\omega}(t) - \omega(t-1)\ |, a_w\Delta_t) \qquad \text{and} \qquad v_c = \min(|\ \overline{v}(t) - \check{(}t-1)\ |, a_v\Delta_t)$$

And, finally,we have

$$\omega(t) = \begin{cases} -\omega_c & \text{if } \overline{\omega}(t) <= \omega(t) \\ \omega_c & \text{if } \overline{\omega}(t) > \omega(t) \end{cases} \qquad v(t) = \begin{cases} -v_c & \text{if } \overline{v}(t) <= v(t) \\ v_c & \text{if } \overline{v}(t) > v(t) \end{cases}$$

This just corresponds to increasing or decreasing the velocity by atmax acceleration times Δ_t. If the velocity change required is less than acceleration times Δ_t, then that update is carried out without any modifications. But if the velocity change required is more than acceleration times Δ_t, then the velocity change is capped at acceleration times Δ_t. The noise associated with the controller motion model is assumed to be a gaussian noise with variance of 0.01 (1cm) in case of x and y co-ordinates and 0.02 (1 degree) in case of robot's orientation

B.9 - Results discussion on the controller

The results of the applying the controller are shown in the figure below

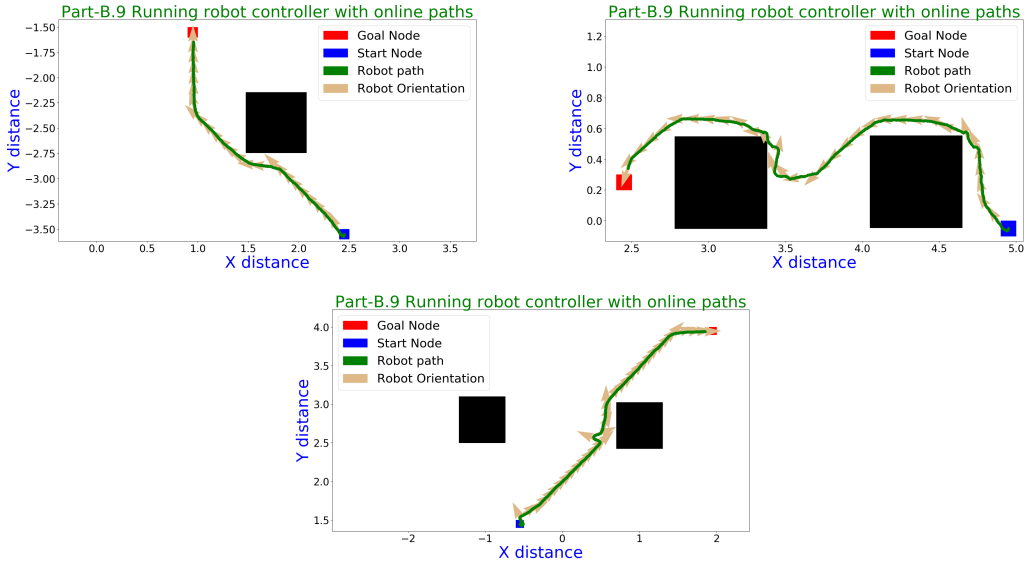


Figure 6: Using Controller to navigate paths generated by AStar

It can be seen that the robot follows the trajectory smoothly. However, these were the results of carefully tuned parameters values $k_{wp} = 0.75, k_{wi} = 0.75, k_{wd} = 0.3, k_{vp} = 0.7$. As explained in the controller design, values of $k_{wp} > 0.75$ causes the robot's orientation to oscillate and forces it to go in loops. This phenomenon is clearly observed in the Figure 8, where the robot starts looping in the highlighted area. Another experiment was run using $k_{wd} = 0.7$. This value makes the robot too sensitive to noise and disturbs the smooth motion of the robot. This phenomenon is clearly observed in the Figure 8, where the

robot does eventually reach its goal destination but it makes a lot of unnecessary rotations, in response to the noise in the sytem, where it ends up overcompensating due to the high k_d value.

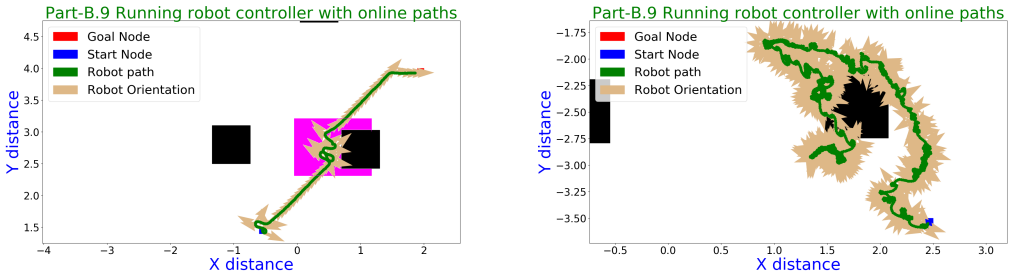
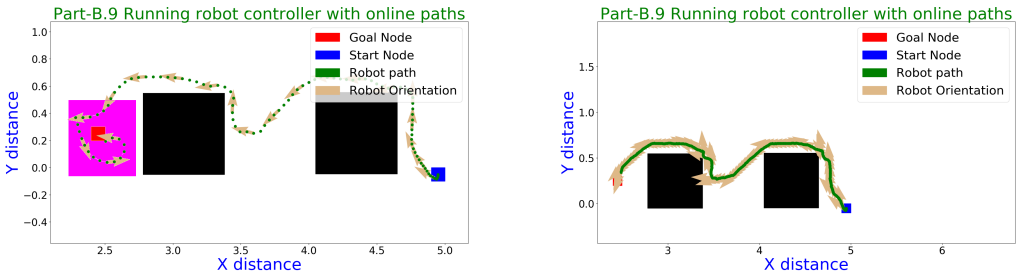


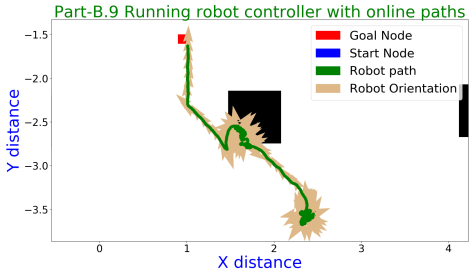
Figure 7: (On the left) Effect of using a high k_{wi} value(0.95) and (on the right) Effect of using a high k_{wd} value(0.7)

The values of k_{wp} and k_{vp} are little interdependent. An experiment was run using high k_{wp} value (0.85) and high k_{vp} (0.7) value. It can be seen from the figure that in such a setting, the robot overshoots from its trajectory sometimes. When the same k_{wp} (0.85) was used with a lower k_{vp} value (0.1), overshooting didn't happen. This observation is consistent with our logical notion that when the robot posses high linear and angular velocities, the robot loses its ability to take sharp turns and hence, the radius of curvature of the turning tends to be larger. It must also be noted that very high k_{wp} starts inducing oscillations regardless of its k_{vp}. The logical explanation behind this fact is that if the robot rotates too fast, the probability of hitting its intended orientation reduces and hence it starts rotating in place until it finds the target orientation.



(a) k_{wp} =0.85, k_{vp} =0.7

(b) k_{wp} =0.85, k_{vp} =0.1



(c) k_{wp} =0.95, k_{vp} =0.7

Figure 8: Effects of varying k_{wp} and k_{vp} values

Thus, based on the above observations, it is safe to hypothesise that there is a threshold for k_{wp} below which k_{vp} and k_{wp} can be adjusted inversely to achieve smooth navigation. This means that the ability of the robot to take sharp turns can be controlled by these two parameters. But beyond this threshold for k_{wp}, it becomes difficult for the robot to hit its intended orientation accurately. Though a rigorous mathematical proof has not been presented to validate this hypothesis, its seems a pretty intuitive to be

true based on empirical observations. The value of this threshold was found empirically to be 0.90, which means beyond when $k_{wp} > 0.9$, it is pretty difficult or nearly impossible to achieve smooth motion. But, below, this value, some combination of k_{wp} and k_{vp} wiil always exist to help us achieve smooth motion.

B.10-Planning while driving

The only change that is needed so that we can plan while we drive is to replace the path (or the sequence of nodes) that was obtained by running the Astar algorithm before staring the controller by the output of the motion model on the fly. Now the motion model tells the robot where it ended up after navigation and Astar access all the neighbors of the current position and gives the next node to follow. All the other aspects of algorithm construction remain the same as before. The results of applying this modification are shown below

It can be seen that the robot planning is no longer smooth since the robot does not land exactly where it plans to travel due to noise and hence, its path is altered periodically to offset the error due to noise. Previously when we used the path from the Astar algorithm that was processed before starting the controller, the sequence of nodes in the path offered very smooth directions to travel whereas now, the robot has to plan based on whereever it ended and hence, we could see frequent changes in its directions.

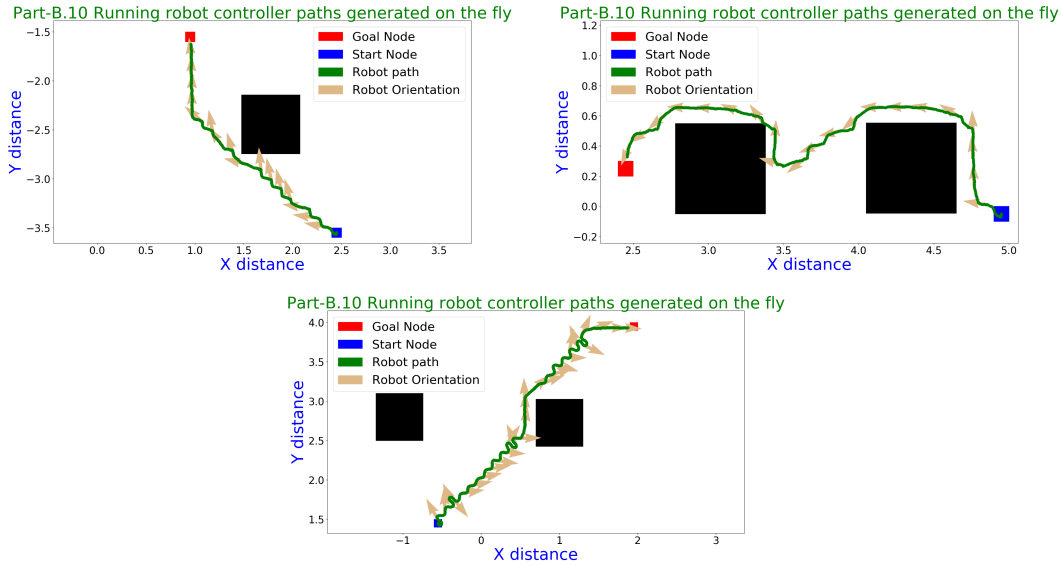


Figure 9: Planning while driving

B.11 - Coarse Vs Fine Grid

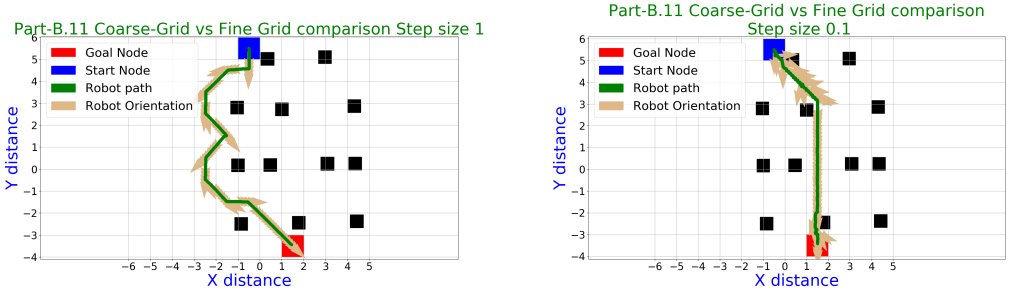


Figure 10: Coarse vs Fine grid comparision

Figure 10 and Figure 11 tell a clear story about the tradeoffs offered by grid sizes. It is very clear that smaller step sizes end up finding the shorter path because they can walk through smaller spaces inbetween obstacles which would otherwise show up as occupied nodes in case of a large step size but they don't offer the same smoothness that a larger grid offers. The figures on the right (smaller step size) clearly don't have the same level of path smoothness that is existing in larger grids.

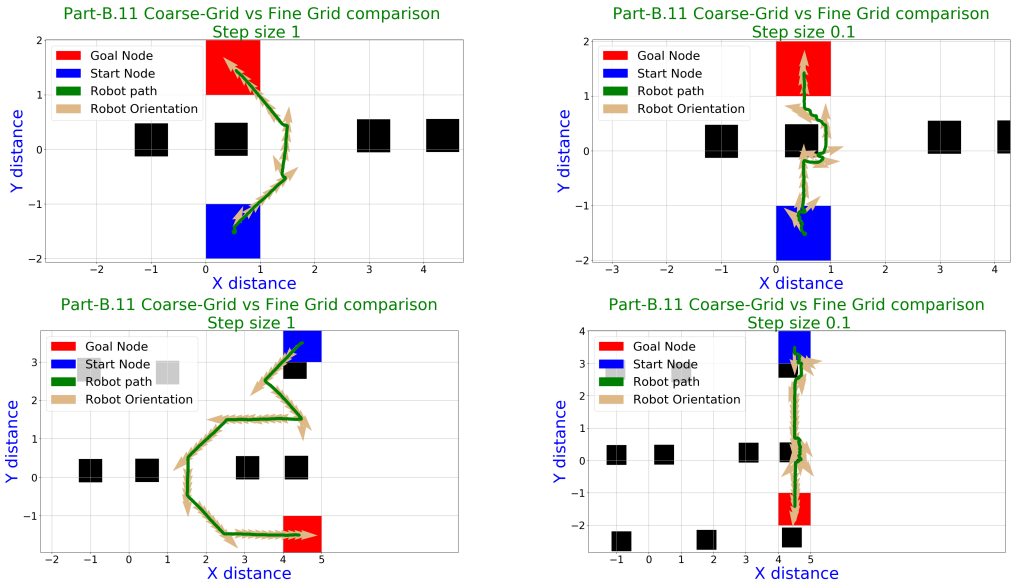


Figure 11: Coarse vs Fine grid comparision

B.12 -Simplifications made by our simulated controller

Some of the simplifications made by our simulated controller are as follows

1. **Absolute certainity about transition**: Though noise is introduced into the motion model to account for the fact that transitions are not fully deterministic, after transition we assume that the robot has absolute certainity about its present state. This is definitely not true. Depending on the application, this could involve using filters to localise the robot with sensor readings and known landmarks or in more complicated cases involve algorithms like SLAM for simultaneously doing the mapping as well as localisation

2. **Simplification due to unicycle model**: For simplicity, a unicycle model is assumed in this assignment. A real robot would involve a lot more controls than just v and ω. For example: A differential drive robot would actually involve 4 different controls, $v_1, v_2, \omega_1, \omega_2$ and its design would also require the knowledge of some details of the robot hardware like the distance between the wheels and radius of each. Though it is fairly straighword to map a unicycle model controls to a differential drive controls to achieve the same motion, a mapping is not always straighforward and in some case may force us to use a more complicated controller with lot more controls.

3. **Simplified action model of the robot**: In this problem, the action set of the robot at any point is only 8. But in reality, the robot has the ability to make more than just 8 transition. Depending on the turning resolution of the robot, the number of states that the robot could transition into can be a lot more than just 8. Though this does not stop this simplistic model to establish navigation between two random points, it can be argued that the efficiency of navigation can be improved a lot more if more transitions are accounted for.

4.**Incomplete Astar algorithm** - It has already been pointed out earlier that the online version of this algorithm is incomplete. Hence, we will need to map the world as we move along to allow backtransitions to take place to make this algorithm complete.