Comp2403 Motor drive and Odometry

C.B.Price September 2020

Purpose	To learn how to drive the robot on a straight line then on a circular arc. To investigate the accuracy of movement measured by the wheel encoders.
Files Required	Webots project folders on website. Use the world **CBP_ePuck_Odometry_1.wbt** with the associated controller c-file for the straight line motion and **CBP_ePuck_Odometry_3.wbt** with the associated controller c-file for motion on a circular arc.
ILO Contribution	2
Send to Me	If you are working online, please send movie-clip of your solution.
Homework	Re-read CBP_Notes Chapter 3 and Chapter 2.5 and make sure you understand the maths. If not, ask me to review this in the next session.

Activities

Maths symbols and code equivalents

The maths symbols are the same as those used in CBP_Notes, and the code variable names are those used in the templates

r	wheel radius	**WHEEL_RADIUS**
a	half angle length	**AXLE_LENGTH/2.0**
ω_L	left wheel angular velocity	**omegaL**
ω_R	right wheel angular velocity	**omegaR**
Ω_{Rob}	angular velocity of robot about its centre	**robotOmega**
π	pi	**PI**

---

1 Moving down a straight line

(a) Open up the world **CBP_ePuck_Odometry_1.wbt** and make sure the robot is pointing to the right (along the z-axis). The desired distance has been set to 0.5m and the time to 10 seconds.

Check out which devices are enabled in the init() function.

(b) Add the following code to calculate the motor angular velocities **omegaL** and **omegaR** before the while(…) loop**.** These are explained in my Notes, equation 3.18.

```
speed = desiredDistance/desiredTime;
omegaL = speed/WHEEL_RADIUS;
omegaR = omegaL;
```

Compile and run.

(c) In the state **STATE_MOVE** add the following lines to read the left wheel encoder, after the gps values are printed (code already in place)

```
double wheelTheta = wb_position_sensor_get_value(leftWheelAngle);
```

Now add this line to calculate the *perceived* distance, the distance the robot thinks it has travelled, then print this out, together with the time, equation 3.1

```
double dist = wheelTheta*WHEEL_RADIUS;

printf("Perceived distance moved = %f , Time = %f\n",dist,time);
```

Compile and run.

(d) Now we have to add code to turn off the motors when the robot thinks it has reached the required distance in the desired time. We will test when the robot has moved for the required time. So write code based on the following pseudo-code. Think carefully about how to code the condition. You should make use of the perceived distance.

```
if(some condition)
  set angular speeds to zero
  go to STATE_DONE
```

(e) Now compile, debug and run your code. The robot should halt after 10 seconds.

(i) Note down the output 'distance' which is calculated from the encoder angle. This is the distance the robot thinks it has moved and compare this with the GPS value. This is the *actual* distance the robot has moved. These are not the same. Can you explain why one is larger than the other?

(ii) Calculate the time-step, the difference between adjacent time values. This should equal what is given **in #define TIME_STEP**

(f) (i) Open up a File Explorer and navigate to the sub-folder CBP_ePuck_Odometry_1 controller.
(ii) Copy the path. Open up Octave and cd to this folder.
(iii) Type **Robot** which will draw a graph of gps distance (horizontal axis) versus perceived distance (vertical axis). This should be a straight line, but the values are not identical.
(iv) Type **axis([0,0.1,0,0.1]);** to magnify the start of the graph. Here you should see that the wheels are slipping when the motor is initially driven.

---

2 Moving on a Circular Arc (refer to CBP_Notes, chapter 3 'Motion on a circular arc')

(a) Open up the world **CBP_ePuck_Odometry_3.wbt** and make sure the robot is pointing to the right (along the z-axis). We have to specify three parameters for movement along an arc. These are set as follows

```
double desiredRadius = 0.5;
double desiredDegrees = 90;
double desiredTime = 10.0;
```

Also note that the gps device is enabled as is the compass **comp** device.

Add the following lines to enable the wheel encoders, returning the angles of each wheel θ_L and θ_R

```
leftWheelAngle = wb_robot_get_device("left wheel sensor");
wb_position_sensor_enable(leftWheelAngle,TIME_STEP);
rightWheelAngle = wb_robot_get_device("right wheel sensor");
wb_position_sensor_enable(rightWheelAngle,TIME_STEP);
```

(b) Now we must use our desired parameters to set the actual wheel angular velocities **omegaL** and **omegaR.** This is done in **STATE_CALC_OMEGAs.** First we calculate **robotOmega** (the rotational angular velocity about its centre. This is just the angle turned divided by the time *T*

$$\Omega_{Rob} = \frac{desired\ angle}{desired\ time}\left(\frac{\pi}{180}\right)$$

Note the conversion from desired angle in degrees to radians, required by the maths. Now add code to calculate Ω_{Rob}. Look at the table of symbols at the top of the worksheet.

(c) Now we have to generate **omegaL** and **omegaR** using Ω_{Rob} and the desired radius *R.* The maths is explained in CBP_Notes Chapter 3, but here's the equations you need equations 3.11, 3.12

$$\omega_L = \frac{(R-a)\Omega_{Rob}}{r}$$
$$\omega_R = \frac{(R+a)\Omega_{Rob}}{r}$$

So, looking at the symbols at the top of the worksheet, code up these expressions in the state **STATE_CALC_OMEGAS** where … the omegas are calculated.

(d) Now we have to calculate the desired distance the left and right wheels should move.

(i) Here's the code for the left wheel, equation 3.2

```
distL_desired = (desiredRadius - AXLE_LENGTH/2.0)*desiredDegrees*(PI/180);
```

(ii) Repeat for the right wheel

(iii) Now calculate the desired distance moved by the robot centre, equation 3.6

```
distC_desired = (distL_desired + distR_desired)/2;
```

(e) Now in **STATE_ROTATE** we are going to get the perceived distance moved by both left and right wheels and also of the centre of the robot. Also we are going to get the actual GPS location of the robot. These sets of coordinates will be written into the **logFile** which we can call in Octave.

(i) get the perceived distance travelled by the left wheel:

```
wheelThetaL = wb_position_sensor_get_value(leftWheelAngle);
distL_perceived = wheelThetaL*WHEEL_RADIUS;
```

(ii) repeat for the right wheel

(iii) calculate the perceived distance travelled by the centre of the robot

```
distC_perceived = (distL_perceived + distR_perceived)/2;
```

(f) Finally, in **STATE_ROTATE** we must add code to set the wheel omegas to zero when the turn is complete. You should do this by comparing the perceived distance moved by the centre of the robot with the desired distance moved by the centre. Something like this

if(some condition)
stop the motors
transit to STATE_REPORT.

---

3 Investigating the Circular Arc

(a) Let's use the values given for a 90 degree turn on a radius of 0.5m. Before we start the work, THINK Which wheel is moving faster to make a left-hand (counter-clockwise) arc?

Run the simulation and capture the data in the console. Also use Octave to plot out the perceived (red) and the desired (blue) trajectory.

Here are some questions

(i) Which wheel is likely to slip more? Does the trajectory agree with this?
(ii) Do the differences between perceived and desired wheel distances make sense?
(iii) Look at the perceived coordinates. These are close to (0.5, 0.5) as expected. So the robot thinks it has arrived at the right spot. But it's fooled itself. Calculate the percentage errors in x and y coordinates:

% error = (val2 – val1) * 100 / val1

(iv) Look at how bad the final angle is. Calc the percentage error.

(v) Do I understand what I am seeing

(b) Repeat for another final angle. Or change the radius. But only change one thing.