Robotic Systems Coursework 2

Team: SKCR-1

April 24, 2018

Member	Contribution
Stan McCarthy	25%
Kipp McAdam Freud	25%
Charlie Kruczko	25%
Rizwan Ur Rehman	25%

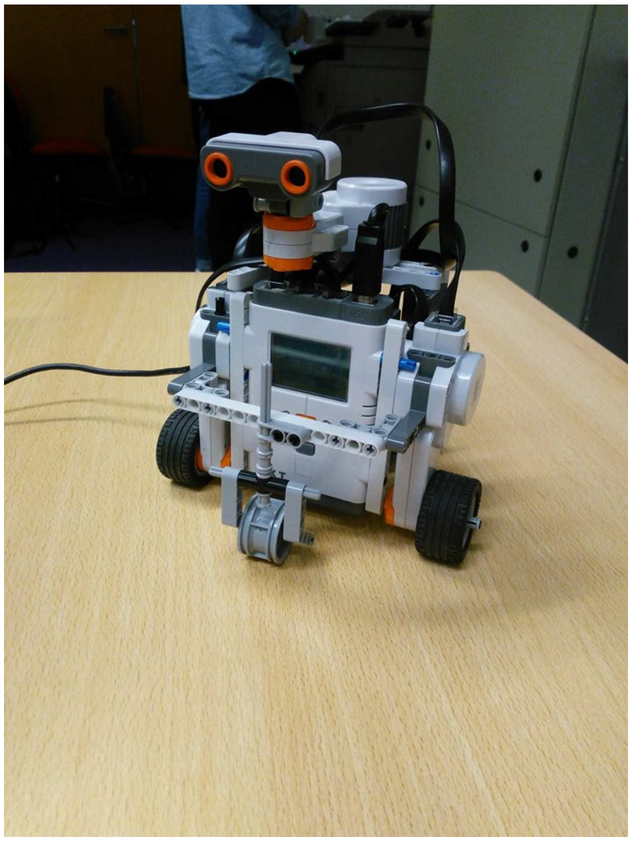

Figure 1: *Our robot*

1 Creating the robot

1.1 Building

Initially, the robot had to be designed and built. In its first iteration, the robot had two motorised front wheels and one caster wheel at the rear. The robot moved well in this setup, however when turning, the rear caster wheel meant that the back end of the robot swung out. It was decided that this could easily cause unnecessary crashes when navigating close to walls, and hence the design was altered. In this new design, the motorised wheels are aligned on the central axis of the robot, with caster wheels placed centrally at the front and rear of the robot. This allows the robot to turn on a sixpence, reducing the risk of collisions. This new design became the final configuration of the robot, as seen in Figure 1.

Additionally, efforts have been made to increase the rigidity of the robot's core structure between the motorised wheels. This is in an attempt to minimise motion and turning noise from flexibilities in the robot's structure, which should in turn allow for more reliable modelling of the motion. In the same vein of thought, the majority of the weight of the robot is positioned over the axis of the motorised wheels to reduce the risk of wheel slip, which contributes to turning noise. In contrast to the structure around the motorised wheels, the caster wheels have been attached in a way that allows for a large amount of flexibility. This gives the caster wheels something akin to suspension when travelling around the map, reducing the risk of an uneven surface causing the robot to lose a point of contact with the floor (particularly one of the motorised wheels), which results in wildly unpredictable motion and almost certain loss of localisation.

The ultrasound sensor is placed atop the robot, mounted on an NXT motor. This allows the robot to take a scan of its surroundings without turning the whole robot, making it fast and energetically efficient. The scanner is positioned centrally on the robot, on the line between the two motorised wheels. Given the symmetrical design of the robot, this means that the sensor is on the axis of rotation of the robot, resulting in more straightforward mathematical calculations of the position of the robot.

1.2 Robot API

The NXT Robot Matlab Class. To simplify the software design for the rest of the project a robot API was created using a MATLAB class. Upon initializing the robot class the robot would first open a connection to the NXT, it would then load the calibration variables from the 'calibration.mat' to the robot properties. These variables describe the relationship between NXT motor commands and distance moved or angle turned. There is also an individual calibration for the motors of the left and right wheels, this was useful when the robot weight further from the center of the wheelbase but is no longer needed. After loading the calibration variables the last part of intialisation is opening the ultrasound sensor and turning the ultrasound sensor by 20 degrees from straight ahead. The reason for this is described below. In a similar manner to the initialisation there is also a close command that resets the scanner position, untwisting the wire if needed then closes the robot connection. The close command was often called during testing if a 'Try' block failed, calling the Close function upon catching the error.

Movement functions. There are two functions which move the robot, a forward movement function and a turn function. We opted to avoid moving the robot along curved trajectories as this we felt would add unnecessary noise to the motion for a minimal speed increase, not to mention a large increase in software complexity. The move function can take a positive or negative distance to move in cm. The turn function can take a positive or negative turning quantity in radians. If the turn is over pi radians the robot will move in the opposite direction instead to save time and increase accuracy of the movement. Below is an example simple script to execute some movement commands for the robot

```
nxt = Robot() % initialises the robot
nxt.move(10) %move forward 10cm
nxt.turn(pi/2) %turn anticlockwise by pi/2
nxt.move(-5) %reverse 5cm
nxt.close() %close the connection with the robot
```

Figure 2: *Simple movement commands for the Robot*

Scanning. In order to scan the scanner starts with an offset of about -20°. The ultrasound scanner then continuously rotates 400° (-20° to 380°). The reason for this rotational offset is due to the motor than rotates the scanner having a small amount of uncontrollable wobble giving the start and end positions some innaccuracy. The robot then takes ultrasound readings in the region of $0° \geq scans \leq 360°$. The scans are evenly spaced as dictated by the number of desired scans, we found 32 readings to be appropriate for use in localisation and checking position following movements. When the scanner moves -400° starting at 380° we took readings in the $360° \leq scans \geq 0°$ region. Each sequential scan is in the opposite direction of the last to prevent the scanner data wire from tangling, by being able to take scans in either direction (then standardising the returned vector) we save time and battery by not having to repeatedly reset the scanner position.

Calibration. The movement produced from NXT motor commands depends on the anatomy of the robot and the surface upon which the robot moves. Thus within the robot class is a calibration function for the robot forward distance, angular deviation in forward movement and turning angle. The robot is placed in the arena it will be tested in then the calibration is called. A ruler is not needed as the calibration relies on the tiles of the arena floor. The forward distance is measured by attempting to move 22cm, the length of one of the square tiles. The user can then observe the movement between tile edges and report 'under/over/correct', the robot continuously makes movements, and increments or decrements a calibration variable until the robot can move 22cm accurately. A similar 'under/over/correct' paradigm is used for a 90°turn on between tile edges and a 'left/right/straight' paradigm is used to adjust the power of the left and right motors if needed. At the end of this function the variables are saved to 'calibration.mat' that is read on the initialization of the robot.

2 Localisation

The strategy used for localisation of the robot was designed to closely mirror the strategy used in CW1, with a few changes made to accommodate the imperfect nature of working with hardware as opposed to simulation. Most notably, noise in sensor readings and movement noise had to be factored in far more heavily than had been in simulation. We used 2000 particles in a Bayesian Particle Filter for the localisation. This is a large number of particles (particularly for a map of these dimensions) and this does result in a noticeable computation time after each scan during localisation, however given so many particles the robot is often able to localise after a single iteration with a high precision. On balance, this was deemed to be the preferable option when compared with reducing the number of particles in order to reduce computation time, and localising after a greater number of iterations.

During testing of the NXT ultrasound scanner's accuracy, it was concluded that the only readings which were truly reliable were those which were orthogonal to the walls. Hence a method of finding the scan readings which were orthogonal to the walls was devised. Upon the initialisation of localisation, the NXT

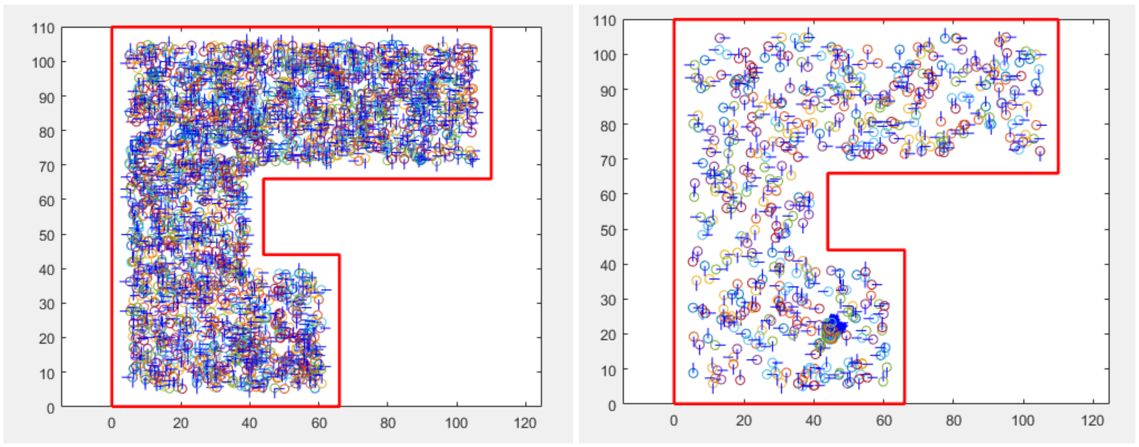

Figure 3: *The particle robots at initialisation (left), and at convergence (right)*

robot takes a scan of its surroundings containing 72 distances. From here, we utilised the mathematical fact that the shortest distance from a point to a plane occurs along a vector orthogonal to the plane. From this, we are able to infer that the shortest scan reading taken by the NXT must lie along a vector orthogonal to a wall of the map. Given the layout of this map, from here it can be assumed that scans at 90°, 180°, 270° rotation from the shortest scan line are also orthogonal to walls, and therefore reliable.

From here, the real robot's orthogonal scans were then compared to the particle's orthogonal scan readings, each particle's readings was circularly shifted by one position 3 times, with the minimum difference being logged for each particle; this made the similarity between scans of the bot and the particles invariant to the orientations of either. The approximate probability that the particle was at the same position as the true robot, given their scans, was calculated, as was the number of shifts required to achieve the highest probability. The particles were each rotated by a angle corresponding to this number of shifts, turning to the angle deemed "most probable" to be similar to the robot angle:

```
for shift = 1:(size(scanLines,1)-1)
        disparity = mean(abs(circshift(distances(:,i),shift)-botScan));
        if (exp(-(disparity^2/(2*sensorNoise)))+dampeningFact)>
            particleProbs(i)
                particleProbs(i)=exp(-(disparity^2/(2*sensorNoise)))+
                    dampeningFact;
            particleShift(i) = shift;
        end
end
```

Figure 4: *Logging maximum particle probabilities, and the angular shift required for each particle to achieve this probability*

Once these probabilities have been logged, the vector of probabilities was normalized, and a cumulative sum vector was taken. This transformed the vector of probabilities into a cumulative distribution function. particles were redistributed according to this function; in short, particles were more likely to be redistributed to areas near particles with high probabilities, and with angles similar to these particles.

The convergence was set to be true if the product of the standard deviation of the particles positions was less than 8, this seemed to ensure that the particles had localised into a single cluster, but did not impose a stringent condition on the spread of that cluster, we deemed this acceptable as an almost correct localization would become correct during the path following steps. Once convergence was met the estimated robot location was given as the mean of all particle locations. A ghost particle was setup with that location as a means of tracking the robot in the simulation.

If the localisation filter did not converge on the first iteration, the robot was turned to the orthogonal angle (i.e to face parallel to a wall), and would move forward if it was safe to do so. Each particle underwent an equivalent turn and move sequence, except with added noise. After this, the robot would rescan and run the localisation loop again. This process was repeated until the convergence condition was met.

Once localisation had been achieved, an additional check was undertaken to more reliably estimate the orientation of the robot. For this, we scanned in 72 directions and our ghost particle also ran the same scan. The two scans were compared with the ghost scan being shifted by one 72 times and the one with the least difference was used to calculate the orientation of the robot.

```
for i = 1:numScans
            auxScan = circshift( ghostScan , -i ) ;
            score(i)=sum(abs(orientationScan(orientationScan >=9)-
                auxScan(orientationScan >=9)))
end
[~, minScanInd] = min(score);
bestAng = ((minScanInd -1)/numScans)*(2*pi);
botGhost.setBotAng(bestAng);
```

Figure 5: *The scan taken by our robot "ghost particle" is compared to the true robot scan, and the angular shift required to obtain the highest similarity between these scans is logged. The robots orientation estimate is then changed to the angle corresponding to this angular shift*

Another observation we made was that the robot localised well in the corners, in other regions it would either usually take longer or end up moving to a corner and then localising. So, if the robot found itself in one of these regions it would move to a corner, this essentially made the localisation a lot faster.

3 Navigation to Goal Position

3.1 Path Finding

In order to find a path from the estimated robot's position deduced during localisation to the goal position, the map is divided up into nodes on a grid. Each node is then tested to find those within the map, and any outside are discarded. From here, nodes are tested for distance from the walls of the map, and any which are deemed too close to the walls are also discarded. The desired distance from the wall can be simply specified according to the size of the robot and confidence in the motion of the robot to perform movements as commanded. The path finding then takes place on the remaining nodes, using the A* algorithm.

Once a path is found via the nodes, it undergoes simplification. This is performed recursively along the calculated path, starting with the first node. Individual direct paths are plotted from the current node to every later node on the path. If a direct path does not come within a certain distance of the walls, and is less than a maximum move distance (max. 30cm used in final code) this direct is deemed "viable". The furthest viable node is accepted as the next point on the shortened path, and all nodes which were previously in between are deleted. After this, the newly accepted next node is taken as the current node and the process is repeated. By reducing the number of nodes traversed, the time taken to complete the path following is greatly reduced, since at each node the robot stops and re-estimates its position.

3.2 Following the path

In order to mitigate the effects of motion noise as the robot travels along the planned path a number of steps have been taken. Since the robot is moving point-to-point, the process takes an iterative form.

- First, the robot turns toward the next node on the path.
- At the same time, particles are spawned on the robot's start position and programmed to move the same amount as the real robot with the addition of turning noise and distance noise, each respectively proportional to the magnitude of the turn and move distance.
- Upon completion of the step, the robot and all particles take scans. Particle scans are compared to that of the real robot, and the scan with the highest probability is chosen as our best estimate of the robot position and angle (unless the best botSim scan is too different to the real NXT scan, in which case path following is cancelled and relocalisation must happen).
- If the estimated position is too far from the path (but still considered a good estimate) then the path is re-planned and the process starts again. Otherwise, this process repeats until the final step in the path has been completed.