# Homework 1 16782: Planning and Decision Making in Robotics

Harsh Dhruva
Catch me if you can

October 5, 2021

## 1 Method

### 1.1 Pre-Compute a Goal

The approach I took pre-computes a goal position, which is a cell in the target's trajectory that the robot can intercept the target in time, also considering the cost of getting to the cell and waiting there. To select the particular cell, a forward Dijkstra search is run with the initial Robot position as the starting position. This gives us the cost to get to every point in the given trajectory of the target. We also get the time steps or distance it takes for the robot to get to every trajectory point. Then selecting the cells that the robot can reach in time, I sort the potential goal cells in a priority queue, by the minimum of the cost of getting to the cell (calculated using Forward Dijkstra) and the cost of waiting in the cell. Once sorted, the minimum cost cell is selected as the Goal position that is input into 2D A-star. Pseudo-code for the method is shown in Algorithm 1.

```
Data: $unordered\_map < int, int >$HEURISTIC_DF,
      $unordered\_map < int, int >$HEURISTIC_DF, $priority\_queue$ TARGET_G_VAL
Result: $goalPair < int, int >$
Forward Dijkstra populating $HEURISTIC\_DF$ with starting position as Robot Start
  Position;
$time\_buffer \leftarrow 1$ ;          /* Time buffer considering pre-computation time */
for $int\ i;\ i < target\_steps;\ ++i$ do
    $h\_val \leftarrow HEURISTIC\_DF[target\_index]$;
    $time \leftarrow ROBOT\_STEPS[target\_index]$;
    if $time + time\_buffer \leq$ then
        $g\_val \leftarrow h\_val + (i - time)$ ;        /* Cost of getting there + waiting */
        $TARGET\_G\_VAL.push(cell)$;
    end
    $goalPair = TARGET\_G\_VAL.top()$;
    return $goalPair$;
end
```

**Algorithm 1:** Pre-compute Goal Position

## 1.2 Heuristic

A 2D Backward Dijkstra search is performed with the starting position set to the goal position to get the Heuristic value for the A-star search. This is stored in an unordered map HEURISTIC_DB, with the index of the Cell, and the corresponding heuristic value. The pseudo-code is shown in 2. The forward Dijkstra used to pre-compute the goal follows a very similar pseudo-code, but also keeps track of the time or steps for the robot, and has the initial robot position as the start Cell. Here the h_values of each Cell are the heuristic values for A-star in the next section, and in technicality, they are the G-values in the Dijkstra itself. This is both an admissible and consistent heuristic to use.

```
Data: unordered_map < int, int >EXPANDED_DB,
      priority_queue < Cell, vector < Cell >, minimum h val sort > OPEN_DB
Result: unordered_map < int, int >HEURISTIC_DB
Cell target = Cell(goalposeX, goalposeY);
target.h_value ← 0;
HEURISTIC_DB[target_idx] = 0;
OPEN_DB.push(target);
while !OPEN_DB.empty() do
    Cell s = OPEN_DB.top();
    OPEN_DB.pop();
    if EXPANDED_DB[s_idx] == True then
        continue
    end
    EXPANDED_DB[s_idx] = True;
    HEURISTIC_DB[s_idx] = s.h_value;
    for int dir; dir < NUMOFDIRS; ++dir do
        Cell neighbor_cell = Cell(neighborx, neighbory);
        if Valid Neighbor Cell and Free then
            new_cost = s.h_value + map[neighbor_idx];
            if neighbor_cell.h_value > new_cost then
                HEURISTIC_DB[neighbor_idx] = new_cost;
                neighbor_cell.h_value = new_cost;
                OPEN_DB.push(neighbor_cell);
            end
        end
    end
end
```

**Algorithm 2:** Backward Dijkstra Heuristic Values

## 1.3 2D Weighted A-star

Once the goal position is computed, I use 2D Weighted A-star to compute the path from the Robot position to the goal position. The weight value epsilon is set to 100 for all the results using 2D A-star. Backtracking is also done within the aStar() function, which then populates the path the robot will take in a global variable. Each time planner() is called, the next position of the path is set to the action pointer. The psuedo-code is seen in Algorithm 3. I have also attempted an implementation of 3D A-star, which is a function in the code, however it is not being used for the planner. 2D a star is implemented in void aStar(..)

```
Data: $unordered\_map < int, int >$CLOSED, $unordered\_map < int, int >$PARENT,
      $priority\_queue < Cell, vector < Cell >, minimum\ f\ val\ sort >$ OPEN
Result: $vector < int >\ astarPath$
$Cell\ start\_cell\ =\ Cell(robotposeX, robotposeY)$;
$start\_cell.g\_value \leftarrow 0$;
$start\_cell.f\_value \leftarrow epsilon * Heuristic$;
$OPEN.push(start\_cell)$;
$PARENT[start\_idx]\ =\ -1$;
while $!OPEN.empty()$ do
    $Cell\ best\ =\ OPEN.top()$;
    $OPEN.pop()$;
    if $CLOSED[best\_idx] == True$ then
        continue
    end
    $CLOSED[best\_idx] = True$;
    $Break\ if\ best\ cell\ is\ goal\ cell$;
    for $int\ dir;\ dir < NUMOFDIRS;\ ++dir$ do
        $Cell\ neighbor\_cell\ =\ Cell(neighborx, neighbory)$;
        if $Valid\ Neighbor\ Cell\ and\ Free$ then
            $new\_cost\ =\ best.g\_value\ +\ map[neighbor\_idx]$;
            if $neighbor\_cell.g\_value > new\_cost$ then
                $neighbor\_cell.g\_value = new\_cost$;
                $neighbor\_cell.f\_value = neighbor\_cell.g\_value\ +\ epsilon * Heuristic$;
                $OPEN.push(neighbor\_cell)$;
                $PARENT[neighbor\_idx]\ =\ best_idx$ ;        /* Setting parent of
                 neighbor */
            end
        end
    end
end
$current\_idx\ =\ Goal\ Index$
while $current\_idx\ != start\_idx$ do
    $aStarPath.push\_back(current\_idx)$;
    $current\_idx\ =\ PARENT[current\_idx]$;
end
```

**Algorithm 3:** 2D - Astar

## 2 Results

The planner was able to catch the target in Map 1 to 4. System information on which I run the code:

Operating System:

Ubuntu 18.04.6 LTS

CPU

Intel(R) Core(TM) i9-10980HK CPU @ 2.40GHz
1 physical processor; 8 cores; 16 threads

RAM

32.696784 GB

## 2.1 Map 1

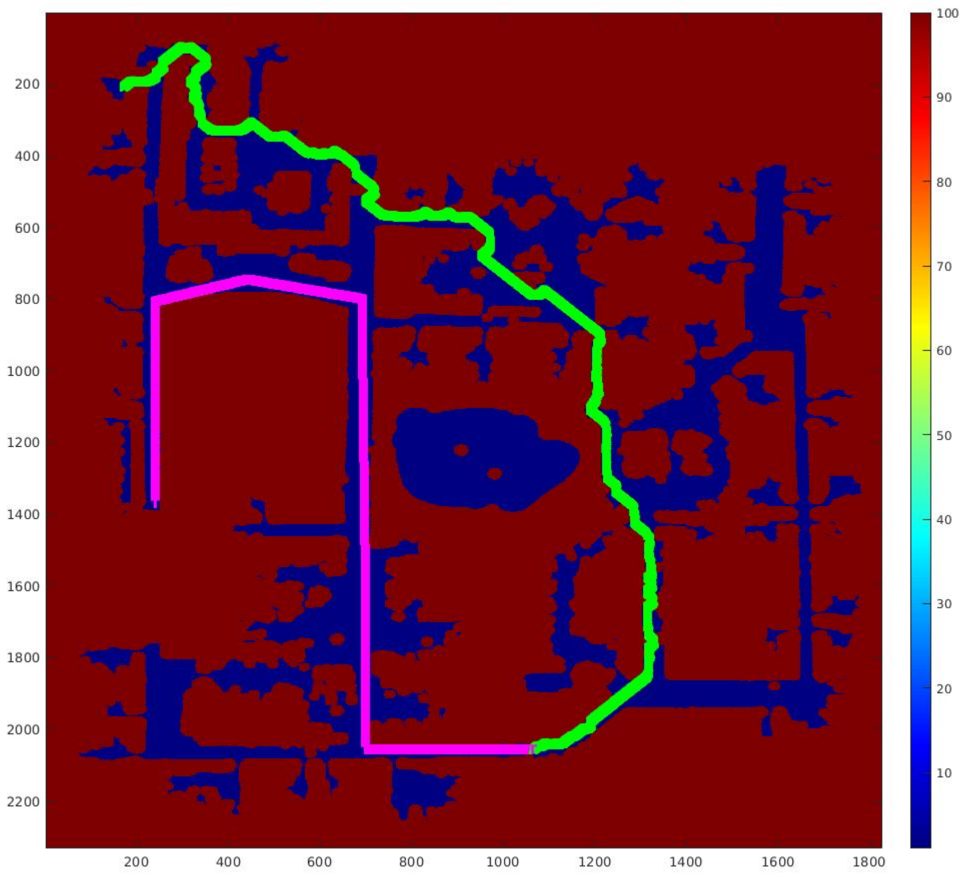

Figure 1: Map 1 Visualization

| Method | Heuristic | target caught | time taken(s) | moves made | path cost | pre-compute time (s) |
|---|---|---|---|---|---|---|
| 2D-Astar | 2D-Dijkstra | 1 | 2641 | 2639 | 2641 | 0.996967 |

Table 1: Map 1 Results

## 2.2 Map 2

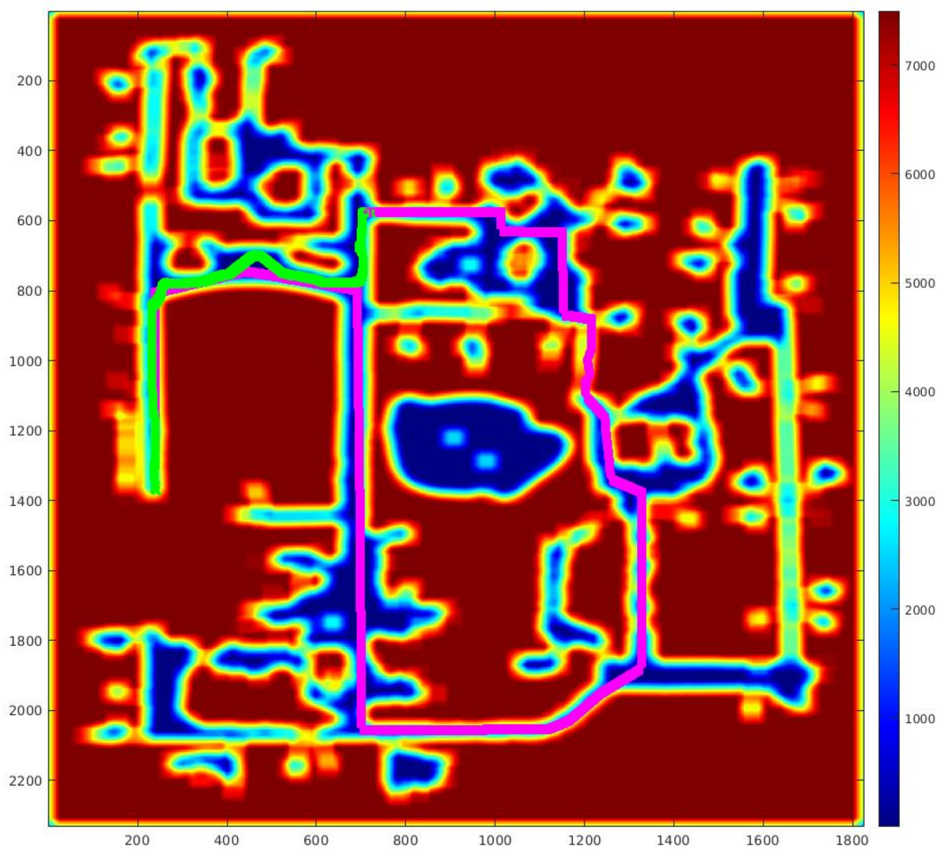

Figure 2: Map 2 Visualization

| Method | Heuristic | target caught | time taken(s) | moves made | path cost | pre-compute time (s) |
|---|---|---|---|---|---|---|
| 2D-Astar | 2D-Dijkstra | 1 | 4673 | 1235 | 4457697 | 2.70191 |

Table 2: Map 2 Results

### 2.3 Map 3

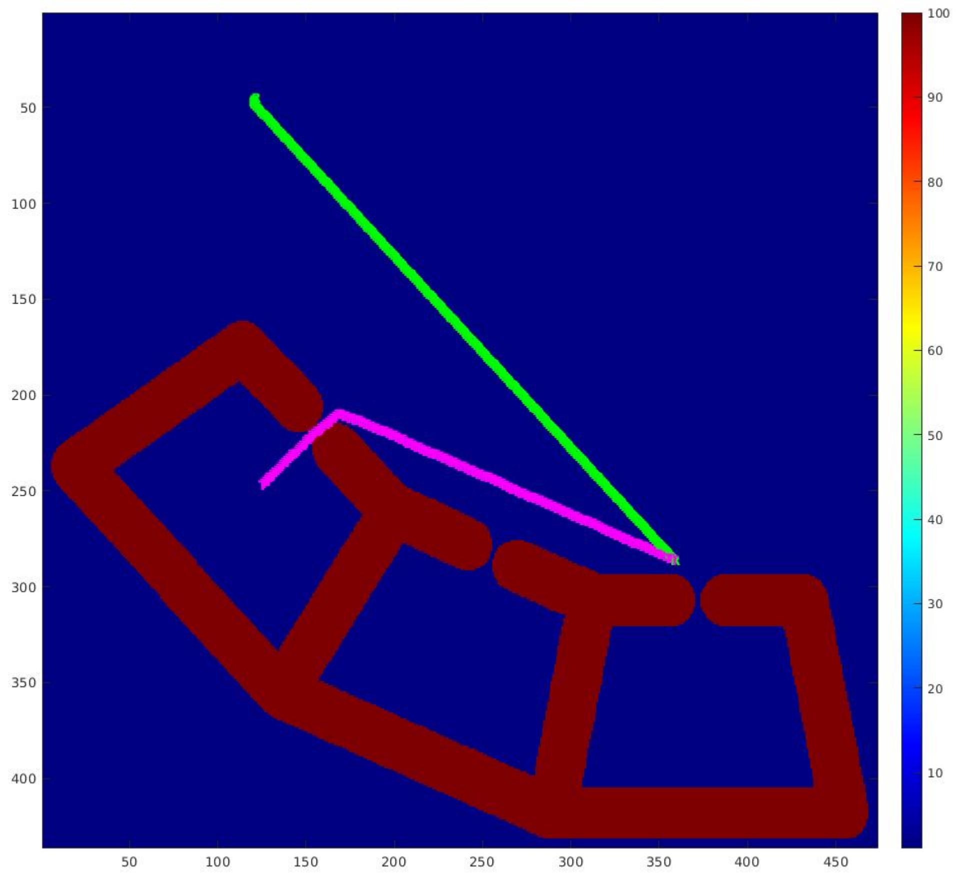

Figure 3: Map 3 Visualization

| Method | Heuristic | target caught | time taken(s) | moves made | path cost | pre-compute time (s) |
|---|---|---|---|---|---|---|
| 2D-Astar | 2D-Dijkstra | 1 | 242 | 241 | 242 | 0.214428 |

Table 3: Map 3 Results

## 2.4 Map 4

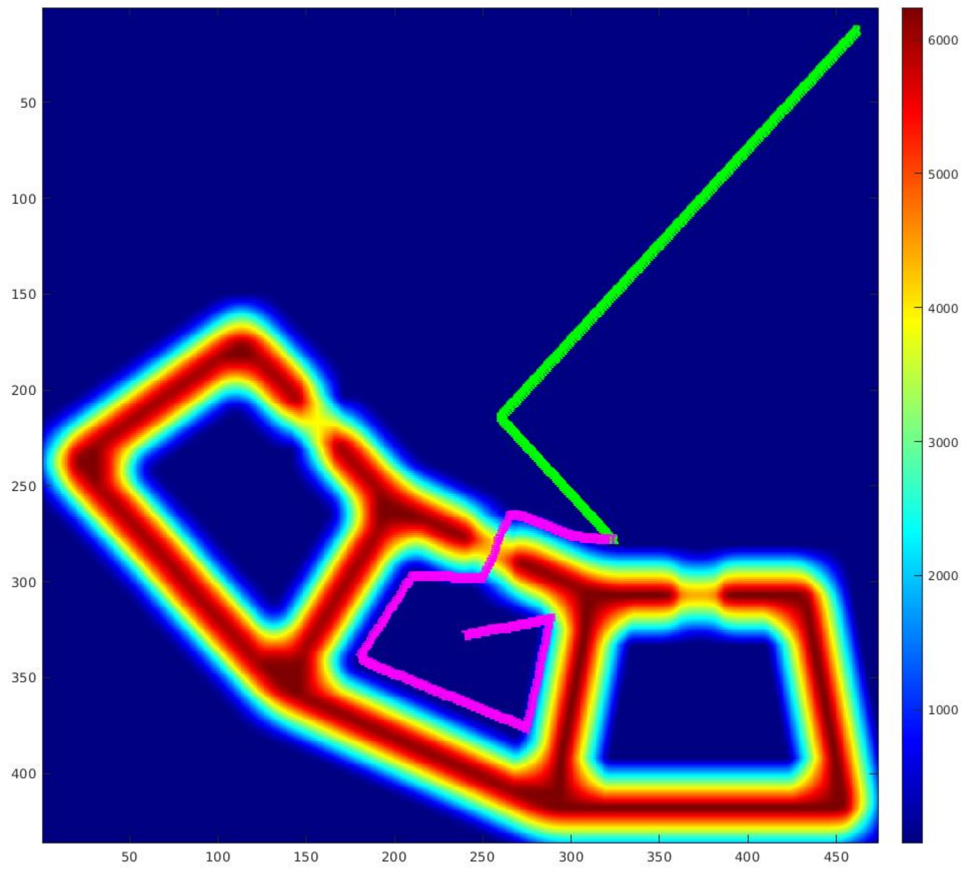

Figure 4: Map 4 Visualization

| Method | Heuristic | target caught | time taken(s) | moves made | path cost | pre-compute time (s) |
|---|---|---|---|---|---|---|
| 2D-Astar | 2D-Dijkstra | 1 | 380 | 266 | 380 | 0.254418 |

Table 4: Map 4 Results

## 2.5 Map 5

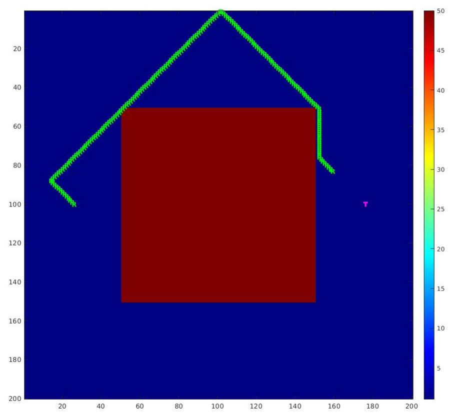

Figure 5: Map 5 Visualization using 2D-Astar

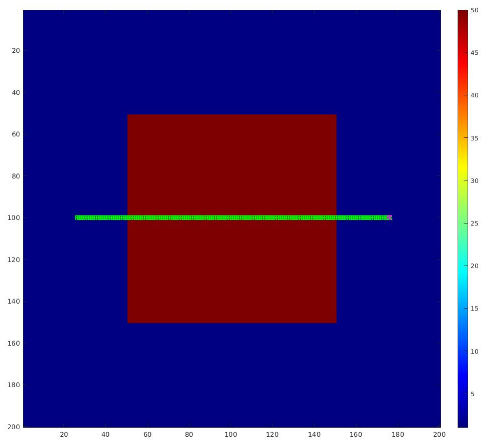

Figure 6: Map 5 Visualization using Greedy Search

| Method | Heuristic | target caught | time taken(s) | moves made | path cost | pre-compute time (s) |
|---|---|---|---|---|---|---|
| 2D-Astar | 2D-Dijkstra | 0 | 182 | 182 | 182 | 0.021189 |
| Greedy | – | 1 | 150 | 150 | 5050 | – |

Table 5: Map 5 Results

## 2.6 Map 6

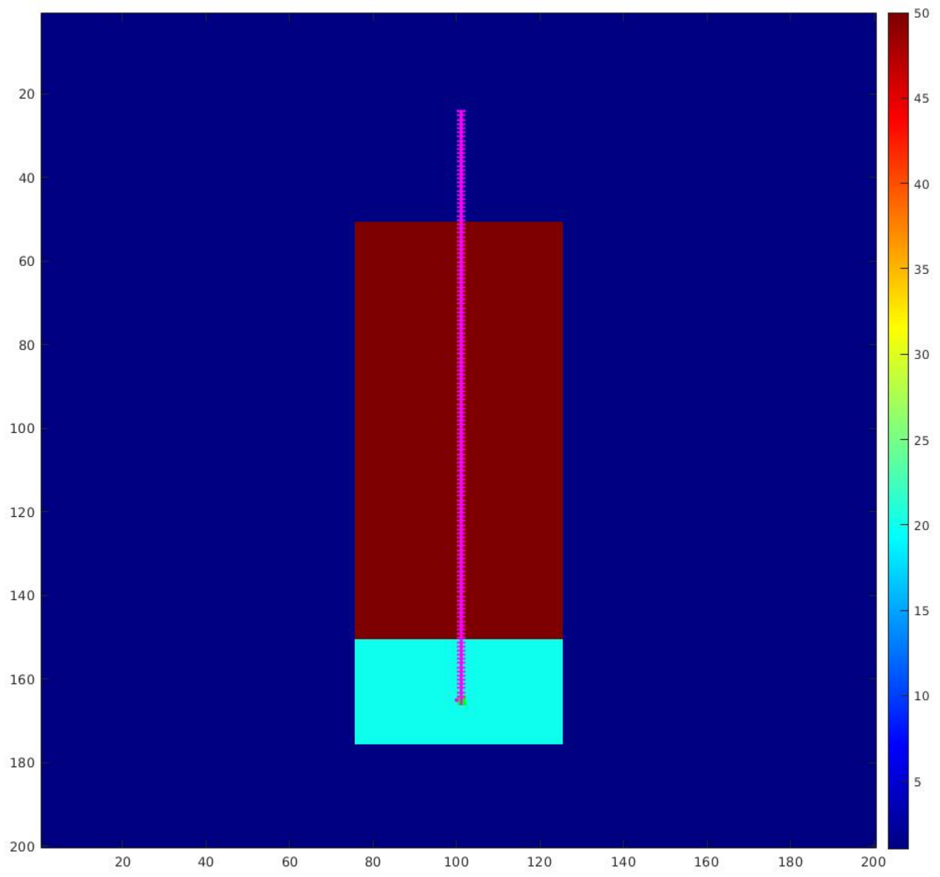

Figure 7: Map 6 Visualization

| Method | Heuristic | target caught | time taken(s) | moves made | path cost | pre-compute time (s) |
|---|---|---|---|---|---|---|
| 2D-Astar | Euclidean | 1 | 141 | 0 | 2820 | 0.024797 |
| 2D-Astar | Dijkstra | 1 | 141 | 0 | 2820 | 0.038948 |

Table 6: Map 6 Results

## 3 Running the Code

```
>          mex planner.cpp
>          runtest('map#.txt')

If run again

>          clear all
>          close all
>          runtest('map#.txt')
>          ...
```

MEX configured to use 'gcc' for C language compilation.

gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1 18.04)

Comments:

I have left relevant comments in the code for the data structures used, and described the method in Section 1.

To use Greedy Search or Euclidean Heuristic with 2D weighted A-star, comment out the relevant lines in the code (have been indicated //***)

3D A-star has been written but not used, since I was unable to get it working. It is in the aStar3D function. The allocated Data structures for that can be ignored.

***Important***
A time_buffer is included in the computation of the Goal function to account for the pre-computation time. This is currently set to 5 secs, because of uncertainty of what system the code will be run on. In Map 1, if the pre-computation time is more than 1.1 seconds (and if buffer is 0), the robot will fail to reach in time. If the buffer is increased, the goal position will be less optimal, and the cost of the path will increase, but this will guarantee that the robot is reaching the intended goal before the target does.

*The results have been obtained with time_buffer = 0, for the most optimal cost.