To be included – Simulation

Main flow of the code.

Move all + Sliding – to avoid the particles from stopping at map corners.

Correct shift – norm pdf

Scan – get the priority angles.

Multiple movements.

Results achieved!

**Cyclical shift**

In order to compare 2 signals we used the cyclical shift. In the beginning we used the Fourier transform to compute the difference because 2 signals in the Fourier domain are rotation and translation invariant. We, then decided to compute the shift in a regular way so that we could find the amount of shift and then rotate the particle back in the position of lowest error.

This method helps during exploration because even if the robot hits a wall or gets tangled, the particles will still correct their angle and try to go to the robot’s position again.

**Weight**

The value of the weighting is based on the mean of the likelihood of each measurement of the particle. The likelihood is modelled with a gaussian with a mean equal to the measurement got from the robot and standard deviation based on the sensor calibration.  For each measurement of the particle we compute the likelihood with respect to the robot’s measurement and then take the mean of them.

**Resample**

After weighting the particles we take the cumulative sum of the weights and we do the resampling. We compared two methods for resampling: the first is the standard random roulette selection and the second one is the fair roulette selection. This second method consist of evenly distributing the roulette draws so that there is no bias. This would guarantee that if there are 2 clusters of particles, the cluster with highest weight will get a fair share in the resample instead of counting on pure chance. Those are the results of our comparison on the real robot:

We decided to use the random sampler for the simulation because of the reliability of the readings and the other method for the real robot so that convergence would occur later.

To guarantee more precision, the resampled particles are jittered based on their weight. The higher the weight the more the jitter. This is done because a particle with high weight may be close to the correct position and, by jittering around it, we can increase the search space for more accuracy.

**Convergency**

At each iteration we check the standard deviation of the positions of the particle. If they are all close together it means that they are converging and so we stop. In the real robot we add another condition:

we cluster the particles using k-means

we pick the first 5% best particles and count how many of them cluster to each cluster-center

if most of them cluster near a specific centre we check the sum of the weights of the particles around that cluster centre and if it is higher than a threshold we consider it a convergeny.

If either this or the standard deviation condition are satisfied the robot is considered localised.

After this we delete the old particles and generate 100 particles around the mean position of the best particle closest to the selected cluster centre. We scan and correct the shift of those particles and measure the average angle and use that as the estimation.

**Exploration**

The simulation and the real robot use the same approach for exploration. The purpose is to move as forward and as far as possible while resampling. The robot chooses a random distance to travel and the checks 8 directions (45° each) with priority:

as soon as one of those directions allows for such movement the robot turns that way and travels. In the simulation, in case the robot cannot travel, it will tilt until it finds a feasible solution; the real robot will shorten the distance until it finds a suitable one.

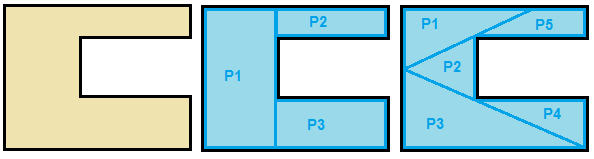
The real robot cannot 100% rely on the sensor readings so it has to make decisions based on a combination of the readings. For instance if the front reading tells that there is enough room but the reading in front-right contradicts it then the front and front-right direction are removed.

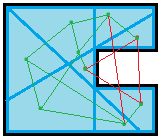
**Navigation**

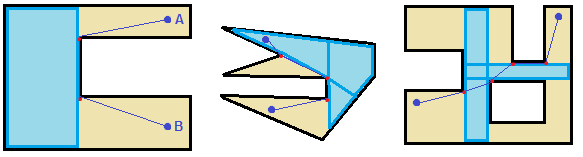
We compute the path using A\*. The input is the visibility graph we build from the map. In theory the visibility graph has to take in account all possible corners.  For our purpose we figured out that only one type of corner has to be used in the graph. There are 2 types of corners convex and concave:

We figured out that only the concave corners have to be used in the visibility graph. The reason for that is because the nature of convex and concave polygons.

Proof**:**

By definition, a convex polygon is a polygon in which every 2 points can be connected by a segment which lies entirely within the polygon. If the robot is inside a convex polygon the current position and the goal point will always be connected by a segment that lies entirely within the map. In case of a concave polygon this condition fails, however every concave polygon can be created by combining other convex polygons:

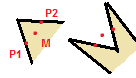
This means that 2 points in a concave polygon can be connected by a segment that lies entirely within the polygon only if those 2 points lie on the same convex sub-polygon. If those 2 points lie in different sub-polygons then the line that intersect those 2 points will intersect the boundaries of the polygon. 

The quickest path for getting to a common sub-polygon is moving along the corners that create the concave section which are the concave polygons (2 concave polygons connected by a line always lie on the same convex sub-polygon):

In figure 1 we show the basic step of moving to a concave corner that lies on the subpolygon; iin figure 2 we show the case in which the path is longer but it is just a repetition of the base case; figure 3 shows that even obstacles, if considered as space outside the map, work with the same approach.

In order to distinguish between convex and concave corners we developed this simple algorithm:

* *given a corner*
* *pick 2 points P1 and P2 each on the adjacent lines connected by the corner each at equal distance from the corner (at a very small distance ~2-5 pixels)*
* *find the median point M between P1 and P2*
* *if M lies within the map it is convex*
* *else it is concave*



An analogous algorithm is used for building the panned map.

Since in this project no map is going to have angles different from 90° we implemented a simplified algorithm:

* *given a corner*
* *pick 4 points at the north, south, east and west of the corner (at a very small distance ~2-5 pixels)*
* *if all points lie within the map it is concave*
* *else it is convex*

https://lh6.googleusercontent.com/MVUwCV5hUsu1oULOsbypAQe22l5-hU46rttQ3I_qzlD-683CdX07LpySjVJpTBUITPX3_ZgyO24o2HpTQidgdQ97tQVYnQXCrx2rIM_ZG8_kNZHizW8Wa9j7NDLXGA

After extracting all the concave corners we build the visibility graph and connect them together and connect the estimated position and the goal point and, if those 2 points are directly visible we move the robot towards the goal, otherwise we run A\*.

This section covers the problems encountered while working on the hardware side of the project, and provides an insight into the techniques employed to overcome these difficulties as well as the reasons behind them.

## Robot design

The design of the robot had gone through several phases of alterations, in response to the physical limitations posed in the experiments. Figure 1 and Figure 2 show the final design, the description of which will be covered in the following subsections.

|  |  |
| --- | --- |
| C:\Users\zy13643\Desktop\RS\illustrations\DSC_0964.jpg  14 cm  18.5 cm  Figure 1 Top view of the robot | C:\Users\zy13643\Desktop\RS\illustrations\DSC_0965.jpg  4.1 cm  14 cm  Figure 2 Side view of the robot |

### Caster wheel

A caster wheel was added for the purpose of keeping balance. The original version was bidirectional, i.e. it could only move forwards and backwards. The problem with this approach was that the caster wheel gets caught on the surface of the arena while turning. The workaround for this problem was to build an omnidirectional wheel so that it can move freely in all directions, which was proved to have rooted out the problem in test runs.

### Motors

A total of 3 motors were used, 2 of which are attached with wheels for movement. A problem encountered while moving near a wall was that one of the tyres would rub against the wall, and the robot would turn towards the wall as a result. The solution to this problem was to mount a wheel shell on each side to prevent the wheels from making direct contact with obstacles.

When the robot navigates around the arena, it relies on the use of the ultrasonic sensor to perceive its surroundings. To enable the robot to perform such task, one can either command it to turn its body and take measurements as appropriate, or place the sensor on a motor to allow it to do a 360° scan while staying put. The former has a disadvantage of accumulating turning error; therefore, the latter was adopted to keep the turning error at minimum. The sensor is located slightly off the centre of the robot (see Figure 1), so that the readings are approximately the distances between the centre of the robot and the walls.

### The Brick

The brick is placed upside down to keep the robot as high above the ground as possible, which would reduce the likelihood of being interfered by the bumps.

### Other

The wire keeps coming in the way of the sensor whilst it is turning was another problem that had to be dealt with, and the solution was to build a structure using beams to hold the wire down.

## **Motion functions**

A set of motion functions were created to enable the robot to roam around, each of which will be explained in detail in this section.

### Move

In order to enable the robot to move back and forth, a move function was created. This function makes both wheels turn in the same direction at a given speed for a given number of degrees. However, since the readings from the ultrasonic sensor are in centimetres, it would be more sensible to command the robot to travel a specified distance measured in the same unit. To do this, the relationship between the degrees at which the wheels turn and the desired travel distance has to be established. Since the distance travelled by each wheel per rotation is equivalent to the circumference of the wheel, the circumference equation was used to form part of Equation 1, where *D* denotes the desired travel distance.

Equation 1 Correlation between the degrees to turn and the travel distance

However, since the values used to calculate the distance were rounded off, the result is only an estimate; hence, calibration was done with regard to the conditions of the arena by trial and error. To prevent the robot from dashing into the walls, the ultrasonic sensor is used to actively detect the distance to the wall in front while moving, and a threshold distance was set at which the robot ceases going any further.

### Turn

As well as moving forwards and backwards in a straight line, the capability of turning is also essential for the robot to traverse through the arena. In general, a mobile robot can perform three types of turns.

The first method is to make the robot turn on the spot by turning the wheels in reverse directions for the same number of degrees, the centre of rotation is at the centre of the distance between the wheels as illustrated in Figure 3. However, since the wheels are not aligned with the centre of the robot, when the robot performs such a turn, the centre of the robot will orbit around the centre of rotation, as shown in Figure 4 and Figure 5.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Figure 3 Turn on the spot | Figure 4 Before rotation | Figure 5 After rotation |

Ideally, the centre of the robot should not be shifted. Thus, Equation 2 is used to work out the offset that the robot should move before and after turning, where *D* represents the distance between the wheels, and *rad* is the desired degrees to turn in radians.

Equation 2 Turn on the spot offset

The second method is to turn one wheel in either direction while keeping the other wheel still, the resulting position is shown in Figure 6. An obvious disadvantage of this type of turn is that the centre of rotation is at one of the wheels depending on in which direction the robot takes the turn, the circle drawn is subsequently two times bigger than that of turning on the spot; taking the size of the robot and the physical attributes of the arena into consideration, this approach is likely to cause collisions with the walls, and it was therefore not implemented.

|  |  |
| --- | --- |
| Figure 6 180° turn with one wheel being still | Figure 7 Manoeuvre around a corner |

The last turning tactic is manoeuvre, by which means that each wheel of the robot rotates for a different number of degrees, with the wheel on the opposite side of the turning direction moving further. This type of turn plays a major role in a situation that resembles Figure 7, where performing a turn on the spot is not only insufficient to bypass the walls, but may also result in a crash. The robot does a local scan to check the distances to its surrounding walls in 8 directions, and each direction is split into a number of readings on the basis of the number of scans. For instance, in the case of 18 scans, the north direction consists of readings 1, 2 and 18 (i.e. front readings, assuming that the robot is facing north with respect to the orientation of the arena). These readings determine whether if a direction is blocked, based on which the direction of manoeuvre is decided. For example, if the mean of the front readings is below a threshold value, and each reading is below another threshold value, then the front is asserted to be unnavigable.

### Correct angle

It was noted that the motor is not guaranteed to turn a specified number of degrees each time, which implies that there is a turning noise. To improve the precision, the actual degrees turned is subtracted from the expected degrees after each turn, the robot then takes another turn for the number of degrees equal to the offset.



Figure 8 Offset after turning

### Ultra scan

This function allows the robot to perform a 360° scan of the distances to its surrounding walls on the spot. The function had gone through several changes, in pursuit of efficiency. Originally, the idea was to rotate the sensor 360° in one direction and pause to take a measurement at every 45°. Although the accuracy is retained using this strategy, it is extremely time-consuming. To boost the efficiency, the strategy was switched to taking measurements while rotating, and the sensor rotates 360° clockwise and anticlockwise in turn to prevent the wire from getting tangled.

## Sensor calibration

It was noted that the ultrasonic sensor has its limits, in terms of its effective measure distance and the angles at which the measurements are accurate. Others have pointed out that the effective working angle for ultrasonic sensors is approximately 30° (Generation Robots, n.d.), as seen from Figure 9.

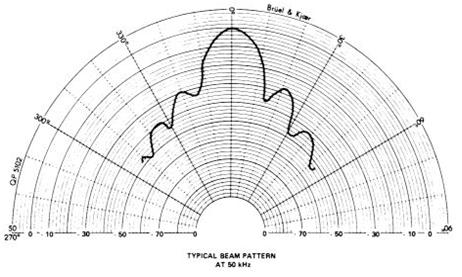


Figure 9 Typical shape for an ultrasonic beam (Generation Robots, n.d.)

To have an idea of how the accuracy of the sensor varies as the angle and distance change, the sensor was positioned at different angles and distances to a wall inside the arena, and measurements were taken. As a result, the following Gaussian distribution was generated.



Figure 10 Gaussian distribution of the sensor readings from different angles and at different distances

Particles are evenly spread around the map in order to not bias the initial estimation. All of them are, then, shifted towards the direction that mostly resembles the robot readings and they start moving according to the robots movement. After n movements they are shifted again and weight is assigned.

Conclusion

## References

1. Generation Robots. (n.d.). *Ultrasound sensor ? high quality ultrasound sensors available now.* Available: http://www.generationrobots.com/en/content/65-ultrasonic-sonar-sensors-for-robots. Last accessed 19th Mar 2014.