Intelligent Autonomous Robotics Report
Task 2

Babak Esmaeili, David Fullerton

October 21, 2015

Abstract

This paper describes the program we have written for a Khepera robot that allows the robot to search the lab environment while keeping track of its position and returns to the starting position. after a specified amount of time. We are recording the amount of wheels rotations using the wheel encoders to calculate how much the robot moved at each time step using the Odometry formulas. In this paper we also describe how we keep track of time so the robot knows when to return home. We have used simple geometry to find the angle that the robot needs to turn to point towards home. It turns towards the home direction then follows a direct path to it. If there are obstacles in the way, it follows a curvy path around the obstacle until the they are no longer in the way. It then tries to go back home again. In this paper we describes the difficulties we had when there were obstacles on the way back home, also the effect of the Odometry error in time. We tested the robot 10 times in arenas of different structures, both with and without obstacles. It managed to find home 8 times overall. The main limitation we faced at the end was the occasional circular movement near home but never fining home (infinite loop), which we describe in more details later.

1 Introduction

The task for the robot is to move around the environment lab (at a reasonable speed), while avoiding obstacles for 30 seconds, and then return to within 10cm of its starting position. The idea here was to implement a basic Odometry using the wheel encoders on the robot to keep track of its position and then uses this path to find its way back home. As our time step was short (0.1s), it was difficult for us to just print the position at each time step and read it. Thus, we decided to use the plot function to draw the robot's path over time. This made the observation a lot easier. Since there were no restriction for moving around (before returning home), we decided to use the same movement algorithm from the previous task, with the exception of adjusting the parameters of PID control for better movement. We approached the returning home task by using simple geometry to find the angle to the starting position, which we initialize to (0,0). We also set an error range for the angle. If the robot is pointing towards the (0,0) position in an angle within the error range, we allow the robot to go straight, unless there is an obstacle in the way.

2 Methods

2.1 Odometry

Initially we attempted Odometry by checking the wheel encoders to find how many times the wheels had rotated since the last time step and dividing this by the amount of time between time steps to find the speed of each wheel. This was then used to find the angle and the new x and y co-ordinates. The following are the functions we used during these calculations.

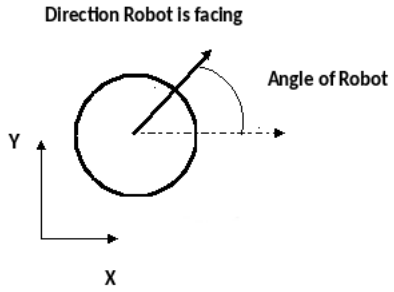


Figure 1:

v_{left}	Speed of left wheel
v_{right}	Speed of right wheel
θ	Old angle
Θ	New angle
D	Distance between the two wheels

$$\Theta \leftarrow \theta + \Delta\theta = \theta - 0.5 * (\frac{v_{left} - v_{right}}{D}) \tag{1}$$

$$x \leftarrow x + \Delta x = x + 0.5 * (v_{left} + v_{right}) * cos(\Theta) \tag{2}$$

$$y \leftarrow y + \Delta y = y + 0.5 * (v_{left} + v_{right}) * sin(\Theta) \tag{3}$$

We measured the distance between the wheels as about 52mm. After thorough testing of the Odometry we realised that something was not right with our calculations. First of all we simplified the calculations so that instead of dividing the wheel encoder counts by the time step to get it in m/s we left them as they were, essentially changing the units to metres per time step. This simplified the code and had no detrimental effects as the unit of speed doesnt matter since there isnt a measure of time anywhere else that affects the Odometry. This improved the results but not completely. It was at this point we noticed an inconsistency with our distance units. While the speed was calculated with units meters per time step, the measurement for the distance between the wheels had accidentally been entered as 0.52 instead of 0.052. Essentially the Odometry was calculating the angles as though the wheels of the robot were half a metre apart instead of only 52mm apart. This was why the Odometry was thrown off when turning. We decided to change all units to be millimetres instead of meters as this also resulted in much larger values for the x and y co-ordinates. Once these changes had been made the Odometry was far more successful, we graphed the robot moving around a rectangular arena and each corner was close to a right angle and the start and end points matched up.

2.2 Returning to original position

A function call 'home_direction' was created to deal with returning the robot back to its original position. After the robot had been running for 30 seconds the 'home_direction' function would be used to tell the robot what direction it should move next. Using odometry the robot sends the 'home_direction' function the x and y co-ordinates of the robot and the angle that the robot is facing. This angle is the angle from the x axis as shown in figure 1.

With these inputs the function returns the direction the robot should drive as a string. This output initially could be one of three options; 'Right Turn', 'Left Turn' or 'Straight'. The final version that was used in the demo had slightly different outputs than this which will be discussed

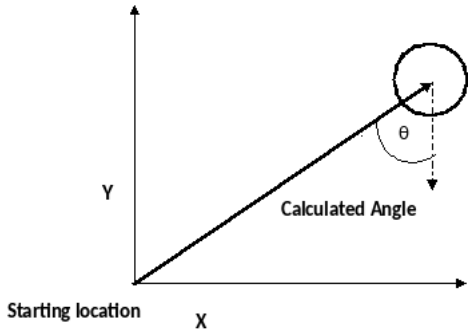


Figure 2:

later. The idea behind the 'home_direction' function was to have a simple but adaptive way of getting home. This function does not require keeping track of the path you have taken and is not affected by new objects encountered on the route home, as long as the robot has an accurate reading of its current position and angle it will attempt to head home. The way the function decides which instruction to give is fairly simple. Since the x and y co-ordinates are created by the robot when it sets off it is safe to assume that the starting location is the point (0,0). From this if we have the x and y co-ordinates of the robot at any time we can calculate the angle the robot is at with respect to the start position, this is shown in figure 2.

The angle is calculated using them matlab function atan2(Y,X), but this gives us the angle at the start not the one at the robot, so we get our angle as /2 atan2(Y,X). Once we have this angle we work out which way the robot should turn to be closer to facing towards home sending either a Right Turn or Left Turn signal which both cause the robot to turn on the spot, this behaviour is shown in figure 3. Once the robot is facing towards home, or within some predefined error of facing home, it sends the Straight instruction which causes the robot to drive straight. Once the robots x and y co-ordinates are close to 0, some error is given since the robot does move slightly between each checks, the robot stops and declares it has made it back to the start.
This method of moving back to the start works perfectly if no objects are encountered on the way home. However if an object is encountered the robot will find itself stuck. This is due to the fact that the only two turning instructions it receives are to turn on the spot. If the robot meets an object it will turn left on the spot. The resulting behaviour is that the robot meets an object, turns left to avoid it, registers no objects and so turns right to try and face home but instead finds the object again. This repeats with the robot making no progress. The current solution to this problem is a simple on that leads to its own problems. The solution is that, when trying to move towards the start, if the robot needs to turn right, instead of turning on the spot it turns in a curving motion. Now when the robot meets an object during the return journey it essentially wiggles its way around the object. This is illustrated in figure 4.
Although this method solves the problem of objects encountered while returning to start it creates a new problem where there can be occasions where the robot ends up with the start location on its right side but cannot turn sharp enough and so will simply circle around the start position unable to move closer as there is no instruction that will tell it to turn more sharply to the right.

The following is the logic involved in the home_direction function. First the world is split into four quadrants: (x¿0,y¿0),(x¿0,y¡=0),(x¡=0,y¿0) and (x¡=0,y¡=0). The logic changes slightly depending on the quadrant. In each quadrant a threshold angle T is calculated, if this ever is within 0.2 radians of 2 or 0 the Straight instruction is given as it will be facing home. If the

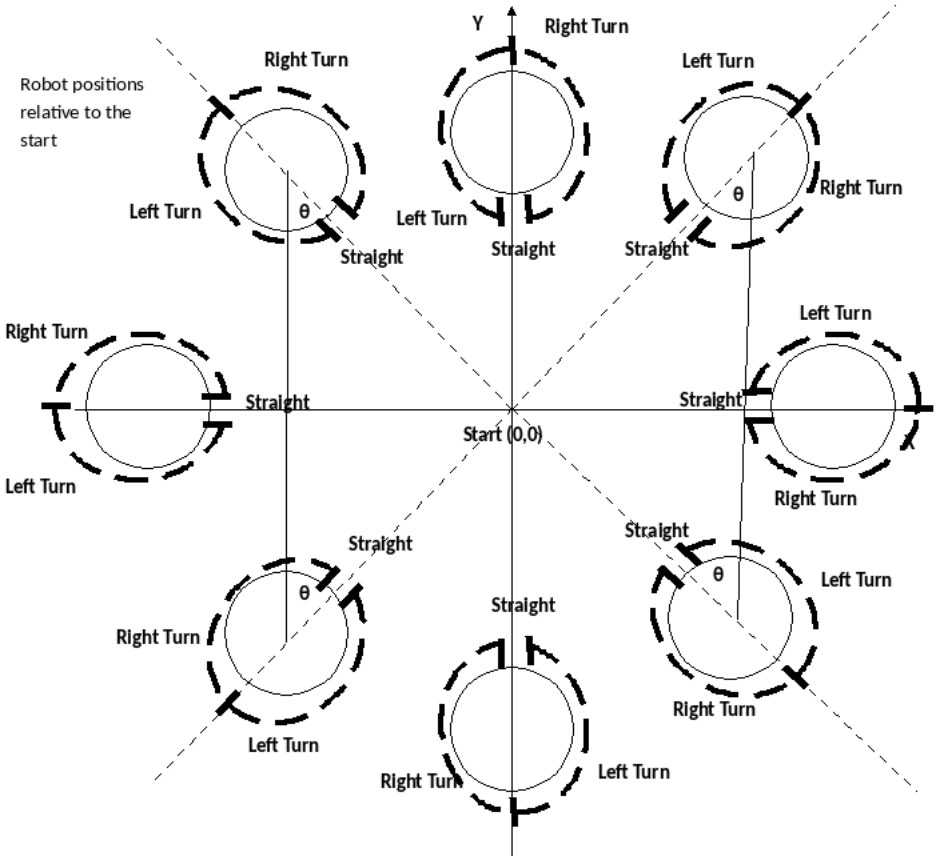

Robot positions relative to the start
Y
Right Turn
Left Turn
Straight
θ
Start (0,0)

Figure 3:

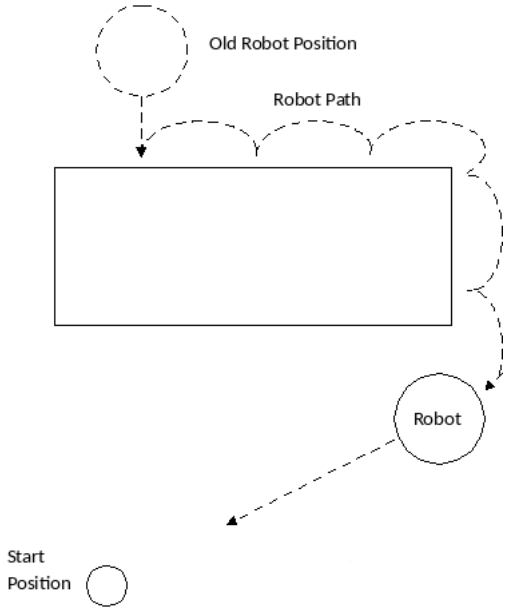
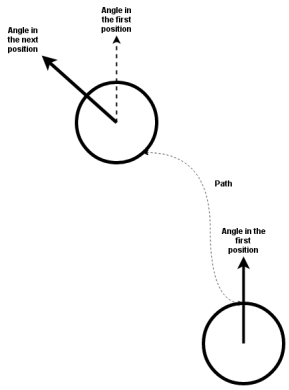


Figure 4:



robot comes within 5 units of (0,0) it declares it has made it home and stops.

2.3 Other staff

How we improved Odometry: Unit and angle

3 Results

The robot is now able to drive for 30 seconds, avoiding obstacles while following around the first surface it encounters using PID Controls. During these 30 seconds the robot keeps track of its angle and its x and y co-ordinates. It will also plot its x and y co-ordinates on a graph, showing the path it took. After 30 seconds have expired the robot will attempt to return home. This involves deciding whether it needs to turn left, right or move straight ahead at any given time. The final version of the robot dealt with objects blocking its direct path home by turning on the spot to avoid the obstacle, then turning in a curve to try and face towards home. This behaviour

results in the robot wiggling its way around the object but this means it can still find its way home even if new obstacles block its path. It will however favour going around the left hand side of an object, even if it would be a shorter path to turn right. When the robot returns to a location where its x and y co-ordinates are close to 0 it will stop and declare that it has reached home. During testing this was usually approximately 5 to 10 cm away from the actual start location due to accumulated error in the Odometry. However the solution to object avoidance, namely to curve right instead of turning on the spot, can lead to a situation where getting to the start location would require the robot to make a sharp right turn when it cannot. Instead the robot will circle around the home location never quite reaching home. Overall the robot is a success and can works its way around objects before returning home by the most direct route it can while avoiding possibly new obstacles.

4 Discussion

1. What helped us? The drawing
2. Limitation: Odometry error addition
3. Advantage: The arena does not need to be fixed, new obstacles can be in the path and we will avoid it. Since we do not follow the same redundant path that we came.
4. Improvement: Do PID controls instead of right turns

One main advantage our approach has is the indpendence from the arena and the history path. A different approach that many people have used is to follow the ecat same path that you come until it reaches home. But the followed path is most likely a redundant path and there is usually a more direct path to home. Also, there might be moving objects in the arena, resulting in the followed path being blocked. In our approach, it does not matter if obstaced have moved while the robot was searching. The robot finds the most direct way home regardsless. We also make a plot of the path the robot is following, when the program runs. This was extremely helpful for observation.

The main difficulty we faced in this task was returning home when there was an obstcale in the way. As described in the previous section, our original approach was to point towards the starting position with some error, and then go straight. But this appraoch had a problem. If it faces an obstacle on the way back, it turns left until the obstacle cannot be detected anymore by the front IR sensors. However, once it thinks the obstcale is not infront of it anymore, it will turn towards home again, therefore facing the obstcale. As a result, the robot wil be stuck in an infifnte loop. We fixed this problem by changing the 'turning on the spot' method to 'curvy path' method. But the ideal solution which we will work on in the future is for the robot to go around the obstace using PID control until the obstacle is no longer in the way.

5 Appendix

5.1 Main Controlller

5.2 Calculation of Wheel's Speeds

5.3 Odometry

5.4 'Reurning Home' Algorithm