Prac 2 Report: Path planning and robot control

Cody Cripps

N9945008

Queensland University of Technology

EGB439 Advanced Robotics

Preface:

This reports details an approach to the task of creating an algorithm to plan a path in a scene defined by an IR image and subsequently controlling a differential drive robot to follow that path.

1. APPROACH

The approach to the task of making a robot traverse an unseen set of obstacles that was used, was to break it into key systems. These systems are as follows:

- Mapping (from localiser) and Path Planning
    - Distance Transform
    - Path Planning Algorithm
- Robot Control
    - Drive to Point
    - Moving Goal Algorithm (Pure Pursuit)

1.1. Path Planning

Path planning was approached as a cost-to-come function.

This was achieved by first creating an occupancy grid of the space by thresholding an IR image obtained from the Localiser above the workspace. As the robot has volume, it cannot be fully treated as a point, therefore the occupancy grid was then dilated to allow the robot space to avoid the obstacles.

A distance transform to a given goal was then computed. The transform used a Euclidean distance, that is that it assumed that each of the compass directions had a cost of 1, while diagonal movements cost $\sqrt{2}$. As this transform computes in relation to a goal only, it can be used for any starting position or orientation. The process of computing this can be seen in the pseudocode below:

while $numberofinfinites > 0$
    For Cell in OccupancyGrid:
        W $\leftarrow$ Window (3x3 grid) centred on cell (from OccupancyGrid)
        $$\text{cell} \leftarrow min\left(W + \begin{matrix} \sqrt{2} & 1 & \sqrt{2} \\ 1 & 0 & 1 \\ \sqrt{2} & 1 & \sqrt{2} \end{matrix}\right)$$
    EndFor
EndWhile

This means of ending the while (checking if there are any infinite values remaining) is not ideal, as it results in an infinite loop if there is any space that is seperated from the goal. This was therefore later rectified by adding the while condition → && number of infinities has reduced. The result of this distance transform can be seen in figure 1.

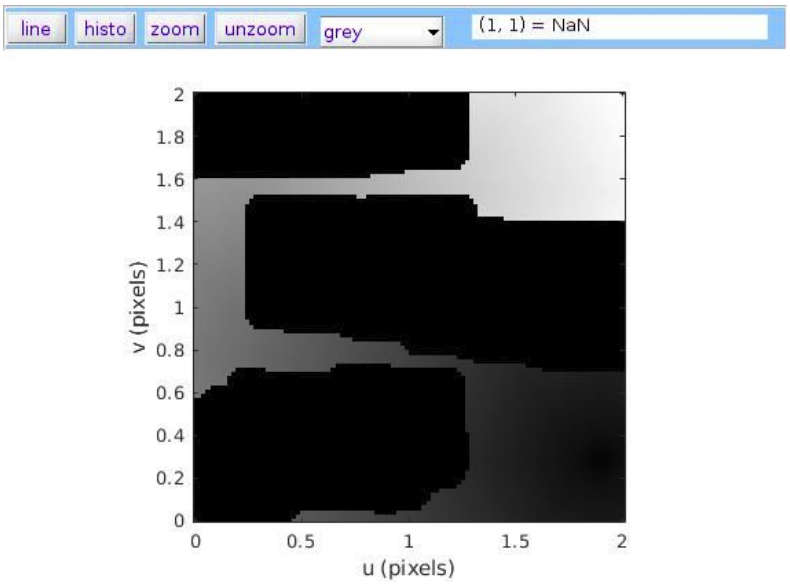

Figure 1: Distance Transform

This distance transform was then used to compute a path with a sort of rolling ball algorithm. That is based on the each successive adjacent cell from a start point with the least distance to the goal. Where there was two, the first found by matlab's find function were used as either way would have equal distance (cost). This path would then be sent to the robot control system to make its way around the track. An example of a path planned using the code can be seen in figure 2.

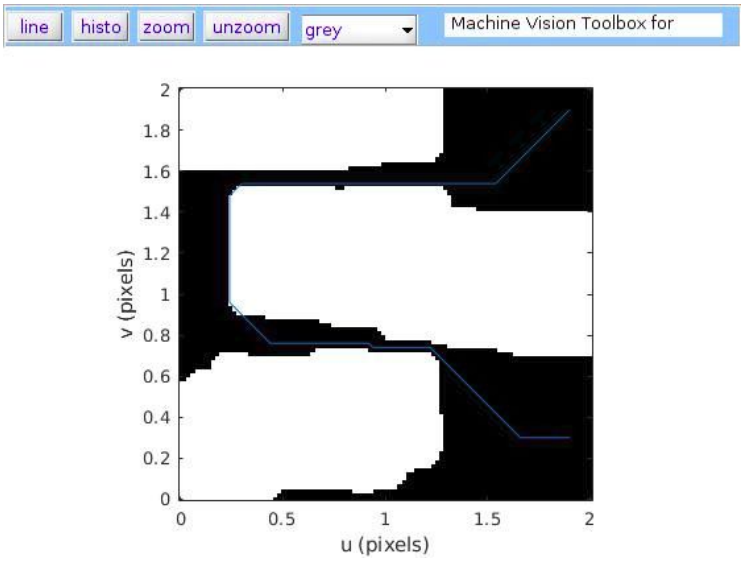

Figure 2: Planned Path

1.2. Robot Control

A simple control algorithm for following a given series of points is pure pursuit. This control algorithm uses a simple drive to point controller with a moving goal point.
For the drive to point control, there were two parts.

1. Heading control → A simple proportional controller was implemented for the heading. This was all that was required with the heading as the robot is easily maneuvred in a yawing motion.
2. Velocity Control → This was completed with a PI (Proportional-Integral) controller. A proportional controller would be sufficient for the majority of the distance however it resulted in a steady state error that left the robot a short distance from the goal.

The algorithm initially intended for the receding goal point was for it to recede with the movement of the robot, so as to stay a minimum distance from the PiBot. This however was scrapped for a simpler time variant goal, which proceeded through the way points at a set interval. This algorithm was less efficient if the robot was in a pose that was not ideal, however with good control code, it was sufficient and much simpler to code.

2. RESULTS

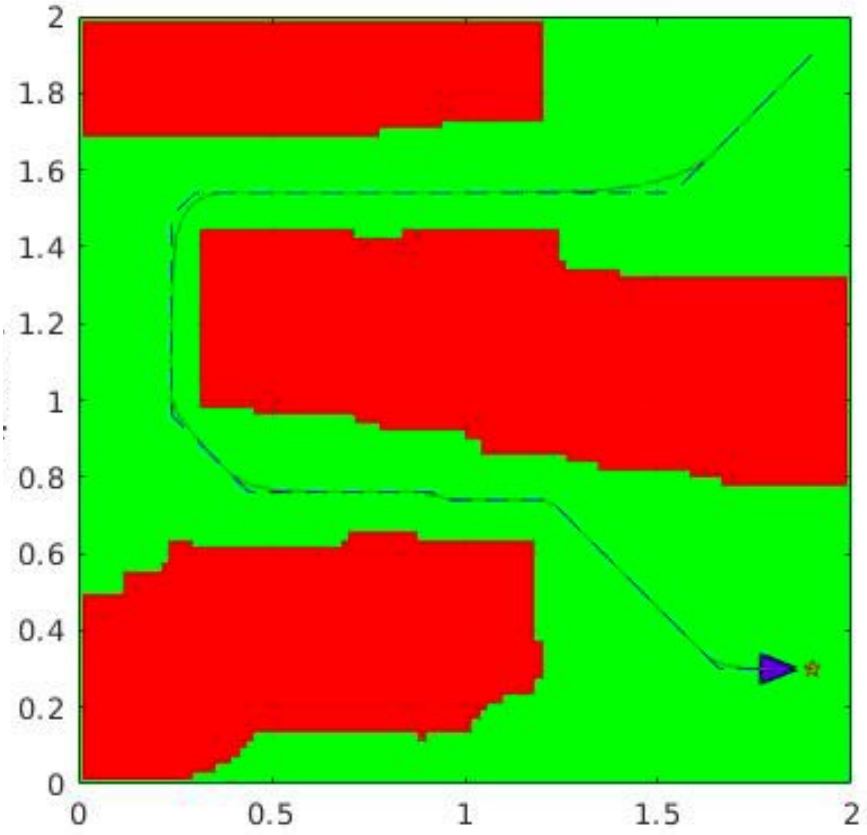

Figure 3: Simulated robot following the planned path.

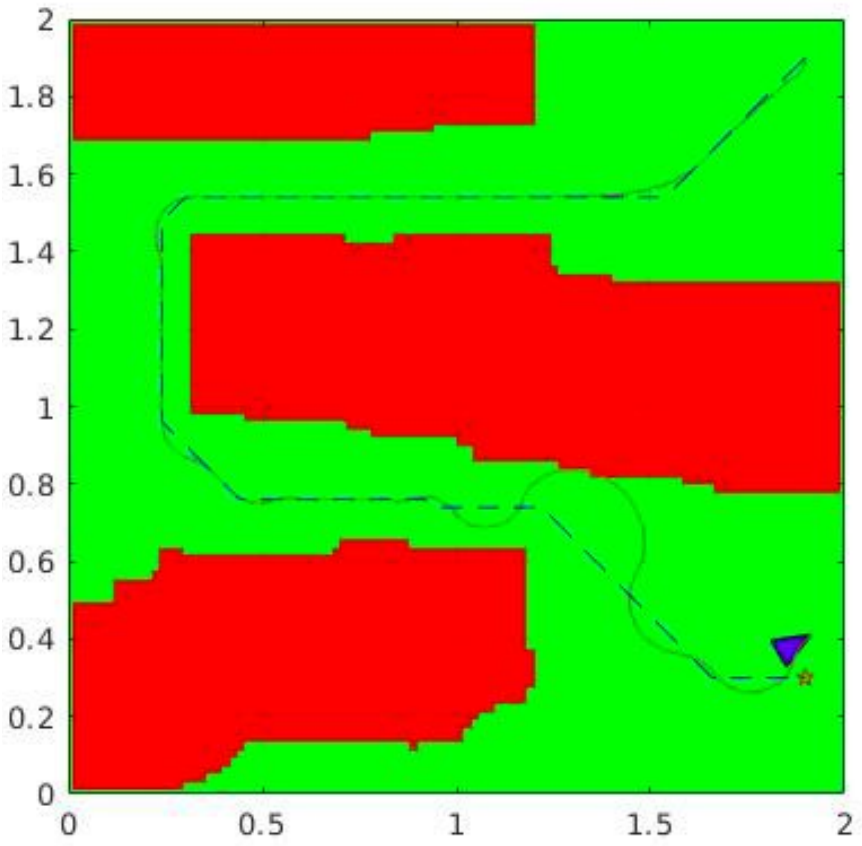

Figure 4: Actual robot Following Path

Format	Time (s)
Simulation	20.39
Actual	20.89

3. Discussion

It can be seen in the results the actual robot followed a much more chaotic path than the simulation. It can also be seen that this caused inefficiencies in it's time to complete the path. The likely cause of this was a badly modeled simulation. The way in which the robot deviated was that it turned more sharply than the simulation and it had a higher speed which implies that the conversion from velocity vectors to motor commands is badly simulated and

may also be incorrect in the motor control that was written for the robot. Further refinement of the motor conversions is key to a better result in future.