Space Engineering 3
Assignment 2
1st May 2016

Global Navigation Satellite Systems

Lydia Drabsch
311217591
ldra3557@uni.sydney.edu.au

STUDENT PLAGIARISM: COURSE WORK - POLICY AND PROCEDURE COMPLIANCE STATEMENT

INDIVIDUAL / COLLABORATIVE WORK

I/We certify that:

(1) I/We have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure*;

(2) I/We understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to the University commencing proceedings against me/us for potential student misconduct under Chapter 8 of the *University of Sydney By-Law 1999* (as amended);

(3) this Work is substantially my/our own, and to the extent that any part of this Work is not my/our own I/we have indicated that it is not my/our own by Acknowledging the Source of that part or those parts of the Work.

Name(s): Lydia Drabsch

Signature(s):
Lydia Drabsch

Date: 24/3/16

Contents

1 Grid Based Robot Motion Planning **1**
1.1 Introduction . . . 1
1.2 Martian Terrain Map . . . 1
1.3 Gridded Map . . . 1
1.3.1 Generate Grid . . . 1
1.4 Configuration Space . . . 1
1.4.1 Generate Obstacles . . . 1
1.5 GESTALT . . . 2
1.5.1 Footprint Points . . . 2
1.5.2 Least Squares Fit of Plane . . . 2
1.5.3 Step Hazard Evaluation . . . 2
1.5.4 Pitch Hazard Evaluation . . . 2
1.5.5 Roughness Hazard Evaluation . . . 3
1.6 Dijkstra Algorithm . . . 3
1.6.1 Identify Neighbours . . . 3
1.6.2 Dijkstra Implementation . . . 3
1.7 A* Algorithm . . . 3
1.7.1 A* Implementation . . . 3

2 Question 2 **4**
2.1 Introduction . . . 4
2.2 Dynamic Model . . . 5
2.3 Cost and Constraints . . . 5
2.4 Methodology . . . 6

3 Appendix A: Grid Based Robot Motion Planning **7**

4 Appendix B: Question 2 **12**

List of Figures

3.1 Output from Section 1.3.1 Generate Grid. The start node is in blue and the goal node is in red . . . 7
3.2 Output from Section 1.4.1 Generate Obstacles. The start node is in blue, the goal node is in red and obstacles are black . . . 8
3.3 Output from Section ?? ??. Traversability map: Darker colours indicate a higher cost to travel over. White indicates the rover is unable to travel over. . . . 8
3.4 Output from Section 1.6.2 Dijkstra Implementation. The start node is in blue, the goal node is in red and obstacles are black . . . 9
3.5 Output from Section ?? ??. . . . 9
3.6 A* Euclidean Distance . . . 10
3.7 A* Manhattan Distance . . . 11

List of Tables

2.1 text . . . 5

1. Grid Based Robot Motion Planning

1.1 Introduction

presents the output of each function
discussion of your understanding of each section
how you implemented the relevant algorithms.

1.2 Martian Terrain Map

nothing

1.3 Gridded Map

1.3.1 Generate Grid

The function **generateGrid** calculates the size of the matrix that reflects the discretised map by *mapDim*/*cellDim* and creates the matrix as *map*. The start and end node are calculated using the provided function *pos2cell* and set in *map*. The map is displayed by the predefined function **showmap**, see Figure 3.1.

Inputs:

- *mapDim*: The width and height of the map in meters.
- *cellDim*: The required width and height of each cell on the grid.
- *startPos*: The x and y position of the start point in meters from the centre of the grid.
- *goalPos*: The x and y position of the goal point in meters from the centre of the grid.

Outputs:

- *map*: The discretised global map with start and goal node defined

1.4 Configuration Space

1.4.1 Generate Obstacles

The function **generateObstacles** defines the cells on the map where the rover cannot travel due to predefined obstacles. The map reflects the configuration space of the rover, therefore the radius of the rovers footprint was added to the radius of each obstacle. The x domain of each obstacle was defined in meters discretised by . Using the conic equation for a circle Eq(1), the x domain was to calculate the upper and lower edge of the obstacle in meters.

$$y_i = y_c \pm \sqrt{r^2 - (x_i - x_c)^2} \tag{1}$$

Where x_i is the discretised domain, (x_c, y_c) is the centre of the obstacle, r is the new radius of the obstacle and y_i is the discretised upper and lower edge of the obstacle. The coordinates (x_i, y_i) were converted to indices in *map* by **pos2cell**. For each x_i index, the coordinates between y_i^+ and y_i^- were set as an obstacle in *map*. The output *map* is displayed using **showmap**, see Figure 3.2.

Inputs:

- *map*: The discretised global map from **generategrid**.
- *obstacles*: The x,y coordinates and the radius of each obstacle

- *cellDim*: The required width and height of each cell on the grid.
- *mapDim*: The width and height of the map in meters.
- global *ROVER_FOOTPRINT_RADIUS*: The radius of the footprint of the rover in meters.

Outputs:

- *map*: The discretised global map with start node, goal node and obstacles defined.

1.5 GESTALT

1.5.1 Footprint Points

The function **getRoverFootprintPoints** uses the built-in function **rangesearch** to identify points in *pointCloud* that fall within the rover's footprint radius from the centre of a cell.

1.5.2 Least Squares Fit of Plane

The function **fitplane** fits a plane using least squares method with all the points in the current cell. The mean average vector $\vec{p}$ was calculated from the point cloud *points* and the residuals R were found. The first eigenvector of $R'R$ is the unit vector normal to the plane.

Inputs:

- *points*:

Outputs:

- n: vector normal to the plane
- p: average vector

1.5.3 Step Hazard Evaluation

The function **stepHazardEval** assigns a cost associated with how much of a step hazard the current cell is.

Algorithm 1 Step Hazard Evaluation

```
max step ← max z - min z
if max step < threshold/3 then
    step hazard ← 0
else
    step hazard ← 255 × % max step from threshold
return step hazard
```

1.5.4 Pitch Hazard Evaluation

The function **pitchHazardEval**

The pitch angle θ is calculated using the vector normal n to the plane of best fit as returned from **fitplane** using Eq(2).

$$\theta = \frac{\pi}{2} - \tan^{-1}\left(\frac{n_z}{\sqrt{n_x^2 + n_y^2}}\right) \tag{2}$$

Algorithm 2 Pitch Hazard Evaluation

```
pitch angle ← Eq(2)
if pitch angle > threshold then
    pitch hazard ← 255
else
    pitch hazard ← 255 × % angle from threshold
return pitch hazard
```

1.5.5 Roughness Hazard Evaluation

The function **roughnessHazardEval** evaluates how rough the current cell is by calculating the standard deviation of the residuals,

1.6 Dijkstra Algorithm

The purpose of implementing Dijkstra's Algorithm was to find the path of least cost without having to test every possible combination. It operates by storing the path of minimum cost from a start node to each point on a grid. Dijkstra moves outwards from the start node as a breadth search, regardless of the direction the goal node is in.

1.6.1 Identify Neighbours

The function **getNeighbours** identifies the valid neighbours around the current node based on information from *map*, *mapTraversability* and size of the gridded map. A neighbour is considered invalid if it is outside the global map indices, if it has already been visited or if it was above the traversability threshold. There are two options for how each node is connected to its neighbours; four connections (up,down,left,right) or eight connections (including diagonals). The indices of the valid neighbours are returned from the function and set as 'on list' in *map*.

1.6.2 Dijkstra Implementation

The function **dijkstraBody** identifies if going from the current node to a neighbour is better than any previous calculated paths.

Algorithm 3 Dijkstra Body

```
for each neighbour around node do
    if cost from start(neighbour) > cost from start(node) + traversability(neighbour) then
        cost from start(neighbour) ← cost from start(node) + traversability(neighbour)
        best path so far ← node
    nodes to do ← neighbour
```

1.7 A* Algorithm

The A* algorithm follows the Dijkstra structure, except that it searches towards the goal node first. This was done using a heuristic and implementing cost-to-go and cost-from-start metrics.

1.7.1 A* Implementation

The function **aStarBody** identifies if going from the current node to a neighbour is better than any previous calculated paths.

Heuristic	Weighting factor	numExpanded	Cost increase	% Cost Increase
Dijstra reference	0	1743	60306	-
Euclidean distance	1	1731	0	0%
Euclidean distance	10	1612	0	0%
Euclidean distance	25	1366	587	0.97%
Euclidean distance	50	957	8020	13.30%
Euclidean distance	75	159	15156	25.13%
Euclidean distance	100	67	22644	37.55%
Manhattan distance	1	1730	0	0%
Manhattan distance	10	1573	0	0%
Manhattan distance	25	1235	324	0.54%
Manhattan distance	50	306	7891	13.08%
Manhattan distance	75	74	26397	43.77%
Manhattan distance	100	53	28143	46.67%

Algorithm 4 A* Body

for each *neighbour* around *node* **do**
  if cost from start(*neighbour*) > cost from start(*node*) + traversability(*neighbour*) **then**
    cost from start(*neighbour*) ← cost from start(*node*) + traversability(*neighbour*)
    cost to go(*neighbour*) ← cost to come(*node*) + heuristic to goal(*neighbour*)
    best path so far ← *node*
  nodes to do ← *neighbour*

2. Question 2

2.1 Introduction

The final GEO orbit is defined as having a period of one sidereal day, 23 hours 56 minutes 4.0916 seconds. Using Kepler's third law Eq(3) and the vis-viva law Eq(4) the semi-major axis and velocity of the circular orbit were calculated, see Table 2.1.

$$a = \left(\frac{\mu \mathbb{P}^2}{4\pi^2}\right)^{1/3} \tag{3}$$

$$v = \sqrt{\frac{\mu}{a}} \tag{4}$$

The satellite parameters in the park orbit are

The initial guess of the control parameters were modelled on the Hohmann Transfer and orbit plane change manoeuvres[1]. The burns occurred at the nodes of the intersecting planes of the park orbit and final orbit. The first burn in the park orbit was designed to induce a hohmann transfer orbit, which is an elliptical orbit with the perigee as the radius of the park orbit and apogee as the radius of the final orbit. The plane change was modelled to occur only at the second burn at the apogee as the required velocity change was less.

Table 2.1: text

Orbital Parameter	Initial Value	Final Value
Semi-major axis	6655937 m	
Period		86164.0916 s
Velocity		
Eccentricity	0	0
Inclination angle	-28.5°	0°
RAAN	0°	
Argument of Perigee	0°	
Mean Anomaly	0°	free
Epoch	0 s	free

$$a_t = \frac{a_f + a_p}{2} \tag{5}$$

$$e_t = \frac{a_f - a_p}{2a_t} \tag{6}$$

$$v_{perigee} = \sqrt{\mu_{earth}\frac{1+e_t}{a_p}} \tag{7}$$

$$v_{appogee} = \sqrt{\mu_{earth}\frac{1-e_t}{a_f}} \tag{8}$$

2.2 Dynamic Model

The dynamic model used was the Universal Conic Section (UCS) model as it provided an analytical solution for any point in time of the system. This allowed for faster calculations over a propagation model such as the modified equinoctial elements. The Keplerian model is also an analytical solution, contains singularities when the inclination or eccentricity are zero. While other disturbing forces such as atmospheric drag or the oblateness of the Earth are not accounted for, as the model is only used for one period the effects are negligible. The satellite trajectory was modelled in **dynamics** as Cruise1 (ΔE_1), Burn1 ($\Delta V_1, \theta_1, \phi_1$), Cruise2 ($\Delta E_2$), Burn2 ($\Delta V_2, \theta_2, \phi_2$). The cruise sections used the UCS model and returned the final state vector and the duration in seconds that cruise occurred for. The burn sections applied an instantaneous velocity increase on the given state vector in the ECI frame.

2.3 Cost and Constraints

The Goddard rocket equation in Eq9 describes the system dynamics of the model; where m is the rocket mass, h is the height above the ground, v is the velocity of the rocket, g is the gravitational constant, u is the engine thrust control parameter and D is the atmospheric drag.

$$\dot{v} = \frac{1}{m}(u - D(v,h)) - g\Delta v = g_0 I_s p \ln(\frac{m_o}{m_o - F}) \tag{9}$$

$$F = \frac{dp}{dt} \tag{10}$$

$$\int F dt = \int dp \tag{11}$$

$$p_1 + \int F dt = p_2 \tag{12}$$

$$P_1 + \sum \int F dt = P_2 \tag{13}$$

$$P_1 + \sum (F \Delta t) = P_2 \tag{14}$$

$$\tag{15}$$

What are constraints - why are they important.
The constraints on this optimisation problem defines the final GEO orbit. They are The radius and velocity are enforced by requiring the ratio between the

$$J = |\Delta v_1| + |\Delta v_2| \tag{16}$$

2.4 Methodology

Due to the problem being multidimensional, non-linear and constrained, the system is approximated as quadratic locally and successively solved by the Newton method. This is called Sequential Quadratic Programming.

The Hessian of the Lagrangian $H_{\mathcal{L}} = \nabla^2_{xx}\mathcal{L}$
BFGS method. Approximation of the Hessian update

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k - \frac{\boldsymbol{H}_k s_k s_k^T \boldsymbol{H}_k}{s_k^T \boldsymbol{H}_k s_k} + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T s_k} \tag{17}$$

$$s_k = x_{k+1} - x_k \tag{18}$$

$$y_k = \nabla f_{k+1} - \nabla f_k \tag{19}$$

Algorithm 5 Sequential Quadratic Programming

Input: Initial guess of control (Y0), initial state (X0)
Output: Optimal Solution (Y*)
Initialise:
while error > tolerance **do**
  (c,g,G) ← calculate matrices (Y_i)
  L ← Calculate Lagrangian (Y_i, λ_i)
  H_L ← Calculate Hessian of Lagrangian (Y_i, λ_i)
  (p,λ_{i+1}) ← Solve KKT for direction and Lagrange multiplier (H_L,c,g,G)
  α_i ← linesearch
  Y_{i+1} ← Calculate next iteration $Y_i + \alpha_i p$

References

[1] Rocket and space technology. [Online]. Available: http://www.braeunig.us/space/orbmech.htm

3. Appendix A: Grid Based Robot Motion Planning

Figure 3.1: Output from Section 1.3.1 Generate Grid. The start node is in blue and the goal node is in red

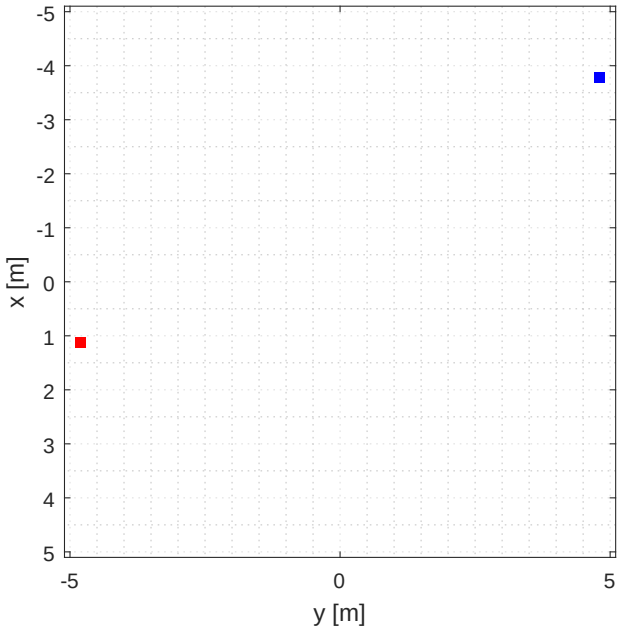


Figure 3.2: Output from Section 1.4.1 Generate Obstacles. The start node is in blue, the goal node is in red and obstacles are black

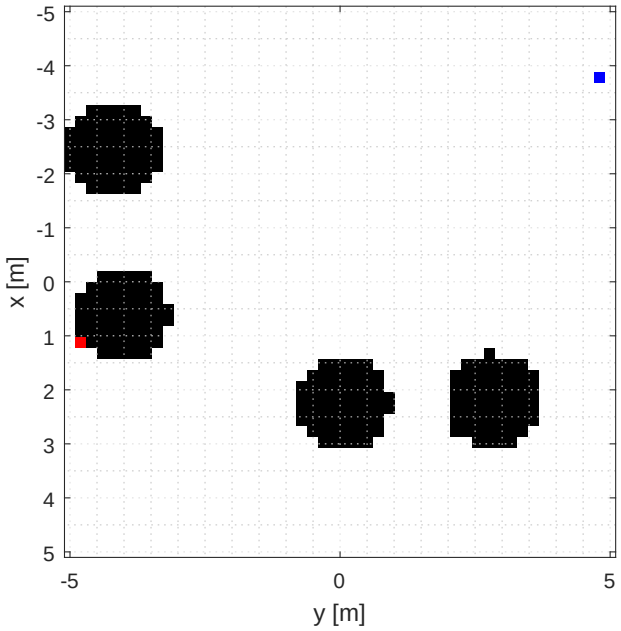


Figure 3.3: Output from Section ?? ??. Traversability map: Darker colours indicate a higher cost to travel over. White indicates the rover is unable to travel over.

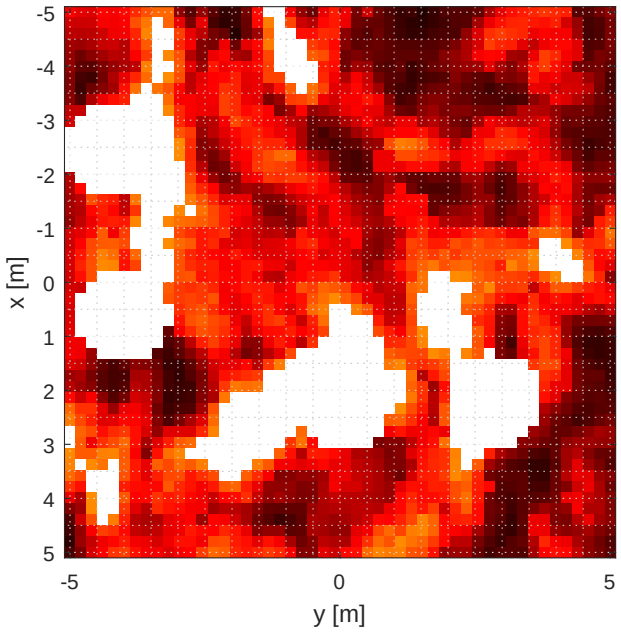


Figure 3.4: Output from Section 1.6.2 Dijkstra Implementation. The start node is in blue, the goal node is in red and obstacles are black

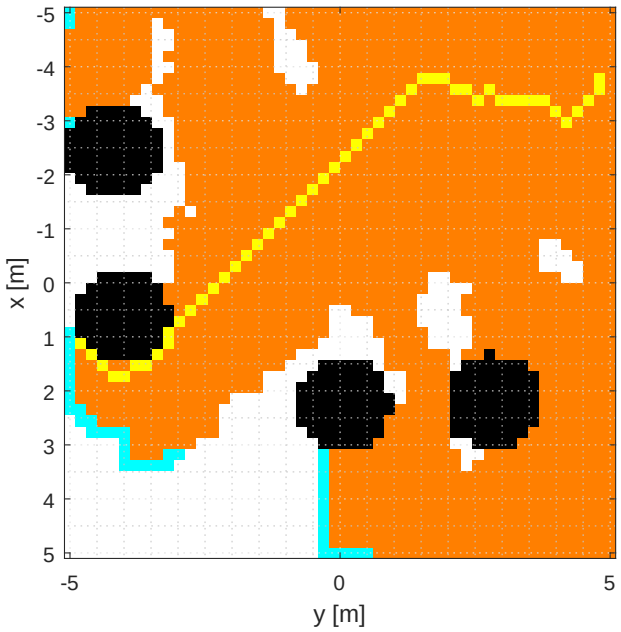


Figure 3.5: Output from Section ?? ??.

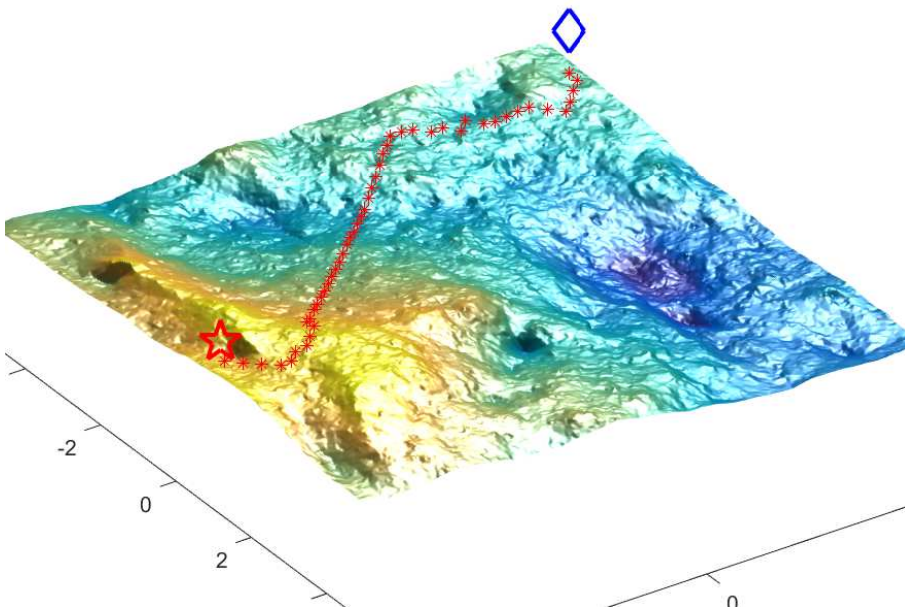


Figure 3.6: A* Euclidean Distance

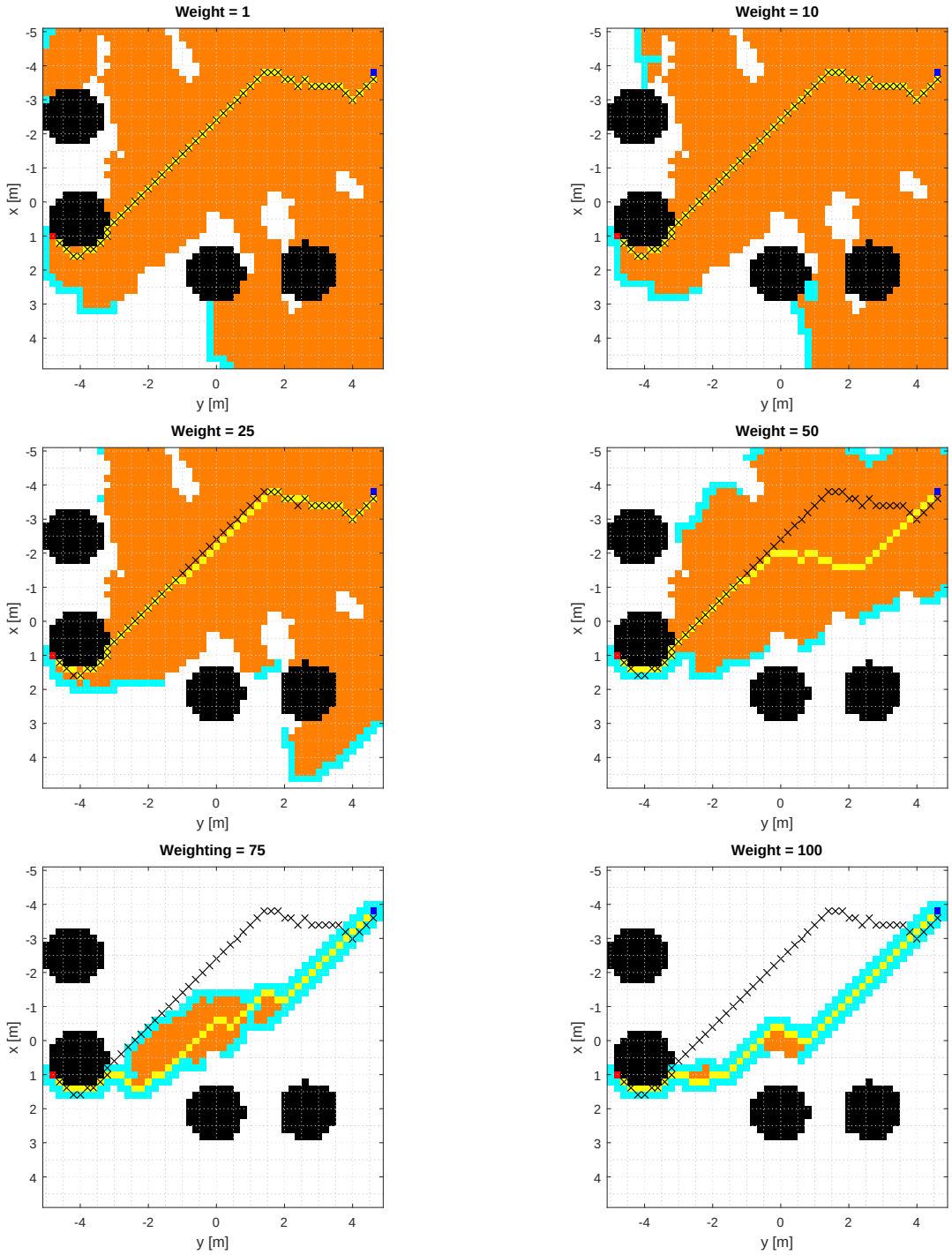
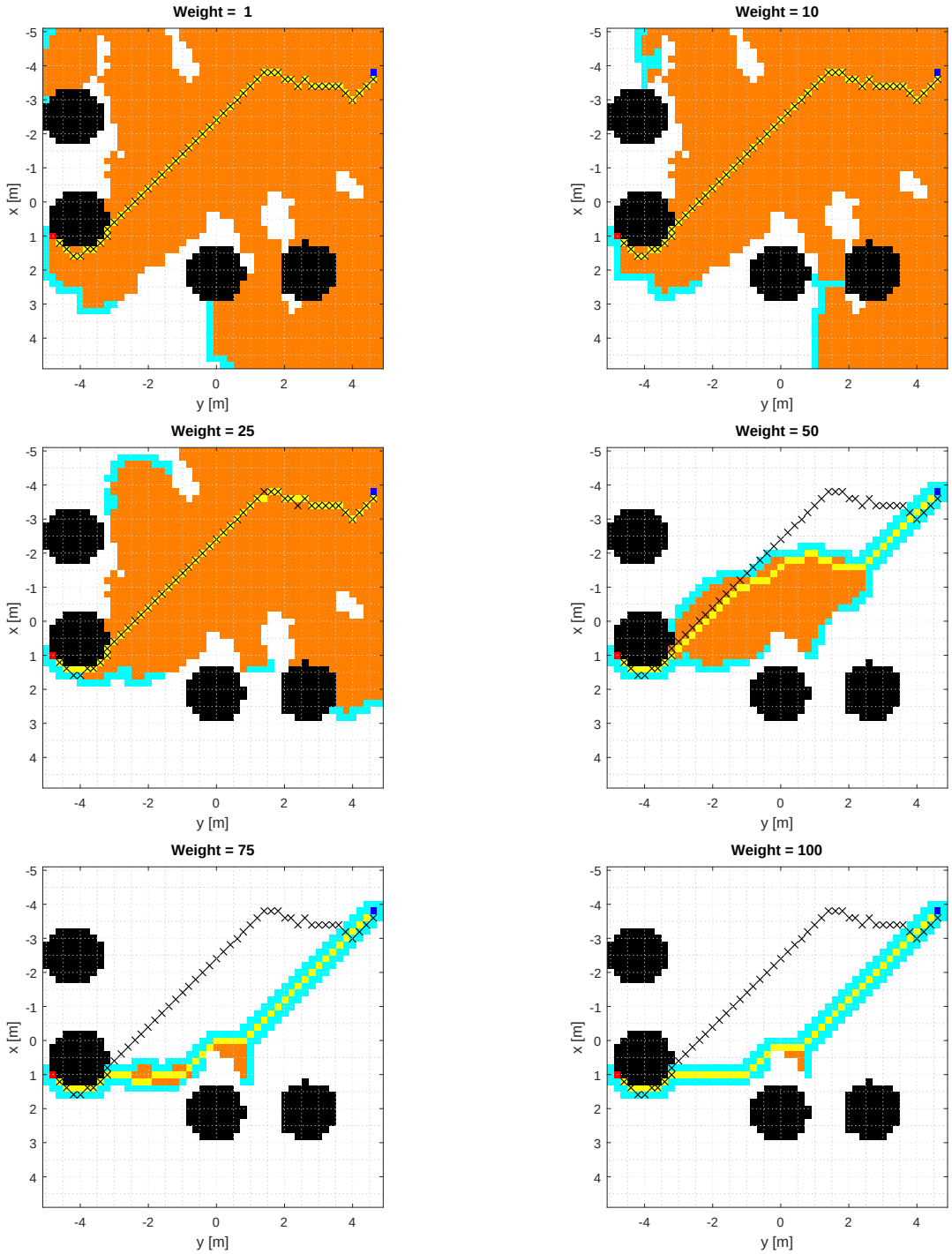


Figure 3.7: A* Manhattan Distance



4. Appendix B: Question 2