Robotic Systems 2013-14

Coursework One

Team Lassie

MR HARRY BLOXHAM (50%)

Particle filter, distance and turning calibration, developed simulator, co-developed robot program

MR PIETRO CARNELLI (50%)

Navigation system, ultrasonic sensor calibration, written report, co-developed robot program

Contents

Introduction

A localisation and navigation algorithm for a Lego NXT robot was designed, assembled and developed in order to find its way to a target within a defined arena. The Lego NXT kit was kindly provided by the University of Bristol and the NXT firmware 'MotorControll22' developed by the University of Aachen was used in order complete this assignment.

Aim

To assemble a Lego NXT robot, develop a localisation and navigation algorithm for it to localise itself within an arena and reach a target location both in real life and within a simulated MATLAB environment.

Physical Design Requirements

1. Only a maximum of three motors can be used for the robot
2. Only one ultrasonic sensor and a simple push switch may be used
3. The NXT brick must run the MotorControl22.rxe firmware
4. Only Lego parts provided within the kit may be used for the assembly of the robot
5. The robot must be small enough to pass through a 40cm opening within the arena

Program and Modelling Design Requirements

1. Simulator may not use actual location coordinates of modelled robot
2. Sensor and movement error does not need to be included

Performance Requirements

MATLAB Simulation

1. Simulator program must be able to solve the modelled problem in less than 120 seconds
2. Final simulated robot location must be within 30 cm of the final target position

Physical Experiment

1. Reach the desired final target location within 150 seconds
2. Reach the desired final target (assuming on time) within 20 cm of it

Robot Design

The robot design went through a couple iterations before the final design was settled upon. Final design features are explained and shown (figure 1) below:

- *Low-slung chassis:* In order to mount the ultrasonic sensor below the height of the arena walls such that 360 degree scans (distance measurements) could be achieved. This not only achieves better estimates of location (when using a particle filter) but also reduces turning of the robot which takes longer and has more error associated with it.
- *Central mounting of ultrasonic sensor:* This was achieved by moving the rotating 'arm' and attaching it across the front of the robot. The ultrasonic distance sensor was mounted

directly above the rotating axis (wheel geometry) to further improve accuracy without unnecessary programming corrections

- *Strengthening beams:* In the first iteration a weakness was noticed, whereby the wheels either side of the NXT brick would often come apart if the robot crashed into walls and or was not set down gently. An extra rear brace was added to provide support and robustness.
- *Neat wiring:* A lot of problems with 360 degree scanning were down to poor wire routing for the sensors and all the motors. Eventually the NXT brick was re-orientated and the wires were put together with the chassis so that they were tucked away.
- *Two wheel drive:* Two wheels in parallel were chosen with a third castor bar (for support only) as the drive system due its simplicity to model and navigate both in the simulator and the real robot. It was decided after some experimentation that a simple bar with a ball joint at the end would suffice as a castor wheel, as it didn't change the movement accuracy and it reduced the overall size of the robot.

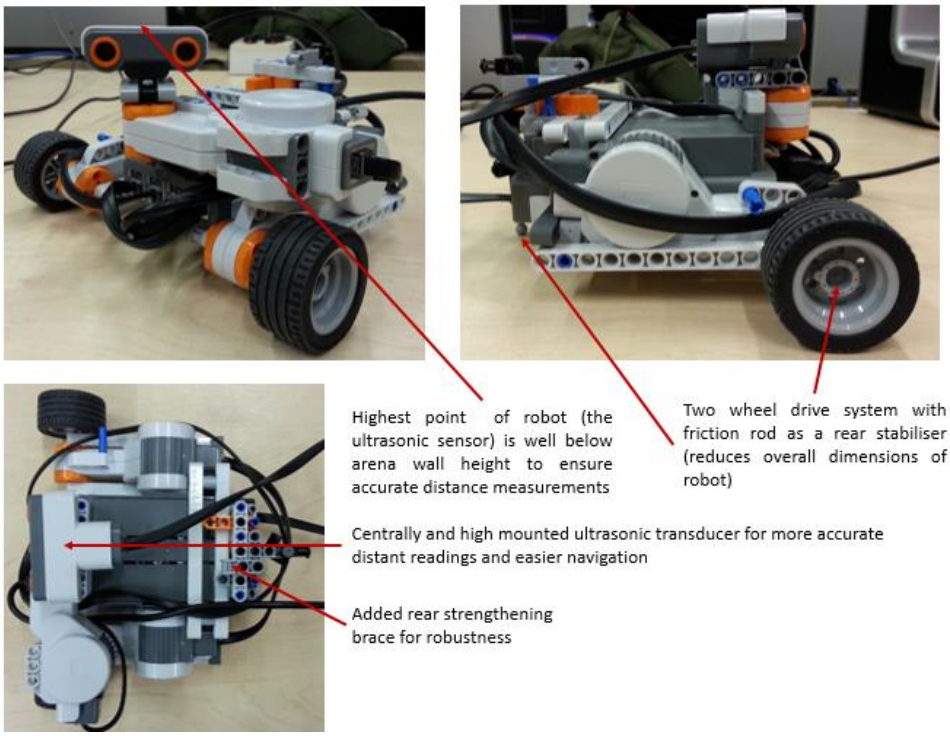

Figure 1. Images to show final design of Lassie (the robot). Key areas such as overall dimensions, location of ultrasonic transducer and drive system have carefully been optimised over the course of the project.

Calibration

In order to better understand the reliability of the ultrasonic and rotation (tachometer) sensors brief experiments were designed and carried out.

Movement

A straight line test arena was constructed in order to measure how well (in a straight line) the robot could move to a chosen position. After changing the relative power sent to each wheel during a straight line movement operation (in order to compensate for the mismatch between the motors) the robot was able to reach and stop at targets within 1cm.

Turning

Another experiment was devised by printing an A3 protractor and placing the robot on it. After several trials and corrections of the power ratios between the motors, the robot was able to turn within 5 degrees over a 360 degree target rotation.

Ultrasonic Sensor

An ultrasonic distance sensor was used to localise the robot. It allows for rapid 360 degree scanning of distances in order to compare them to results generated by a particle filter. The ultrasonic sensor was tested for its ability to measure distances as well as its ability to cope with being at an angle from a wall which was being used for distance measurements.

Distance

In order to evaluate the error and accuracy of the ultrasonic sensor in measuring direct distance, the sensor was placed directly in front of a wall and moved backwards at regular measured intervals. At each interval 1000 measurements were taken. The results of the experiment are shown below in figure 2. It is worth noting that the ultrasonic sensor was very good at estimating distances over 30cm (mean error was less than 1%). It is clear that the sensor generally overestimates, this is further highlighted in figure 2.b where the mean error was positive (1.8cm). The error also tends to get worse at distances very close to the wall.

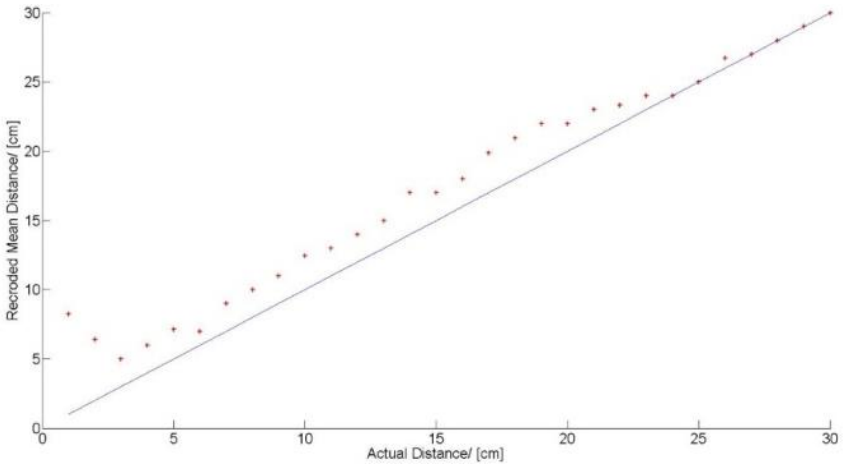

Figure 2. Graph showing the average of 1000 measurements at 1cm intervals between 1 and 30cm (red points, blue line is actual distance) perpendicular to a cardboard wall.

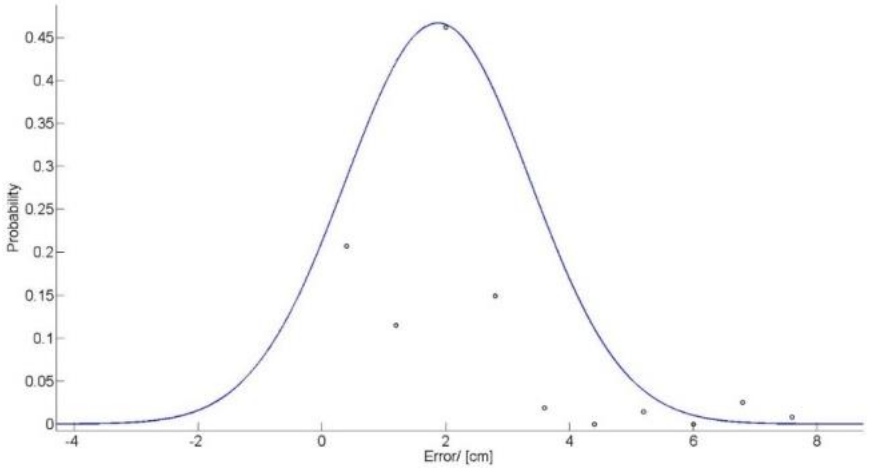

Figure 3. Graph shows the average error of measuring distance (black points) as well as a fitted Gaussian distribution (blue line).

Angle

The ability and accuracy of the sensor at measuring distances to an oblique wall were evaluated. Below in figure 4 are the results of the mean measured distance versus angle of sensor orientation to the wall. Note that the wall was directly in front of the sensor at 0 degrees and 1000 measurements were taken at every 5 degree increment. The ability of the sensor to correctly predict distance decreases as the angle was increased in either direction. However, excellent results (within 2% error) were achieved for the range between -25 and +25 degrees. The measured distances actually improved at around ±60 degrees before they became completely unreliable.

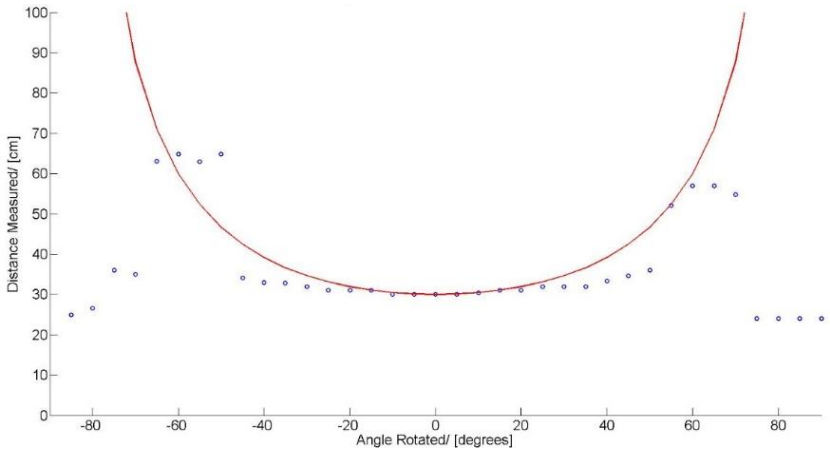

Figure 4. Graph showing theoretical correct measured distance (red line) as well as the measured distance by the ultrasonic sensor.

Navigation System

After some research it was decided that a visibility based navigation system was to be designed for this project. Essentially nodes are placed at vertices of a defined boundary, in this case the modified map (red lines figure 5 below). Lines are then drawn outwards form the points until paths are formed where they cross. Then a function determines the shortest route between the start (green dot) and end (red dot) targets. Four different maps were developed and the output navigation paths

are shown below. One issue later addressed with the real robot tests was finding navigation paths when starting on a boundary line. This was solved by introducing a check that would force the robot to move backwards slightly into the modified map (red boundary) before re-running the navigation algorithm.

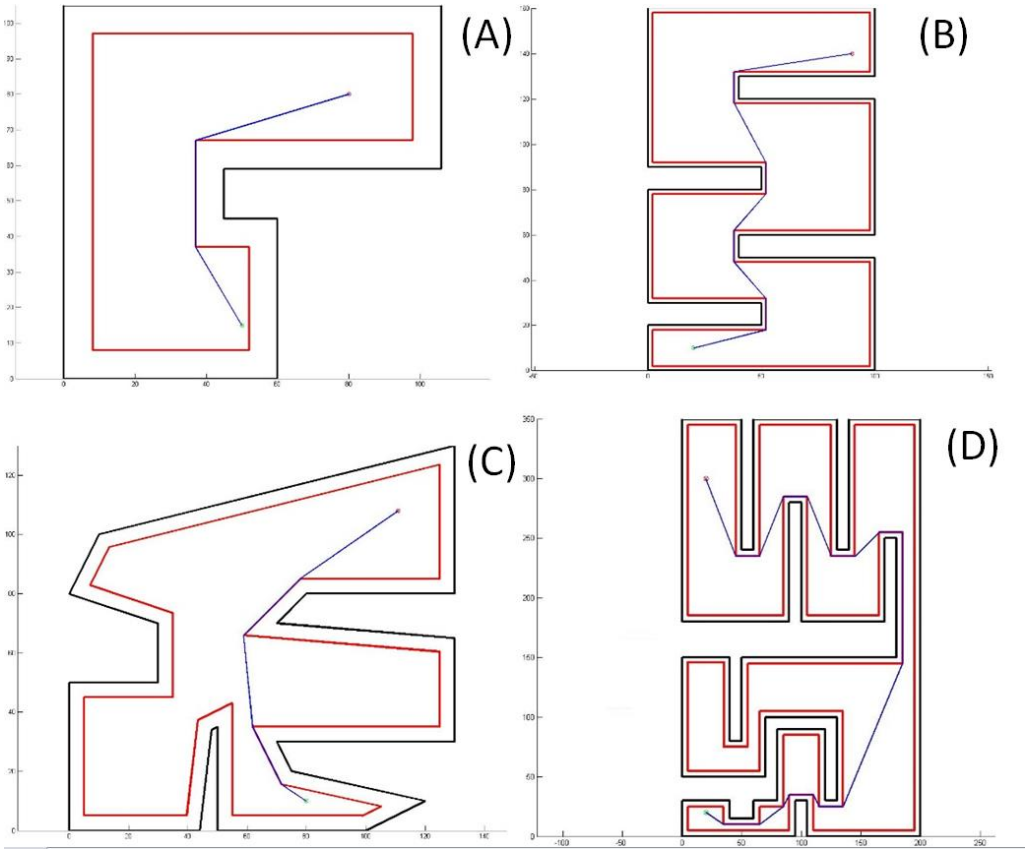

Figure 5. Shows three different maps and four different navigation problems. (A) simplest map, the given one in the test labs (B) more complex, rectangular arena, (C) an oblique shaped map (walls are no longer always perpendicular to each other) and (D) the most complicated map (in terms of navigation at least).

Particle Filter

To determine the robots location within the map, a particle filter was used. This method of localisation selects a large number of randomly positioned and orientated virtual robots (see figure 6a below). The number of particles is based upon the area of the arena; roughly one particle per $4cm^2$ gave the best compromise between computational speed and accuracy. The program then simulates what measurements the real robot would make if it were in the same location. These virtual measurements are then compared with the real measurements and an estimate of the real robot's position can be made.

For the simulation and real robot, ten distance measurements were made at 36 degree intervals to allow a good estimate of the robots full surroundings. It was found that more measurements increased the computational time with a very small increase in accuracy. Fewer measurements meant that some features of the map might be missed and any erroneous readings would have a

larger affect. The probability of the simulated distance being the real distance (given the distance measured) from the real robot was calculated using the following equation.

This was done for each of the 10 measurements per particle and their product calculated to calculate the probability of a given particle being at the robots real position. The MATLAB *'circshift'* function was used to determine which orientation gave the particle the highest chance of being correct. This meant that each particle had 10 possible orientations, which increased the speed and precision of the system. Each of the particles probabilities were then used to assign a weighting to each result. The particles were then redistributed according to these weightings with more particles being assigned to the location and orientation of the highest probability particles (figure 6b). The robot was then moved according to the navigation program and each particle would also be rotated and driven the same distance. However, random noise was added to the particles movement and rotation to simulate the error associated with the movement of the robot (see figures 6c and 56). This randomness was a mixture of noise proportional to distance moved and angle turned and a fixed amount of noise to spread the particles while maintaining the same mean position. To localise the robot initially, the robot was moved in the direction of the largest measurement by a small distance. This process was repeated until the standard deviation of the spread of the particles was sufficiently low. The navigation program was then implemented to drive the robot to the target location. At every turn point or after a maximum of 40cm moved in a straight line, the robot stopped to take a new set of ultrasonic measurements and the particles resampled to localise the robot.

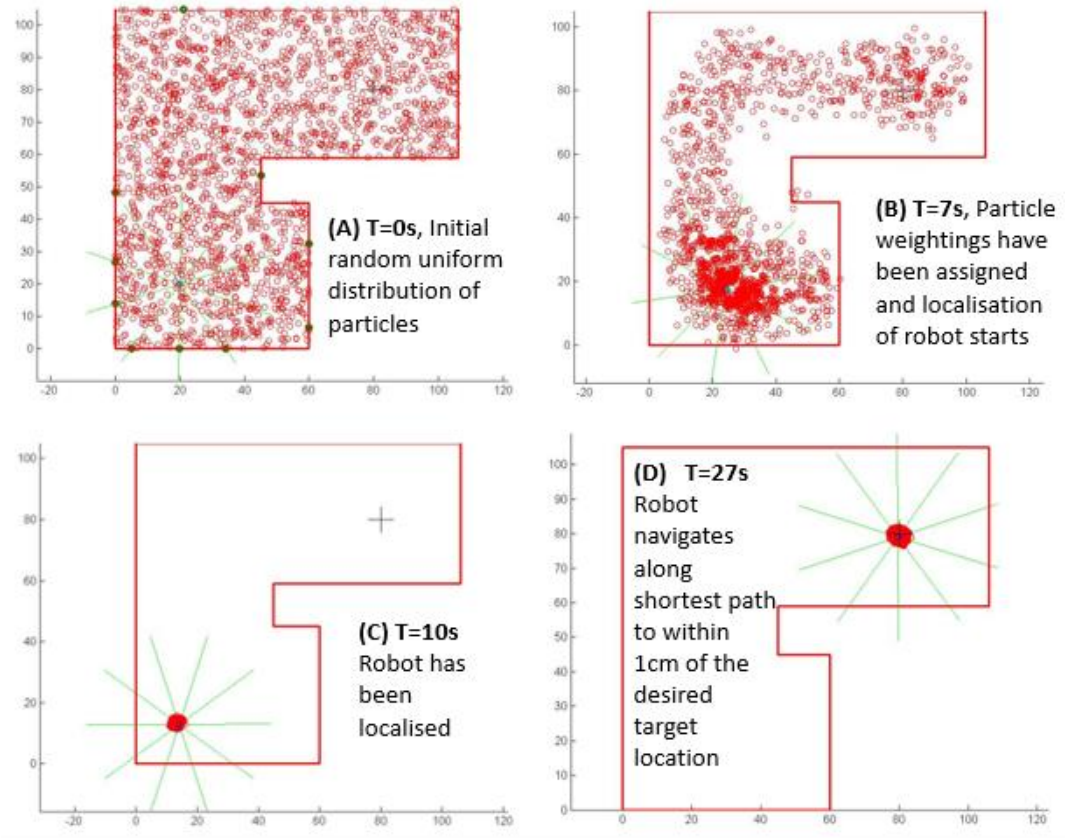

Figure 6. Graphs showing select stages (T = time in seconds of the simulation running) of the particle filter localisation and navigation systems; from (A) where the particles are uniformly distributed across the map to (D) where the robot has been localised and has navigated towards the target.

Robot Control

In order for the robot to move between targets along the navigation path towards the final desired destination, two functions had to be developed in order to determine a) in which direction should the robot turn in and by how much and b) how far should it move along that determined direction.

Turning

A function was written whereby the inputs required to calculate the shortest angle of rotation where the co-ordinates of the next target location, the current orientation and location of the robot. Two vectors were compared to a base line axis (the x axis in this case) in order to define and measure angles. The system relied on the use of the '*atan2*' function within MATLAB which allowed for quadrant specific angle (in degrees) measurements. Figure 7 below shows how the angles were defined and evaluated. By using this system, the robot never had to turn more than 180 degrees.

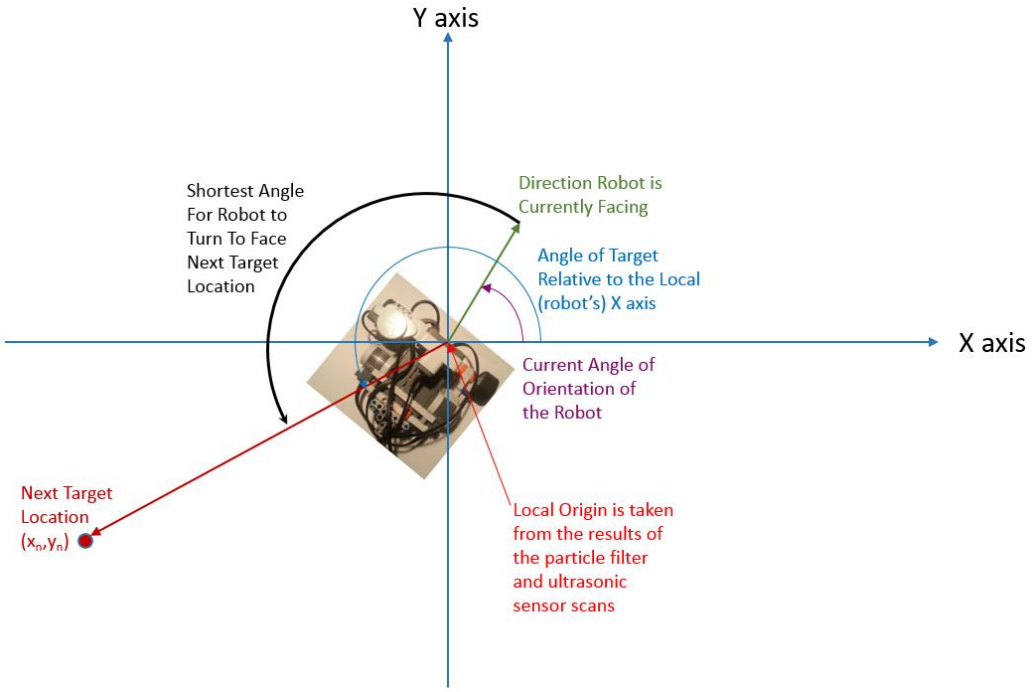

Figure 7. Diagram shows how the shortest angle of rotation is calculated and defined in terms of the target location, the x axis and the robot's current orientation and position. The two vectors representing current robot orientation and position of the next target are shown in green and red respectively.

Movement

Another function was written in order to calculate the distance (and therefore number of robot wheel rotations) required to move the robot towards the desired target location based upon the distance away and the calibration results from the previous section. A limit of 40cm movement was imposed. This meant that the robot could never move further than 40cm without having to re-scan the area and use the particle filter to localise itself. This allowed for better accuracy of movement since rotating the robot introduced the most error.

Conclusion

Simulator

The robot within the virtual environment was able to locate and navigate the robot (taking also into consideration the robot's size and therefore modified map boundaries) regularly within 30 seconds. Different maps were also used (see figure 5) in all cases the virtual robot was able eventually to localise and navigate its way to the desired target location.

Real Robot

Only one test map was available to carry out tests (see figure 5a). There was a lot of difficulty in obtaining accurate and reliable distance data measurements. It was also necessary to change the robot configuration several times in order to prevent the sensor from sensing the robot itself. Further improvements to the design were carried out, for example assembling bumpers to allow a minimum distance between the wall and the sensor.

Eventually Lassie (our robot) was able to localise and navigate its way to a desired location within the test arena in around 3 minutes.

Final MATLAB Code

For sake of brevity and conciseness, the final code for the robot was not included within the report. However, a simplified flow chart, highlighting the key features and processes within the code is included in figure 8 below. The process starts with the user entering the vertices of the arena (map) and the final desired target location. The program then takes over iterating between calculating the probability of the robot being at a location (particle filter) and navigating along a path in order to arrive at the desired location. Eventually the program realises (through checking at every iteration how far it thinks the robot is from the target location) and stops the robot when it is within five centimetres of the target.

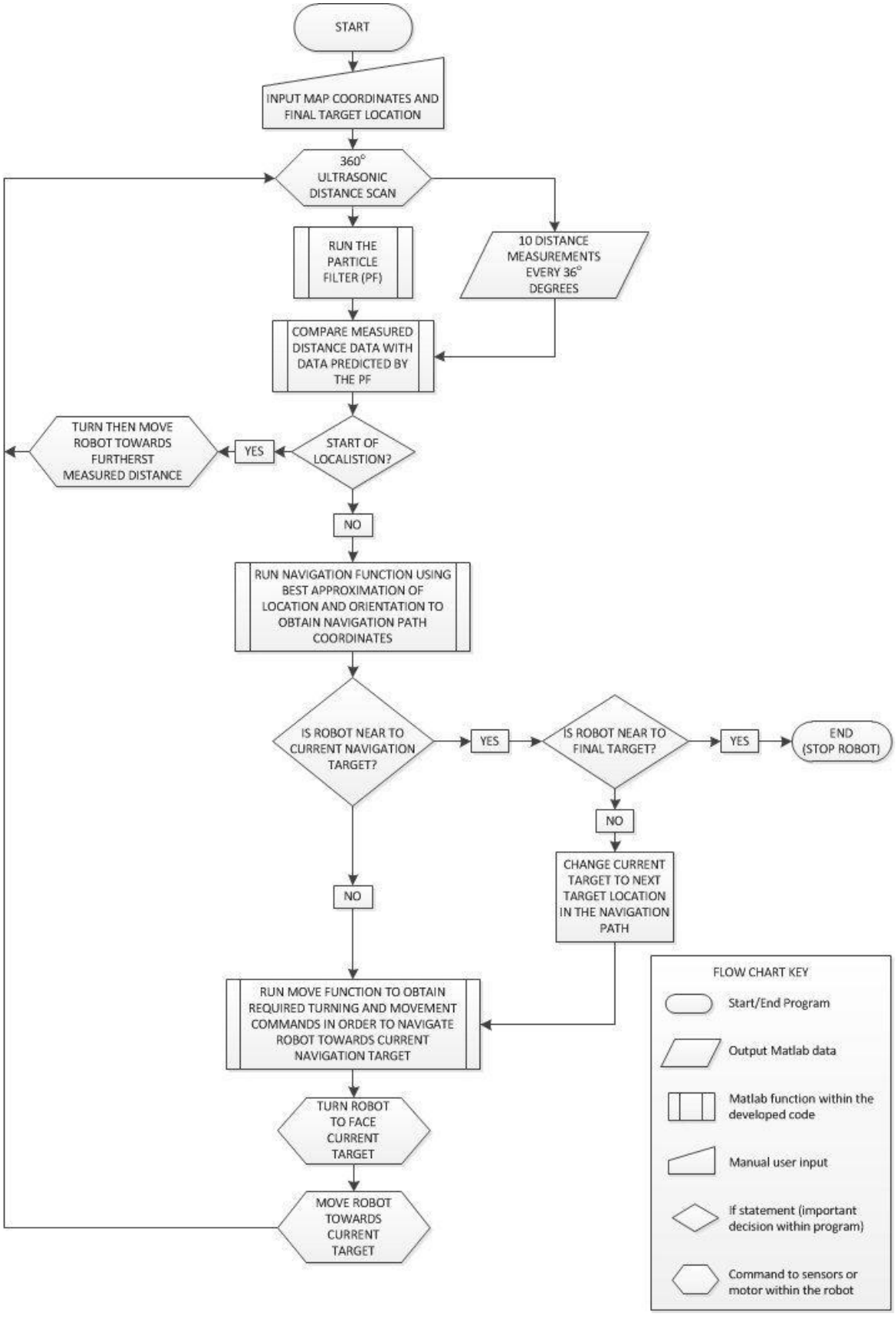

Figure 8. Flow chart showing simplified program used for the localisation and navigation of the robot.