Fall 2015 Robotics Project Report

Automated Robo Taxi using the A* Algorithm in RobotC

By: Michael Annor and Agatha Maison

1.0 Introduction

This project models an automated taxi system that efficiently finds the quickest route for passenger pickups and drop-offs. The project is implemented using Lego Mindstorm kits as a proof of concept for the development of intelligent automated robotic transportation systems using a suitable planning algorithms. The system includes a base station that receives and processes requests for rides from passengers. Processing of these inputs is done using the A* path planning algorithm to determine the closest taxi to deploy to the requesting passenger. Mobile taxi robots then determine the shortest route to complete the given task, also using the A* algorithm. The A* algorithm is used because it guarantees the shortest path between the start and the goal.

2.0 Background

This project is based on the application of a planning algorithm to complete a meaningful task with real-life significance. Developing a proof of concept of an intelligent automated taxi system, dovetails with the current technological advancements in the fields of driverless cars and unmanned vehicles by large corporations like Google. This project is inspired by a paper review of Automated Taxi/Cab System Using A* Algorithm (Kumar and Kumar) and can be applied across other domains including automated factory/warehouse process that involve planned navigation as well as in agriculture.

3.0 Architecture

The automated robo taxi system has two mobile taxis for pickups and drop-offs and a main base station that acts as the controller and user interface. The parts of the system are connected via using Bluetooth for message passing. The system is developed in Robot C and deployed on the Lego Mindstorms platform.

3.1 Hardware Stack

Lego Mindstorm NXT (3)

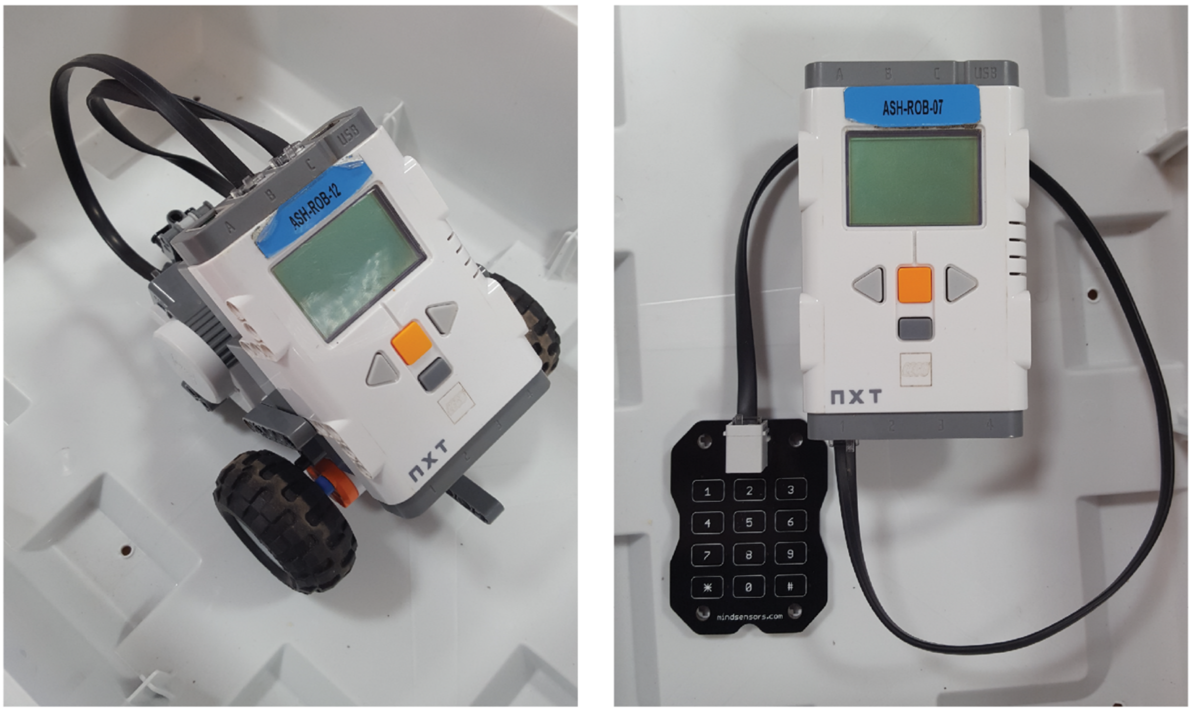
Mindsensors Numeric keypad

3.2 Software stack

Robot C IDE

3rd party Robot C driver suite

Figure 1: Mobile Robo-taxi [Left] and Base station with numeric keypad [Right]

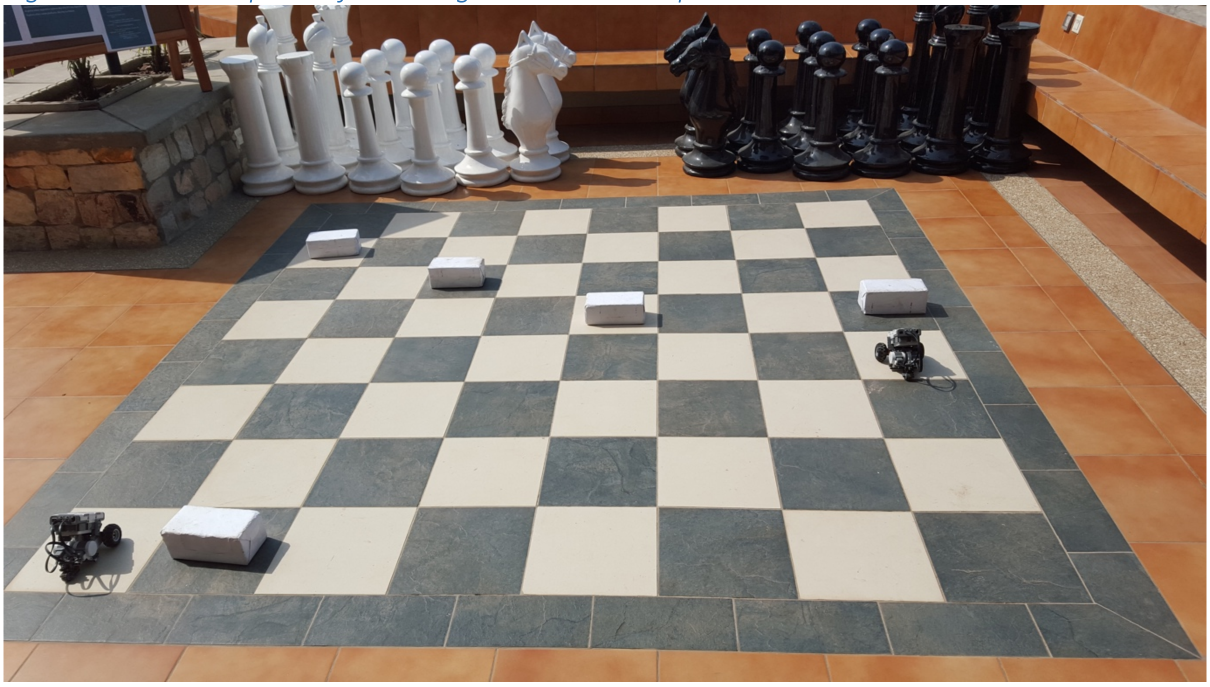


4.0 Approach

4.1 Implement the A* algorithm

Taking the world map as a graph of traversable nodes, the A* algorithm uses heuristic and movement cost estimates to plan the shortest path (set of edges) to a goal node from a given start node. The world-representative graph is modelled using a grid map, implemented with a two-dimensional array of ints. The array is populated with cell values 1s and 0s, with 1s being obstacle cells and 0s being traversable cells.

Figure 2: World map used for testing. White bricks represent obstacle cells

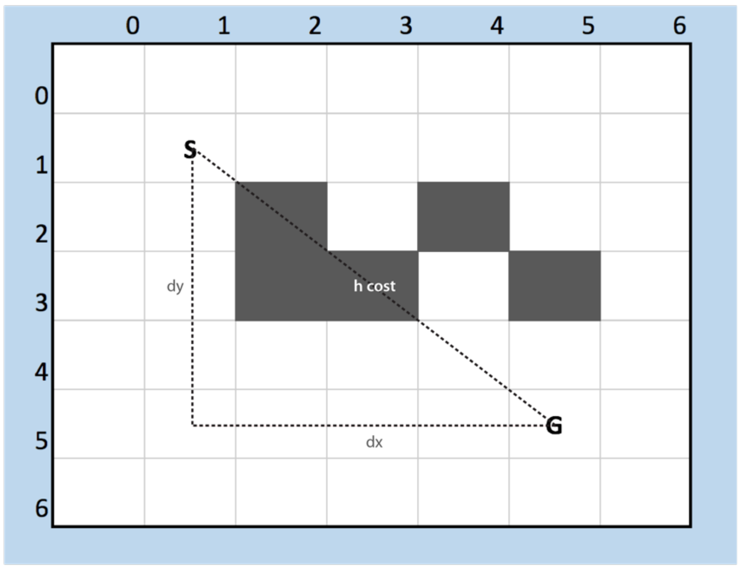


To keep all the information about each cell in the world, a struct is defined containing all the properties of a cell. These include the the heuristic value (h cost), the movement cost (g cost), the F value (a sum of the g cost and h cost), the x and y coordinates, the cell value and the parent cell, as well boolean values for its inclusion on the open or closed list. A node map, equivalent to the world map is then defined to capture the complete state of the world. The node map is created as the first step of the path planning process. This is an expensive process, especially as the world map gets larger but the design decision is made because of the benefits of having complete knowledge of the world.

The algorithm uses an open list to keep track of nodes being explored and to retrieve with an optimal runtime of O(logN), the node with the lowest f cost. The algorithm also uses a closed list to keep track of already explored nodes. To satisfy the O(logN) condition for retrieving from the open list, a priority queue was implemented to store the explored nodes. the nodes in the node array have a Boolean variable to keep track of their open list status. The algorithm also uses a closed list for cells that have been explored and since the list is not accessed by priority, an actual data structure is not used. Instead a boolean value in the node struct determines whether or not the cell is closed. This check, provided the cell's coordinates are known has an acceptable runtime of O(1).

Regarding heuristics, the straight line distance is used to calculate the the h cost. For each node cell, the h cost is given as the hypotenuse of the right-angled triangle formed from the cell to the goal cell.

Figure 3: Grid map showing the h cost and calculations

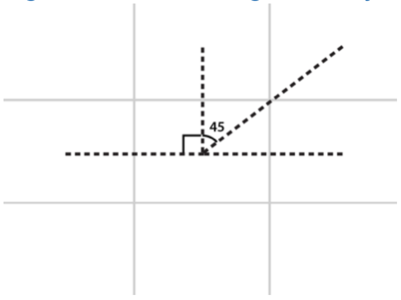


The h cost may be given by:

$$\text{h cost} = \sqrt{((dy)^2 + ((dx)^2))}$$

In this project, planning using the A* algorithm assumes eight-point connectivity to take full advantage of the shortest path that the algorithm gives. The implication of 8-point connectivity is that a given cell has a maximum of 8 traversable neighbours. This implies making diagonal moves which in turn means a greater movement (g) cost than the vertically and horizontally positioned neighbours. Again, using Pythagoras theorem, the movement cost from one cell to another is estimated to be 14 for diagonals and 10 for verticals and horizontals.

Figure 4: Calculating h costs from one cell to another



h cost from one cell to another may be given by:

vertical h cost (vh $= 1 \times 10 = 10$
horizontal h cost (hh) $= 1 \times 10 = 10$
diagonal h cost $= \sqrt{((vh)^2 + ((hh)^2))} = \sqrt{(1+1)} = \sqrt{(2)} = 1.4 \times 10 = 14$

The f cost is a simple addition of the g and h costs and is the determinant for selecting a favourable cell to move to. The lower the f cost, the shorter the path through that cell.

4.2 Pseudocode for implementing the A* algorithm

Code Sample 1: A algorithm pseudocode implementation*

```
OPEN
CLOSED

add the start node to OPEN

loop
    current = node in OPEN with the lowest f_cost
    remove current from OPEN
    add current to CLOSED

    if current is the target node
        return

    foreach neighbour of the current node
        if neighbour is not traversable or neighbour is in CLOSED
            skip to the next neighbour

        if new path to neighbour is shorter OR neighbour is not in OPEN
            set f_cost of neighbour
            set parent of neighbour to current
            if neighbour is not in OPEN
                add neighbour to OPEN
```

4.3 Navigating the real world map with a robot

Once the algorithm has run completely, stepping through from the goal node to the start node using each cell's parent should result in the shortest possible path. To enable the robots navigate the world after the path is found, the path cells are placed in an array. One key thing about navigation is the orientation of the robot. Starting off at North and repositioning to face North at the destination, the movement algorithm initialises the robot's orientation to North. Unable to find a more efficient way to program the movement, 64 branches of code are written to cater the movement to any eight of the cardinal points while facing any one. Nonetheless, with this, the robot can move in a natural manner from a cell to any of its neighbouring cells. To allow for variable obstacle sizes in cells, the navigation algorithm bars the robot from cutting corners at obstacle cells.
For movement, the encode ticks of the robot's actuators are used to move from once cell to another. Considerations that need to be made include moving vertically or horizontally and moving diagonally as well as the angles of turn. Left and Right turns are $\pm 90°$ turns and diagonals were $\pm 45°$ turns.

The actuator encoder ticks for moving forward are derived by:

$$\text{Encoder ticks (forward)} = \frac{\text{Distance to move} \times 360}{Circumference\ of\ wheel}$$

4.4 Multi-robot coordination

To make Robo Taxi, the model of the automated taxi system functional, the different parts of the system need to be well setup and aligned. For the base station which acts as the interface a user interacts with, a Lego Mindstorm NXT Brick is used to accept inputs in the form of pairs of x and y co-ordinates for the passengers' locations and destinations. For better interactivity, the Mindsensors keypad, a third party NXT peripheral is used. The base station is then paired to the two mobile taxi robots (Taxi 1 and Taxi 2) using Bluetooth. The base station using the A* algorithm with knowledge of the starting points of both Taxi 1 and Taxi 2 determines which taxi is closest to the passenger. Given that both taxi robots are paired to the base station in a loosely coupled manner, an additional parameter is sent along with the x and y coordinates for the passenger location and destinations to enable the receiving robots interpret. Table 1 below shows the listed parameter codes and their respective interpretations. This allows only the appropriate robot to act on a sent message.

Table 1: Bluetooth Message Passing Parameter Codes and Interpretations

Parameter Code	Target robot	Interpretation
11	Taxi 1	Message is meant for taxi 1 (closer to passenger) and x,y values represent the passenger location
12	Taxi 1	Message is meant for taxi 1 (closer to passenger) and x,y values represent the passenger destination
21	Taxi 2	Message is meant for taxi 2 (closer to passenger) and x,y values represent the passenger location
22	Taxi 2	Message is meant for taxi 2 (closer to passenger) and x,y values represent the passenger destination

Code Sample 2: Sample Bluetooth Message Passing Code, parameter code in bold

```
sendMessageWithParm(11, x, y);
sendMessageWithParm(22, x, y);
```

With the locations and destinations of a given passenger, the mobile taxi robots in turn use the A* algorithm to plan and navigate the shortest path. To ensure that message passing between robots was accurate, the design decision is made to send messages in smaller sets. This means, the base station does not transmit the path to the mobile taxi robot. Rather, the mobile taxi re-computes the path using the same implementation of the A* algorithm. The surety of the message passing means that the expensive nature of the repeated computation process is a worthwhile cost.

5.0 Results

The automated taxi system implemented in this project is able to deploy taxi requests to the nearest available taxi system, with only one taxi moving at a go. The system uses the shortest routes possible. If the passenger's location is equidistant to both mobile taxis, the system breaks the deadlock by picking one of the robots. The movement implemented using calculated encoder ticks means the robot can to a high degree of accuracy navigate from one cell to another. This means testing on the desired real-world map works as planned. However, the movement is not transferrable to other maps of different cell sizes.

6.0 Conclusion

The work in this project demonstrates that a basic implementation of a planning algorithm can be used to meaningfully automate tasks in real-life operations to improve productivity and efficiency. The A* algorithm is seen to produce the shortest possible path. The overhead processing costs makes way for more efficient ways of implementing such an automated system to be scaled up to life-sized models. This realization largely influences the paragraphs on future works.

A video of the working Robo Taxi can be seen at:
https://www.youtube.com/watch?v=Hkt2jm5N-_A

7.0 Future Work

This project may be improved in various ways to increase efficiency and runtime as well as make it more functional. Future versions of the automated robo-taxi would be developed to enable additional functionality for both mobile taxis to navigate the world map in real-time without obstructing each other. This requires knowing the current locations of all navigating robo-taxis and having a mechanism for breaking deadlocks that arise.
Although the A* algorithm assures a shortest path possible, it has an expensive implementation, as it explores cells that do not contribute to the eventual shortest path. Alternative algorithms like the D* algorithm or the D* Lite algorithm may be used to improve the efficiency of the automated robo-taxi, thus making it more scalable to life-sized robots.
Future developments on the automated robo-taxi would also handle mapping to allow a robot use a select-set of sensors to navigate a world and generate a map that can be used by the current system. This fully automates the system and makes is scalable for use across domains. In addition, the base station could be made more interactive by a complete abstraction of the workings of the map. Instead of giving inputs in the form of x and y co-ordinates, allowing natural language inputs like place names that can be mapped to cell co-ordinates with a dictionary would improve its usability.

Finally, to make all these developments applicable, future developments would explore using more advanced robotics platforms such as ROS and the turtlebot, a step closer to deploying the automated robo-taxi on real cars, forklifts, drones, etc.