Robotics

Practical 2: Sensors and Feedback Control

Andrew Davison
a.davison@imperial.ac.uk

1 Introduction

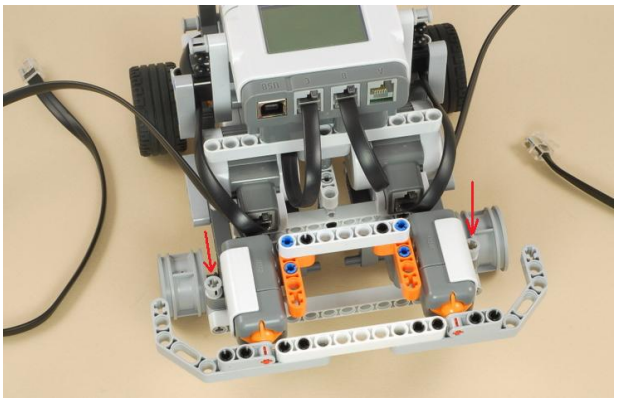
In this practical we will use some of the outward-looking sensors which come with the Lego Mindstorms NXT kits, in particular the simple touch sensor and the more sophisticated sonar (ultrasonic) sensor, and implement servoing behaviours.

You will need to somewhat modify the physical design of your robot to fit the sensors, and making your robot move accurately still may require slight adjustment of the parameters you calibrated in the previous practical, but hopefully not much. Also issue that might require some modification of tuning and calibration is that from now on we will be running the robots on the lab carpet rather than paper.

This week's practical is UNASSESSED. There are several tasks laid out in the following sections, and it will be very useful for your understanding of the examinable material in the course to attempt them, but we will not have a formal assessment session and achievement in this practical will not count towards your overall coursework mark for Robotics.

2 Objectives

2.1 Simple On/Off Forward/Backward Control with Touch Sensors



The touch sensor is a simple binary switch whose state is either 0 or 1 depending on whether it is pressed. Mount two touch sensors onto the front of your robot with a simple bumper mechanism which might be similar to the one in the picture above. Adding the long blocks to the front of the touch sensors creates a bumper which should effectively be able to detect when the robot collides with an obstacle when driving forwards, causing either or both of the touch sensors to be activated. (Note that you will need four cables to connect both touch sensors as well as two motors, and you can come and get another one from us.)

1. Write a simple program which will drive your robot forward until it hits an obstacle and the bumper detects a crash. Your robot should react in a sensible way: if the bump is on its left side, it should reverse a short distance and turn to the right. If the bump is on the right, it should reverse and turn to the left. If the bump is straight ahead and triggers both touch sensors the turn can be either left or right.

2. In either case, after this short avoidance manoeuvre the robot should continue to drive forward until it hits another obstacle.

Have a look at the Dexter example program `LEGO-Touch_Sensor.py` in the `Dexter/BrickPi3/Software/Python/Examples` directory to see how to interface with the touch sensors. You will see that each touch sensor returns 0 when not pressed and 1 when pressed.

Your robot should be driving forward using velocity control, with a loop continually monitoring the value of the touch sensors. If a bump is detected your avoidance manouevre can be executed using position control (move back and turn a certain amount) before returning to the original velocity control loop as the robot resumes moving forward.

2.2 Forward/Backward Proportional Servoing with the Sonar Sensor

The Lego NXT sonar sensor is able to measure the distance in centimetres to an object or obstacle placed in front of it by timing the interval taken by an ultrasonic pulse to bounce off the object and return.

Mount the sonar sensor on the front of your robot facing straight forwards and **parallel to the ground**. Write a program which will control the forward/backward motion of the robot with the aim of keeping it a fixed distance of 30cm from an obstacle in front.

The program should continuously monitor the depth output of the sonar sensor in a loop. Look at the Dexter example program `NXT-Ultrasonic_Sensor.py` to see how to read the sonar measurement. It returns a value in centimetres. Use velocity control and set the velocity demands of both wheels to be proportional to the difference or 'error' between the desired distance and the currently measured distance using **proportional control** with a single proportional gain value. The speed of the wheels, both equal, should be set to be proportional to the negative of this error. The result should be that when the robot is far from the desired distance of 30cm from the wall, it should move rapidly towards that point; and as it gets closer it will gradually slow down until stopping smoothly in the right place. Adjust your gain until you get smooth performance which is neither too sluggish nor jerky.

Note that the sonar sensor will occasionally report a reading which is substantially wrong, usually when it isn't able to get a good strong sonar reflection from the surface it is looking at, and your program

will probably need to ignore these sudden different readings (e.g. by using the median of the last few readings in the control law) in order to get smooth operation.

When debugging this behaviour it will be useful to display on the PC over WiFi some text which shows in real-time what is happening in the servoing process; e.g. the measured and desired distance, their difference and the calculated motor control speed. You can use Python `print` commands for this.

When placed in front of a wall your robot should move to the correct distance and stop smoothly. It should also track backwards and forwards when presented with a hand-held moving object such as a book.

Such forward/backward servoing with a depth sensor would be the basis for instance for a system to keep an autonomous car at a safe distance from the car in front on a motorway (although it is interesting to consider how the detail of the servoing algorithm should be different in that case where the robot car should match the non-zero velocity of the car in front rather than stop when at the required distance).

2.3 Wall Following

If you change the mounting of the sonar sensor by 90° so that it points either left or right then with a small change to your program your robot can use proportional servo control to follow along a wall at a fixed distance of 30cm. Assuming for now that the sensor points right and that we want to drive along a wall at distance 30cm, then when the sensor measures exactly 30cm we want to stay at that distance and therefore should set both motors to the same speed to drive straight forwards. If the measured distance is more than 30cm, we want to turn towards the right, and therefore the left wheel needs to speed up while the right wheel slows down. For proportional control, we should set the difference between the speeds of the wheels to be proportional to the negative 'error' between the measured distance z and the desired distance of $d = 30$cm. The average speed of the two wheels should stay at a pre-chosen constant value (so we are always increasing the left wheel speed by the same amount that we decrease the right wheel speed and vice versa). As explained in the lecture slides:

$$v_R = v_C + \frac{1}{2}K_p(z - d)$$

$$v_L = v_C - \frac{1}{2}K_p(z - d)$$

As in the forward/backward case, and with a well-chosen gain value, this simple law should give smooth behaviour because the robot will make rapid turns when there is a big difference between the measured and desired distance, and gradual turns when it is almost at the right distance.

The robot should be able to cope with walls which are somewhat curved as well as straight ones; though probably not sharp turns such as right angles.