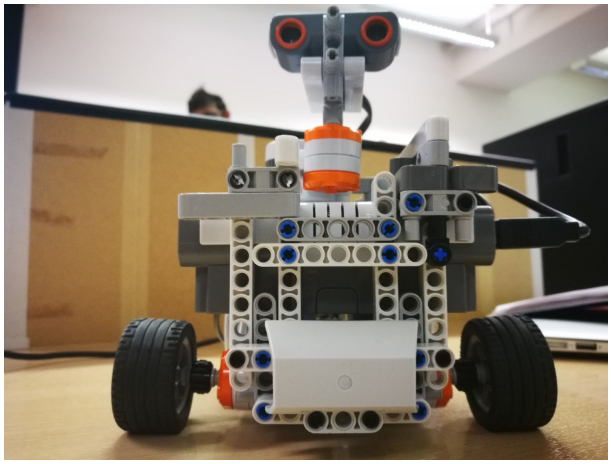
COMSM0012 Robotics Systems Coursework Two
Real Robot Localisation

Team SYFY-2



Work Assigned

Name	Task	Contribution
Guohua Fan	Robot Building; Research and Report; Robot Calibration;	**23%**
Qingqi Shi	Robot Building; Report; Programming;	**27%**
Han Yang	Robot Building; Research and Report; Robot Calibration;	**23%**
Muyu Yang	Robot Building;Report; Programming;	**27%**

1. Robot Design

The robot was built using the Lego Mindstorms kit[1] which consists of one controller brick, one ultrasonic sensor and three servo motors. These components were assembled together using standard Lego pieces to create a structurally rigid robot.

The robot adopts a two-wheel differential drive configuration, in which two drive wheels are separately controlled by two motors. In addition, there is a caster wheel which is utilised to reduce friction and increase stability. The controller is secured in between the wheels with the ports facing the left and right sides, reducing the area footprint of the robot. The ultrasonic sensor is set in the center of the two wheels, this allows more accurate readings as the robot moves around.

The robot was programmed using Matlab and the commands are sent to the robot through the RWTH Mindstorms NXT Toolbox[2]. This toolbox contains functions to manipulate the motors and sensors. A 'Robot' class was written in Matlab to encapsulate low level motor commands into high level motion commands which mimics the BotSim library from the previous coursework. Some of the functions from the class are described below.

Move Function

The move function is used to control the wheel motors in order to drive the robot forwards and backwards. To drive in a straight line, the two wheels need to be turned in the same direction. In this function, the distance to move is converted to taco limit which determines how much the motors spin. The power parameter is also set to make the robot move faster. The taco limit is also calibrated as described in a later section.

Turn Function

The turn function is used to control the wheel motors to turn the robot through a certain angle. The angle is specified using radians to match the BotSim library. In order to rotate the robot about the centre, the two wheel need to be turned in the opposite directions to each other. A formula (*Formula 1*) was devised to convert the angles in radians to the taco limit using two parameters: the distance between the wheels and the circumference of the wheels.

$$tachoLim = \frac{angle \times interWheelDistance \times 0.5}{wheelCircumference \times 360} \qquad \textit{Formula 1}$$

Scan Functions

There are two scan functions, one for taking an ultrasonic scan using the sensor and returns a calibrated result, while the other one is used to control the sensor motor and rotate the sensor so that readings from different directions can be taken. To rotate the sensor, a delta angle was calculated and the motor turns the sensor a specified number of times. Only then, the motor rotates back to the original position.

Problems
Multiple problems were encountered during the design and coding phases.

The structure of the robot was modified dozens of times. Each time discovered and solved a different problem. For example, the robot was very large at the beginning, this results in a very high rate of collision; also, the robot once had the ultrasonic sensor off-centre, which resulted in inaccurate localisation; the cables were once a problem too, which created much more friction when pulling on the walls. In the end, the robot took on a form that is rigid and stable.

Multiple problems were also encountered when coding. At first, each motion command was written in a separate function which resulted in the need to pass many variables around as arguments, this was solved by combining the functions into a single class and the variables became the properties of the class. Another problem we encountered was the reset of the sensor motor, which is that inaccuracy accumulates as the ultrasonic sensor rotates around. This was solved by resetting the position of the sensor motor at the construction of the Robot object, and after each round of scanning, the motor would read the state of the motor, and turns exactly back to the initial state, minimising the accumulation of inaccuracy.

2. Calibration

Accuracy in the movement and scanning was crucial to solving the localisation problem. Therefore, a series of experiments were conducted to measure and calibrate some of the systematic errors of the robot. The results from this also helps to get a feel for the amount of noise that is required.

Move Calibration
To calibrate the movement, we used a simple approach. First the wheel motors were set to turn a certain amount (using the taco limit), and the actual distance traveled in centimeters were then recorded. This was repeated for different taco limits and each repeated five times, and the results are as follows (*Table 1*).

Table 1

	180	**360**	**540**	**720**	**900**	**1080**	**1440**
1	6.6	13	19.2	26	33.1	39.2	52.6
2	6.4	13.2	19.3	26.4	32.6	39.4	51.8
3	6.4	13.2	19.6	26.3	32.5	39.2	52.4
4	6.4	13.1	19.6	26.2	32.7	39.3	52.6
5	6.5	13	19.5	26.2	32.4	39.4	52.4

Using this data, a scatter diagram was plotted and a trendline was generated. The equation of the trendline was then used as the basis of our calibration equation (*Formula 2*).

$$y = \frac{x+0.0929}{0.0364}$$ *Formula 2*

, where x is the distance to move, y is the tacho limit for the motor

Turn Calibration

As mentioned above, a simple formula (*Formula 1*) that takes the inter wheel distance and wheel circumference as parameters was derived as a naive approach for the turn command. However, since the wheel has width the actual contact point with the ground cannot be accurately measured, and hence the requirement for making further calibration. A method was devised to first command the robot to turn a specified angle of radians for a number of repeats, then the actual amount the robot had turned was measured using a protractor and recorded as below (*Table 2*).

Table 2

	Actual (* 4)(degrees)	**Actual (average)(degrees)**	**Actual (radians)**
pi/4	159	39.75	0.693768
pi/2	314	78.5	1.370083
3*pi/4	467	116.75	2.037672
pi	623	155.75	2.71835
-pi/4	-158	-39.5	-0.68941
-pi/2	-312	-78	-1.36136
-3*pi/4	-461	-115.25	-2.01149
-pi	-624	-156	-2.72271

Using this data, a scatter diagram was plotted and a trendline was generated. The equation of the trendline was then used as the basis of our calibration equation (*Formula 3*).

$$y = \frac{x - 0.0044}{0.8649} \qquad \textit{Formula 3}$$

, where x is the angle in radian desired to rotate, y is the actual angle used

Sensor calibration

The sensor can be used to measure distance by sending out ultrasonic pings and listens for the reflected echoes. To calibrate the ultrasonic sensor, the sensor was placed at different distances away from the wall, and the readings from the sensor were repeatedly taken (*Table 3*).

Table 3

	10cm	**15cm**	**20cm**	**25cm**	**30cm**	**35cm**	**40cm**	**45cm**	**88cm**
1	13	19	21	26	30	35	41	46	88
2	13	18	21	26	30	35	41	46	88
3	13	18	21	26	30	35	41	46	88

One observation was that the readings above 25cm are very accurate (∓1cm), therefore calibration is only needed for readings below 25cm. A scatter diagram was plotted for those data and a trendline was generated. The equation of the trendline was then used as the basis of our calibration equation (*Formula 4*).

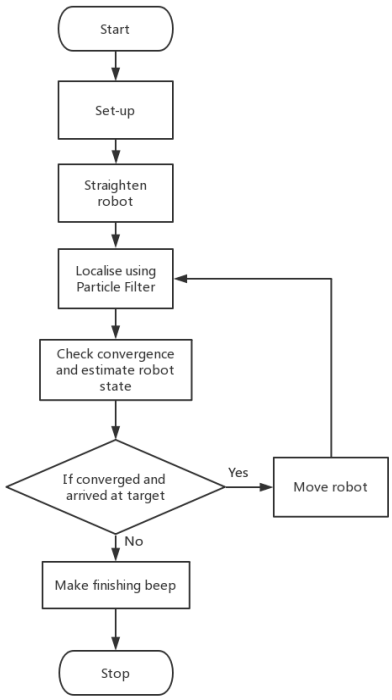
$$y = \frac{x-5}{0.8333} \mid x < 25$$ *Formula 4*

, where x is the sensor reading, y is the actual distance in cm

It was also observed that the ultrasonic readings become inaccurate as the angle between the sensor and the wall increases, at about 150 degrees the readings become completely incorrect. One possible explanation for this error would be that the ultrasonic ping gets reflect away rather than towards the robot. This observation was later used to modify our solution.

3. Solution

The kidnaped robot problem was solved by combining the Particle Filter localisation method[4] with the Wavefront path planning algorithm[3]. The entire process can be illustrated using the flowchart on the right. Our approach differs from many of the others by keeping the localisation running while path planning at the same time. This incorporates the error in movements into account and allows the localisation algorithm to detect if the robot starts to go off-course.



Set-up

The program is first set-up by initialising particles at random positions and creating the Robot object which would open the connection with the NXT controller brick and sets parameters for the motors and sensors. This step also includes the pre-calculation of the wavefront grid map.

Straighten robot

The robot tries to straighten itself before entering the localisation loop so that it is perpendicular or parallel the walls. This will help ensure the accuracy of the ultrasonic sensor. To do so, the robot scans all around it self, recording the direction of the shortest distance, and aligns itself to it - most of the times, this makes the robot directly face a wall. Problems arise when the robot is near corners, as there is no direction perpendicular to the corner point. To resolve the corner problem, the robot then makes a turn of 90 degrees to the left or right, then moves a fair amount. This puts the robot away from the corners, and the robot can then scan again to align itself up to a wall.

Localise using Particle Filter

The program then enters the Particle Filter loop. This includes commanding the robot to take scans of its surroundings, use the particles to take virtual scans and assign weights to each particles and resample the particles using the weights.

As observed in the previous section, the sensor is only suitable for measuring the distance of walls perpendicular to the robot. Therefore, the robot has been instructed to take only four scans - in front, behind, left and right. Assuming that the robot is always perpendicular or parallel to the walls, this allows the scan readings to be accurate.

Check Convergence and Estimate robot state

After resampling, the probabilities of the real robot position is encoded in the states of the particles. Therefore, using the particles, the real robot's position and rotation can be estimated as the average (used median) of the particles. The accuracy of the estimation can also be calculated using the spread of the particles (used interquartile range). If the spread of the particles is less than a defined threshold and the estimated position is within a threshold radius of the target, the program can terminate with confidence that the robot has reached the destination.

Move robot

It is important to move the robot so more data can be gathered in the next iteration. The program moves the robot in different ways according to the current localisation state.

At the beginning, when the particles never converged, the robot has no idea where it is. Therefore the move instructions follow a 'blind moving' strategy. In this strategy, the robot move forward if it has the space to do so, and turns to the left or right if it senses empty space in that direction. This strategy allows the robot to traverse the map and allows the particles to converge quicker.

If the particles start to converge, the robot then moves by following the wavefront grid map. The wavefront map is consisted of increasing numbers radiating out from the target, and therefore to move to the target the robot follows a path that leads to smaller numbers. It stops moving when the robot is about to turn, and allows the robot position to be updated.

In the scenario that the particles have converged, and that the estimated position of the robot is within a defined threshold to the target, the robot would turn to face the target, and move straight to it. It would then make a final check of its position, and ends the program if nothing has gone wrong.

References

[1]Lego.com. (2017). *Home - LEGO.com*. [online] Available at: https://www.lego.com/en-gb/mindstorms [Accessed 28 Mar. 2017].
[2]Mindstorms.rwth-aachen.de. (2017). *RWTH - Mindstorms NXT Toolbox*. [online] Available at: http://www.mindstorms.rwth-aachen.de/ [Accessed 28 Mar. 2017].
[3]Ghai, B. and Shukla, A., (2016) ' Wave Front Method Based Path Planning Algorithm for Mobile Robots', *Swagatam Das Editors*, p.279.
[4]Rekleitis, I.M.(2004) 'A particle filter tutorial for mobile robot localization', *Centre for Intelligent Machines, McGill University*, *3480*.