Autonomous Robotics Lab 2 Report

Master in Computer Vision

UNIVERSITE DE BOURGOGNE

Centre Universitaire Condorcet - UB, Le Creusot

01 May 2017

Submitted to: Prof. Xevi Cufí

Submitted by: Mohit Kumar Ahuja

Marc Blanchon

Aim:

- Convert the measures of the encoders to distances travelled by the left and right wheels. Print the results using a real e-puck.
- Compute now the odometry. You will have to initialize the position of the robot to r0= (0, 0, 0) at the beginning of the simulation and update it according to the change in the encoder values.
- Find the error of the odometry in a translation only movement. Write a code that moves the robot straight until the travelled distance is 10cm, then stop the robot. Repeat the running 10 times measuring the real distance each time. Plot a histogram with the measures taken. Calculate the mean and standard deviation.
- Find the error of the odometry in a rotation only movement. Write a code that rotates the robot until the rotated angle is 90^o, then stop the robot. Repeat the running 10 times measuring the real angle each time. Plot an histogram with the measures taken. Calculate the mean and standard deviation. Print the protractor given in next page to measure the angles.
- Find the error of the odometry in a combined movement. Write a code that moves straight 10 cm, then rotate 90^o, then moves straight 10 cm, then rotate 90^o, until you perform a square trajectory. Repeat the running 10 times measuring the final position each time. Plot a histogram with the measures taken. Calculate the mean and standard deviation.
- What are the causes of the errors? How could the odometry be improved?

Function Definition: We made a function named “update_odometry” to compute the distance travelled by the robot and to update the real coordinates of the robot as well as the angle of rotation.

Solutions:

- Convert the measures of the encoders to distances travelled by the left and right wheels. Print the results using a real e-puck.

  Solution: As we know that the encoder counts only steps and the real measurement was calculated in the code as:
  double l = wb_differential_wheels_get_left_encoder(); // Steps for left wheel
  double r = wb_differential_wheels_get_right_encoder();// Steps for right wheel
  *double dl = l / ENCODER_RESOLUTION * WHEEL_RADIUS; // distance covered by left wheel in meter*

*double dr = r / ENCODER_RESOLUTION * WHEEL_RADIUS; // distance covered by right wheel in meter*

Wheel radius was defined in starting as 0.0205 (i.e the radius of wheel)

- Compute now the odometry. You will have to initialize the position of the robot to r0= (0, 0, 0) at the beginning of the simulation and update it according to the change in the encoder values.

  Solution: Odometry is the use of data from the movement of actuators to estimate change in position over time. In the case of wheeled robots, from the movement of each wheel. The change in position will depend on the rotation of the wheels, on its radius and the width between them. In the starting of every code, the initial positions are being set to zero. You can find this in code:

  double d[3] = {0.0,0.0,0.0};

  And for odometry we made a function update_odometry in which as the robot moves the {x,y} coordinated of robot gets updated automatically.

  ```
  double da = (((dr - dl))/ AXLE_LENGTH);    // delta orientation // axle_length = width
  double dis = ((dl+dr)/2);                   // mean distance of both tires
  disp += ((dl+dr)/2);           // mean distance of both tires stored in another variable
  d[0] += dis*cos(d[2]);                      // calculate and update x coordinate
  d[1] += dis*sin(d[2])                       // calculate and update y coordinate
  d[2] += da;                                 // calculate and update angle
  ```

  So, in this function, we calculate the delta orientation by dividing the distance of both wheels by width of wheel. And the mean distance “dis” and the {x,y} coordinates are being updated by multiplying dis with cos ø and by sin ø. And the {x,y} coordinates are being updated in every iteration.

About The First Code (Project_2.1):

It’s very important first to know about the code before doing any experiment to the code. So, here are some points to know about the code in details:

- Open webot software and choose “your Project” among four options. Now click on open world icon and in the appeared Windows open folder Project_2.1 (attached in mail).

- In Project_2.1 folder, open worlds and select “e-puck” file. A world will appear on the screen with editor and console.
- In this code, after initializing webot and the encoders we set the encoder values to zero and we put one “if condition” that if the distance travelled by robot is 10 cm, the robot should stop and the infinite for loop should terminate. As you can see clearly in the code:

```
if ((0.1<disp) && (disp<0.115)) {              //move robot till it covers 10 cm

wb_differential_wheels_set_speed(0,0);        // stop
update_odometry();                            //update the coordinates of {x,y} and Angle
wb_robot_step(TIME_STEP*10000);
printf("stop \n");
break;                                        // terminate the loop
}
```

We have given a range from 10 cm till 11.5 cm for stopping the robot because of robot speed sometimes it overshoot 10 cm so giving a range it stops at around 10.05 cm. After 10 iterations in real world as well as in simulator we calculated errors. Here is the bar graph of error values.

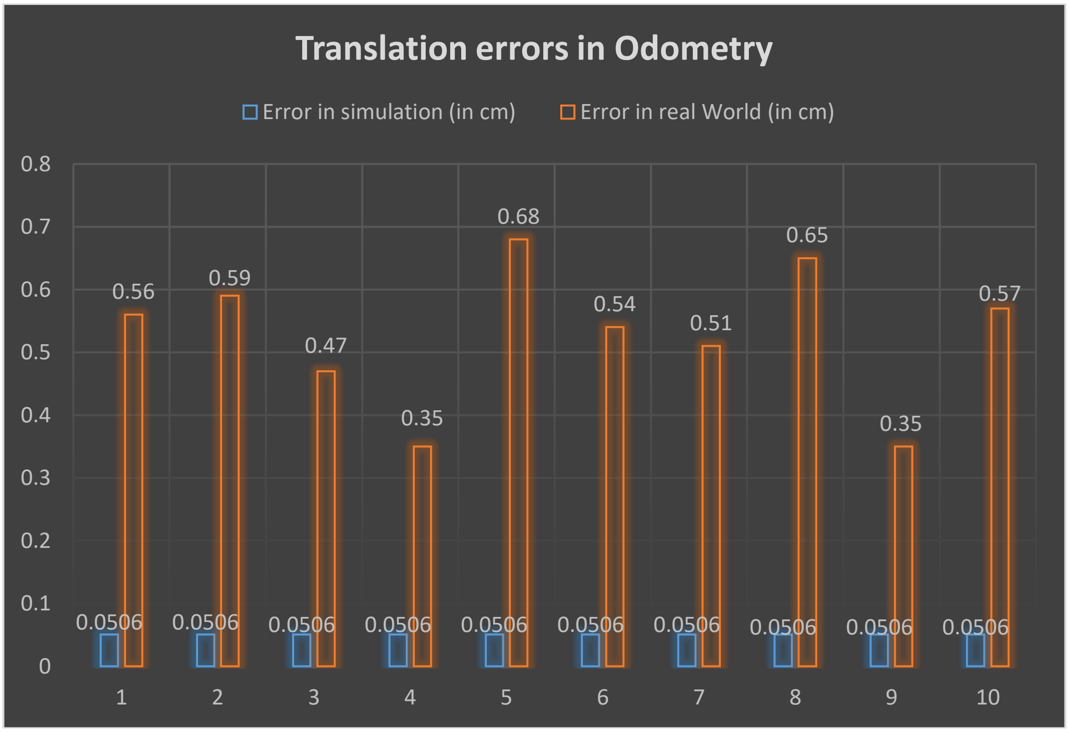

Figure (1): Translation errors in simulation and in real world

{The code is fully commented for better understanding of the reader}

About The Second Code (Project_2.2):

- Open webot software and choose "your Project" among four options. Now click on open world icon and in the appeared Windows open folder Project_2.2 (attached in mail).
- In Project_2.2 folder, open worlds and select "e-puck" file. A world will appear on the screen with editor and console.
- In this code, after initializing webot and the encoders we set the encoder values to zero and we put one "if condition" that if the distance travelled by robot is 10 cm and inside that :if condition", we put a nested if that as soon as the robot completes 10 cm, it will take a turn till the angle is 90 degree and as soon as the angle reaches 90 degree it should terminate the loop. As you can see clearly in the code:

```
if ((0.1<disp) && (disp<0.115)) {                // 1st 90 deg turn when robot covers 10 cm
    wb_differential_wheels_set_speed(-50,50); // turn left
    wb_robot_step(TIME_STEP);
    update_odometry();                          //update the coordinates of {x,y} and Angle
    printf("turn left(90 Deg) \n");

    if (1.54<d[2] && d[2]<1.59){                       // Range given for robot to stop when
                                                       the 90 deg turn is done

      wb_differential_wheels_set_speed(00,00); // stop
      update_odometry();                        //update the coordinates of {x,y} and Angle
      wb_robot_step(TIME_STEP*1000);
      printf("stop\n");
      break;                                    // to terminate the loop

    }
  }
```

We have given a range from 10 cm till 11.5 cm for stopping the robot because of robot speed sometimes it overshoot 10 cm so giving a range it stops at around 10.05 cm. We have also given a range from 1.54 rad (88.5 degree) till 1.59 rad (91.1 degree) to stop the rotation of robot because of robot speed sometimes it overshoot 90 degree so giving a range it stops at around 90.5 degree.

After 10 iterations in real world as well as in simulator we calculated errors. Here is the bar graph of error values.

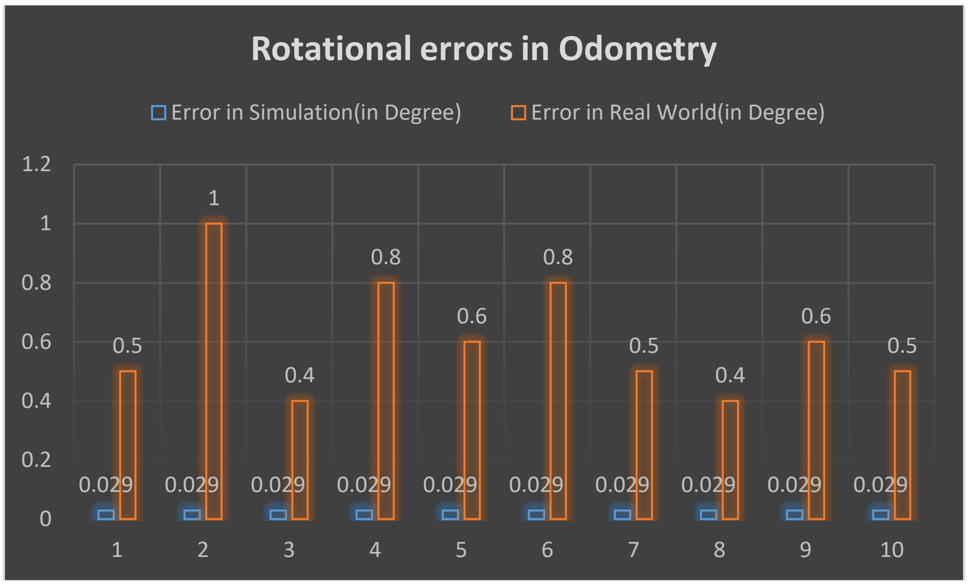

Figure (2): Rotational errors in simulation and in real world

{The code is fully commented for better understanding of the reader}

About The Third Code (Project_2.3):

- Open webot software and choose "your Project" among four options. Now click on open world icon and in the appeared Windows open folder Project_2.3 (attached in mail).
- In Project_2.3 folder, open worlds and select "e-puck" file. A world will appear on the screen with editor and console.
- In this code, after initializing webot and the encoders we set the encoder values to zero and we put four "if conditions".
- In the first "if condition", if the distance travelled by robot is 10 cm and inside that if condition", we put a nested if that as soon as the robot completes 10 cm, it will take a turn till the angle is 90 degree and as soon as the angle reaches 90 degree it satisfies the condition and move forward. As you can see clearly in the code:

```
if ((0.1<disp) && (disp<0.115)) {              // 1st 90 deg turn when robot covers 10 cm

    wb_differential_wheels_set_speed(-50,50); // turn left
    wb_robot_step(TIME_STEP);
    update_odometry();                          //update the coordinates of {x,y} and Angle
    printf("turn left(90 Deg) \n");
    if (1.50<d[2] && d[2]<1.59){                // Range given for robot to move forward
                                                when the 90 deg turn is done

      wb_differential_wheels_set_speed(300,300); // go straight
      wb_robot_step(TIME_STEP);
      printf("run 1\n");
      update_odometry(); //update the coordinates of {x,y} and Angle

    }
  }
```

The range explanation is given above in project 2.2, so I will just mention the range given in the code:
Turn Right if robot covers: 10cm to 11.5 cm
Turn until the angle is in between: 88.5 to 91.1 degree (1.54 – 1.59 rad)

- The same is being repeated for rest of the "if conditions", so I will mention the range's for turning.

  For 2nd if condition:
  Turn Right if robot covers: 19.8 cm to 21.3 cm
  Turn until the angle is in between: 180 to 181.6 degree (3.14 – 3.17 rad)

  For 3rd if condition:
  Turn Right if robot covers: 29.8 cm to 31.3 cm
  Turn until the angle is in between: 270 to 271.5 degree (4.70 – 4.74 rad)

  For 4th if condition:
  Stop if robot covers: 39.9 cm to 40.5 cm

  By doing this the robot completes a square. And after running the code, you will analyze that the error between the starting coordinates and the stopping coordinates are less than 0.6 cm.

  After 10 iterations in real world as well as in simulator we calculated errors. Here is the bar graph of error values.

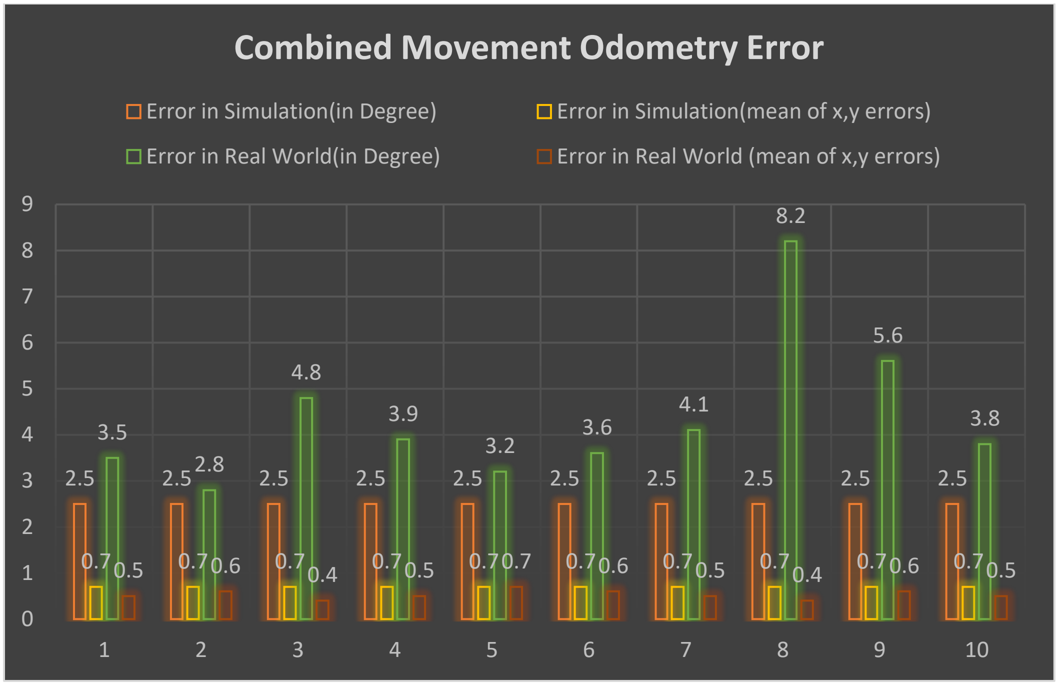

Figure (2): combined movement errors in simulation and in real world

{The code is fully commented for better understanding of the reader}

- What are the causes of the errors? How could the odometry be improved?

  Solution:

  1. The main cause of errors are the speed of robot, if we want to perform some action using e-puck, we have to give a range for turning or rotating or stopping due to which errors occur. To improve this, we have to reduce the speed of robot to minimum.

  2. The other major cause of errors in odomery is that it updates the new coordinates with respect to previous coordinates but if we get error somewhere, that error will be accumulated till the end which leads to major errors.

     This can be improved by calculating the error simultaneously and updating the coordinates in odometry by subtracting the errors simultaneously.

Youtube Link's: For better understanding of wall follower program, how it runs in simulation and in real world, I made a video and you can find the links below:

1. Simulation Link: https://youtu.be/DgnqMCP0Qzw
2. Real World Video Link: https://youtu.be/qeaX1SMvJRQ