Artificial Intelligence in Mobile Robots

Lab 2 - Position estimation

By *Tobias L & Özgun M*

Lab Assistant *Ali Abdul Khaliq*

Örebro University

September 20, 2016

Student information

Tobias Lindwall
870603-6657
tobiaslindwall@gmail.com

Özgun Marinov
920321-2379
ozgun.mirtchev@gmail.com

Report handed in: 2016-09-20

0.1 Overview

In this lab a position tracking function was going to be implemented into the ePuck. By using a given algorithm and a collection of formulas the relative position estimation could be calculated. To know where the ePuck was located in relation to the built-in steps, the position of the ePuck was estimated depending on the odometric readings and the initial position. The orientation angle was also a part of this global position. The position tracking function was going to be updated in a continuous read-update-pause loop. After implementing the position estimation functionality, evaluation of the estimate was done to decide the quality of the function.

0.2 Tasks

Task 1 - Implement position estimation on the ePuck

To implement position estimation into the ePuck, a generic algorithm was given by the professor from the lecture. Written in pseudo-code there was a function named update_position(), which described the algortihm and the steps to implement. This was done by translating the pseudo-code into C-code as much as possible and making some adjustments where needed. The only information the function receives from the ePuck is the amount of steps each wheel has moved from the start. The algorithm is meant to calculate the current position of the ePuck by approximation. The function was named update_position(), for the readability. update_position() first gets the amount of steps each wheel has moved since the last time the current position was updated.

From this, the travelled distance of each wheel (d_L,d_R) was aquired to give the total distance the ePuck has moved: $dist = \frac{d_L + d_R}{2}$

The orientation difference (ang) of the ePuck was also aquired by using d_L,d_R and the diameter of the robot (D): $ang = \frac{d_R - d_L}{D}$

To get the final positions for the ePuck, this equation was used:

$$\begin{cases} x = x_0 + d_x \cos(th_0) - d_y \sin(th_0) \\ y = y_0 + d_x \sin(th_0) + d_y \cos(th_0) \end{cases}$$

Where (x, y) is the position of the ePuck,(x_0, y_0) is the previous position and th_0 is the previous angle of the ePuck.

To get the current global orientation angle of the ePuck:
$th = angle + th_0$

To calculate d_x, d_y the following equation was used:

$$\begin{cases} d_x = dist * \cos\left(\frac{ang}{2}\right) \\ d_y = dist * \sin\left(\frac{ang}{2}\right) \end{cases}$$

The entire code can be viewed in appendix: Code1.

Task 2 - Verify that the estimates are correct over simple trajectories.

A trajectory resembling a rectangle was tested to see if the ePuck would return to it's original position from where it started and if the values would be correct. If the start values was (0,0) it would also be (0,0) or close to after the first lap of the rectangle.

Another trajectory resembling a circle was also tested to verify that the angle of the robot was correct. When it was going around one lap in the circle the angle would be 2π and half circle it would be close to π. This means that the ePuck has gone half a circle from the start position. This was controlled with some help from reference marks on the surface.

Since we know 1 mm is 7.71 steps we moved the ePuck 200 steps and measured the ePuck to have moved $\frac{200}{7.71}$ which is around 26 mm.

Task 3 (Optional) - Measure the error over a closed trajectory

To measure errors the rectangle trajectory was used from the previous task. The ePuck travelled 1000 steps on each side, which translates to around 129.7 mm and turned 90 degrees on each point. Collection of position data from the ePuck was done over several laps on the same trajectory to get a more precise result of the error. The collected data was put into Excel where some simple calculations were done to see how much difference there was between the laps.

-90 degrees								
Reference-point	1		2		3		4	
Position	x	y	x	y	x	y	x	y
Lap 1	0.00	0.00	208.30	0.00	207.26	-208.90	-1.77	-206.86
Lap 2	1.25	0.85	209.21	-3.16	207.18	-212.00	-4.38	-205.99
Lap 3	2.62	1.89	211.37	6.13	202.36	-214.71	-5.53	-204.73
Average-Error:	**1.81**	**0.99**	**1.54**	**-3.07**	**-2.45**	**-2.90**	**-1.88**	**1.07**

90 degrees								
Reference-point	1		2		3		4	
Position	x	y	x	y	x	y	x	y
Lap 1	0.00	0.00	208.69	0.00	207.65	208.51	-0.34	206.48
Lap 2	2.7	-2.4	210.66	1.61	205.64	209.81	-3.17	203.78
Average-Error:	2.7	**-2.4**	**1.97**	**1.61**	**-2.01**	**1.30**	**-2.83**	**-2.7**

0.3 Conclusions

In task 1 we encountered the problem with division by zero when using the equation (given in lecture slides) for calculating d_x, d_y. $\begin{cases} d_x = dist\frac{\sin(ang)}{ang} \\ d_y = dist\frac{1-\cos(ang)}{ang} \end{cases}$

We changed the equation to a different one (also from lecture slides) which didn't divide with ang (which could be zero). We used this other equation which gives an approximative calculation of the variables. The first one is said to be computationally better but we cannot decide if one or the other is better. The equations have similar behaviour and produce close values if they get the same input *ang* (values in the interval [-π,π]).

We also normalized the angle to start from 0 again when going around a full circle to prevent buildup. We also normalized when the angle went smaller than 0 through adding *ang* with 2π.

The errors in the measurement in Task 3 was smaller than we expected. We knew that the smallest factor would make a great difference in where the ePuck would end up after moving continuously and turning a couple of times, but it was suprisingly precise. Greater precision could be maintained through using i.e. IR-sensors that measures the distance to a certain wall or another reference object.

Appendix

```
void update_position(float D)
{
    newSteps = GetSteps();
    diffSteps.l = newSteps.l - prevSteps.l;
    diffSteps.r = newSteps.r - prevSteps.r;
    diffmmL = (diffSteps.l * 1.0) / StepsPermm;
    diffmmR = (diffSteps.r * 1.0) / StepsPermm;

    dist = (diffmmL + diffmmR) / 2;
    ang = (diffmmR - diffmmL) / D;

    //dx = dist * sin(ang) / ang;
    //dy = dist * (1-cos(ang)) / ang;
    // Using functions below instead because division by
       0 is not possible.

    dx = dist * cos(ang/2);
    dy = dist * sin(ang/2);

    prevSteps = newSteps;
    RobPos.x += dx * cos(thPrev) - dy * sin(thPrev);
    RobPos.y += dx * sin(thPrev) + dy * cos(thPrev);

    th = ang + thPrev;
    if (th >= 2 * PI)
        th -= 2 * PI;
    else if (th < 0)
        th += 2 * PI;
    RobPos.th = th;
    thPrev = th;

    printf("dist: %lf mm\nang : %lf rad\ndx/p:
       %lf\ndy/p: %lf\n\n", dist, ang,
      dx * cos(thPrev) - dy * sin(thPrev),
      dx * sin(thPrev) + dy * cos(thPrev));
    printf("Robot position: (%lf, %lf)\nRobot Angle:
       %lf\n\n", RobPos.x, RobPos.y, RobPos.th);
    Sleep(10);
}
```