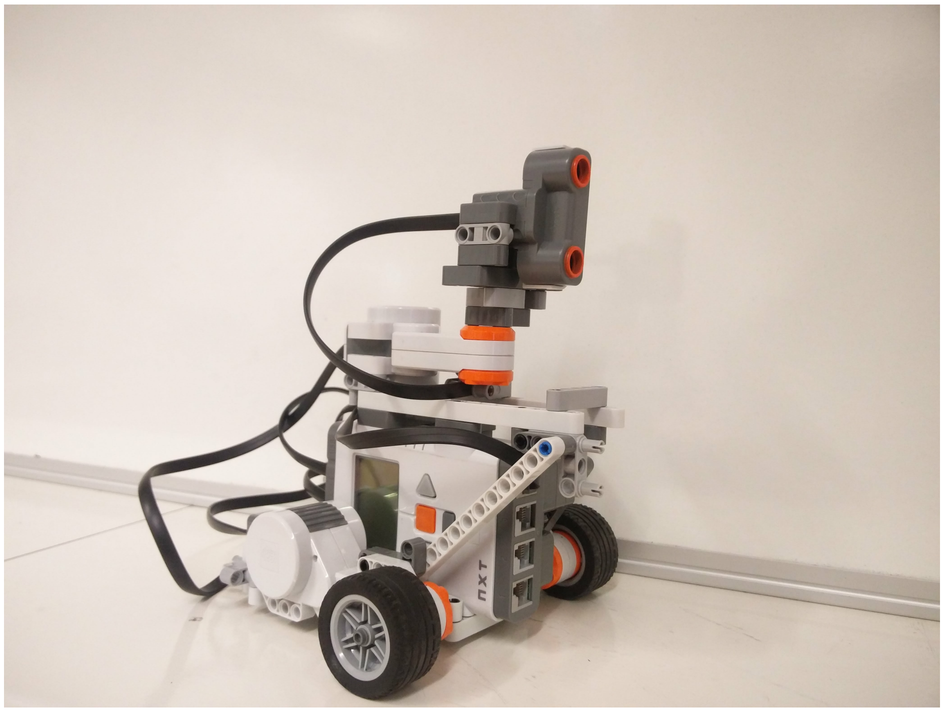
Kidnapped Robot

Coursework in Robotic Systems

University of Bristol, March 28, 2017



Group RLRS

Name	**Email**	**Contribution Percentage**
Shichao Dong	sd16998@my.bristol.ac.uk	34%
Lap Kong Lam	ll12503@bristol.ac.uk	33%
Johan Larsson Horken	jl16326@my.bristol.ac.uk	33%

1 Introduction

The kidnapped robot problem refers to a situation where an autonomous robot operates in a known environment, but without previous knowledge about its location. To be able to efficiently operate, the robot first needs to have information about its location so that actions are performed accurately in accordance with its environment.

This project undertakes a study of the kidnapped robot problem which requires building a robot using Lego NXT and developing a localisation and navigation system which allows the robot to start from any random position on the map and to reach a target position accurately and efficiently. A particle filter is used for estimating the robot's position and orientation. Given the estimated position the A* algorithm is used to calculate the best route from the robot's current position to the target position.

2 Robot

The robot used in this project was based on Lego NXT [1]. The building material available were three servo motors with built in optical sensors for rotation measurement, one touch sensor giving logical value 1 when pressed (otherwise 0) and one ultrasonic sensor to measure distance from the sensor to the closest facing object, as well as various Lego building blocks and the NXT Intelligent Brick.

2.1 Design

Focus during robot design was a modular robot that would be easy to adapt, since later problems requiring reconstruction were foreseen. This was achieved by restricting the design to not use any redundant parts and make sure the NXT Brick was not locked in its position by too many overlapping parts. This also made maintenance easier, such as changing batteries. To enable the robot to take 360°scans with the ultrasonic sensor without having to turn the entire robot it was decided to mount the sensor on a vertically rotating motor. To make the ultrasonic measurements as accurate as possible, the sensor should be placed at the center of rotation. Since only two motors were available for movement, the center of rotation could be estimated to between the two wheels attached to the remaining two motors. For this reason, a design using tracks was not used, since it would render it harder to determine the center of rotation. For stability reasons a small castor wheel with low friction, was positioned opposite the driving wheels.

To make later path finding and movement as easy and accurate as possible, the robot was designed as compact as possible. Since the height restrictions were rather generous, and the height of the robot would not matter as much for movement and navigation, not much focus was spent on optimising the height of the design, as long as the height still allowed the ultrasonic sensor to make valid measurements.

2.2 Construction

The construction was initiated of by placing the motors next to each other and linking them together with Lego blocks to form a base. The width of the base was just wide enough to support the NXT Brick on edge, making the base rather compact (16x15 cm). Since the main part of the width was the motors and the wheels, it was hard to make the base much smaller without making the construction more complicated, thus less modular. The NXT Brick was placed on edge on top of the base, making it easy to detach and move. The third motor was placed on top of the NXT Brick, supported by Lego bricks to place the motors rotation point centered above the two driving wheels. The ultrasonic sensor was then placed on the rotation point of the motor, with its sending and receiving sensors aligned vertically to improve accuracy further. This resulted in the robots final height of 20 cm. A castor wheel in shape of a small knob was chosen as a castor wheel, since it was the part with best fitting size, and placed opposite the driving wheels. The construction was finished off by adding various Lego blocks to support the construction and make it more stable. The final result can be viewed in the Figure on the cover page.

[1] *Mindstorms*, The LEGO Group - `https://www.lego.com/en-gb/mindstorms` [Accessed 26/03/2017]

3 Movement, Optimization and Calibration

For the robot to function satisfactory it requires functions for moving (translating), turning and making ultrascans. This section will introduce these functions as well as describe how the functions have been calibrated to work optimally.

3.1 Translation

The translate function allows the robot to move straight from point to point with millimeter precision. The motor function takes speed(power) and tacholimit as input. Based on the equation

$$tacholimit = \frac{distance}{diameter * \pi/360}, \tag{1}$$

the input distance could be converted to tacholimit.

In the real world, the movement of the robot is affected by friction. The task here is to find a calibration factor to compensate the error caused by friction. The optimal value 1.025 (2.5%) was found by testing. Using this calibration factor the Gaussian noise were less then one centimeter, which was deemed accurate.

The NXT motor has a rotation encoder, returning the position of shaft with 1 degree resolution. The shaft position was reset before movement, the encoder reading was then compared to the set tacholimit. If there was any difference, the program would make an additional small movement for adjustment. Based on observation, there were a few degrees of errors sometimes. The movement adjustment helps to correct these errors and makes the movement function more accurate in long movement.

3.2 Rotation

The turn function makes the robot rotate, specified by an angle in degrees. The robot turn maximum 180°left or right, since all greater angles could be simplified.

Similarly to the move function, the turn function works by setting the tacholimit. The two wheels rotate with the same speed but opposite direction, making the robot turns on the spot.

When trying to calibrate the turn function, there was a considerable difference between turning left and right with the same angle. If the calibration was only done for one direction turn, it would not work very well for the others. This might be caused by the structure of the robot or the mechanism of wheels. The calibration program first checks the turning direction, then picks calibration factors based on direction (left is 300, right is 305). In this way, turning for both left and right were calibrated.

3.3 Ultrasonic Sensor

The ultrasonic scan function allows the robot to make an evenly distributed 360°scan. During testing the position of the ultrasonic sensor did not stay the same as its original position after rotating back. This error was accumulated, which resulted in an angular mismatch. Therefore the same adjustment procedure as in the move function was applied to the movement of the sensor, which created more accurate position after rotating back.

The robot also suffered from noisy readings from the ultrasonic sensor. To calibrate the sensor noise, the robot was placed at different distances from the wall. For each place, ten ultrasonic scan were taken and their average standard deviation was used as noise coefficient.

4 Localisation

The localisation uses a particle filter to estimate the robot's position and orientation using its sensor observations in a known map of the environment. The distribution of the likely states of the robot is represented as particles, where each particle acts as a hypothesis of the state of the robot. After multiple iterations, an estimated position and orientation can be extracted from the particles. [2]

[2]Thrun, Sebastian - *Probabilistic robotics* Communications of the ACM 45.3 (2002): 52-57

An attempt was made to implement the localisation function. Due to the limited time, the estimated location and orientation of the robot was not accurate enough, leading to the inaccuracy in the path planning. Thus, it was decided to focus more on the path planning, using a predefined starting location. While the robot was exploring the map during the localisation, it might collide with the walls due to the size of the robot. Therefore, a collision scan was implemented. Every time before the robot moves during exploration, it scans a portion of the environment in front of it, and selects the minimum distance from the readings and times a factor to limit the moving distance and avoid colliding with the walls.

Inaccuracy in the localisation is suspected because of the readings from the ultrasonic sensor and the inconsistency of the starting position of the ultrasonic seonsor, which creates sensor and turning noise and affects the localisation.

5 Path Planning

The known environment operated by the robot is defined by a map, the map used for this project is based on a vectorial map. The map can be analysed to determine how the robot should behave once it knows its location on the map.

5.1 A* Algorithm

Path planning between start and target location can be achieved using the A* algorithm. It uses best-first search method based on the cost function $f(x) = g(x) + h(x)$ where $g(x)$ is the cost of path from the start node to current node x, and $h(x)$ is the heuristic that estimates the least cost to target. It expands and updates values of the neighborhood step by step from starting node towards the goal node. In this way, the algorithm return the waypoints of the best path between start and goal.

To implement the A* algorithm, the vectorial map needs to be converted to a binary occupancy grid. The origin point in these two representations are different and need to be considered (vector map start at [0,0], binary start at [1,1]). After the grid map is created, goal register map and function maps need to be created in the same dimension. In the goal register map, the goal node needs to be set with value of 1. In the grid map, all the nodes that representing walls need to be set with value of 1. All other empty area need to be set with value of 0. To inflate the grid map, each of these nodes of walls will then be expanded in a 'X' shape, setting value of 1 to its neighbor nodes. (As shown in Figure 1) The purpose of an inflation map is to add a virtual boundary that helps robot to avoid collision.

The resolution of the used grid map is 1cm which is 1/10 of the original map. If the grid map resolution is increased, the planned path will be more accurate, but will also require more time for computation. The 1/10 configuration was therefore chosen. The program takes the value of the connecting distance and then generates coordinates of waypoints as output.

5.2 Probabilistic Roadmap

The Robotics package in Matlab contains a basic probabilistic roadmap path planner (PRM) [3]. The PRM operates on a Binary Occupancy Grid, which is a binary matrix representation of a map, where an occupied location is represented by 1, and a free location is represented by 0 [4]. Given the map representation introduced in the beginning of the section, the format can be converted to a Binary Occupancy Grid using the BotSim help class. By creating a new map array sized by the maximum values original map, the BotSim class can be called for each of the coordinates of the map to test if they are within the map of the BotSim, using the answer to convert the location to binary. The map can thereafter be inflated, meaning increase the simulated size of objects, to counteract physical robot size.

When a map is defined, the PRM generates a specified number of nodes throughout the free spaces in the map. Nodes are connected by an edge if there are no obstacles between them and they are within a specified connection distance. Given a start and end location the PRM can calculate the shortest obstacle-free path

[3] *robotics.PRM class*, The MathWorks, Inc. `https://www.mathworks.com/help/robotics/ref/robotics.prm-class.html` [Accessed 25/03/2017]

[4] *robotics.BinaryOccupancyGrid class*, The MathWorks, Inc. `https://www.mathworks.com/help/robotics/ref/robotics.binaryoccupancygrid-class.html` [Accessed 25/03/2017]

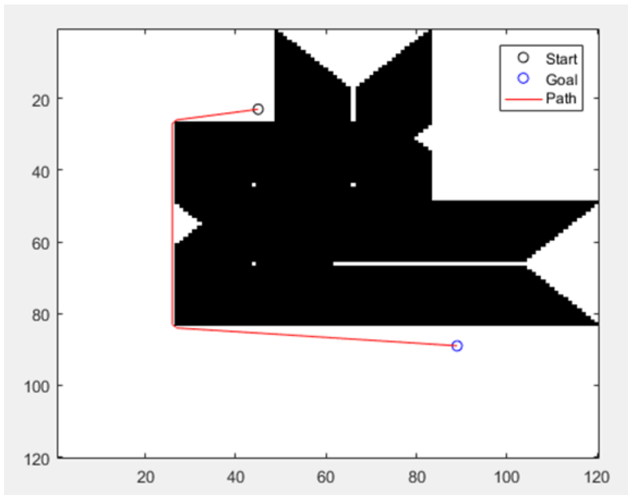


Figure 1: A* Path Planning with Inflated Map

between them. The function returns an array with the coordinates of all waypoints between the start and the goal. The result can be manipulated by changing the number of the nodes or the connection distance between them.

6 Path Movement

A straight forward principle of movement is the rotate-translate-rotate method. Meaning that rotation and translation are two separate processes, where turning ideally change the orientation without changing location, and translating changes location in the direction of the orientation without changing the orientation. It is then possible to track movement using odometry. More advanced movement principles exist, such as differential drive.

6.1 Optimise Path

The robot must regulate the speeds of its motors to turn in desired direction, and can thus not use full speed continuously. Most efficient path is therefore the shortest path from start to goal that also contains the least amount of waypoints. Since that means less amount of turns, so that the robot can keep a higher average speed.

An optimisation function was created to analyse the waypoints between start and goal, and optimised the path if possible. The optimisation loops through the waypoints and compares the distance between *current* and *next* waypoint. If the distance is smaller than a threshold, it removes the *next* waypoint from the waypoints set and continues by comparing current to the waypoint *after next*. Else, if the distance is big enough it increments both current and next position through the waypoints, start and goal waypoint always remains in the set.

6.2 Path Move

The path move function allows the robot to move a path of given waypoints by calculating the distance and optimal angle between current and next waypoint until the target is reached. One optional function in this program is to read an ultrasonic scan before giving command to move, to avoid unexpected collisions. Looking at path planning independent of localisation this might not be necessary. But if also considering localisation, sensing an obstacle in the planned trajectory can restart the localisation to improve accuracy.

7 Results

Attempt	**1**	**2**	**3**	**4**	**5**
Starting position	[22,22]	[88,88]	[40,20]	[20,60]	[80,80]
Starting angle	0°	90°	180°	45°	-45°
Target position	[88,88]	[44,22]	[20,80]	[40,20]	[20,60]
Time	8.97s	17.64s	14.91s	13.20s	13.33s
Distance from target	8.5cm	2.21cm	3.26cm	4.81cm	5.79cm

8 Discussion

Due to time and group size limitations it was decided to divide the project into localisation and path finding, and focused on one of them. It was decided to focus on path planning, since it was relatively achievable to get a path planning solution to work somewhat well, and more time could thereby be spent on optimisation and debugging. A somewhat successful localisation was still implemented, and given more time the solution could be completed and implemented together with the path finding.

By testing the two suggested methods of path planning, it was decided to use the A* Algorithm, since it gave more accurate and stable results from the start. The probabilistic roadmap was still proved working, but less stable and optimal. So instead of debugging the implementation, it was decided to focus on the A* implementation and to optimise it as much as possible.

It was decided to focus on the rotate-translate-rotate movement principle, since it was the most straight forward implementation. It was discussed to change the movement to differential drive to improve speed in the path finding, but it was decided not to, since the robot still was considered to function sufficiently well. If longer timeframe and larger operation environment it might be worthwhile to revisit discussion regarding differential drive to optimise speed further.

The optimise path implementation have a problem that might optimise away important waypoints around corners, resulting in crashes. This was considered but not addressed at the movement, since testing showed that waypoints never came that close to corners to make any risk of crash, due to the inflation of the map in the path planning.

9 Conclusion

To conclude, the robot is capable of navigating itself to a target location with a predefined starting position and orientation and successfully reaches the target location within acceptable tolerance. The key challenge here is to implement the simulation program onto a real robot and resolve issues with various types of noises. Although the localisation has not been completed and merged with the path planning, the particle filter is implemented and proves itself to be working, yet localises itself in a rather inaccurate position and orientation. The noises from turning, motion, and sensor have been minimised as much as possible by calibration. Different algorithms in path finding have been implemented and tested on the robot in order to look for the optimal solution for specific maps.