Robotics
Practical 1: Accurate Robot Motion

Andrew Davison
a.davison@imperial.ac.uk

1 Introduction

Please divide yourselves into groups (listen in the lecture for how big the groups should be), and sign up on the page linked from the course website with the names and login IDs of your group members. Your group will have a number. At the start of the practical, come and see the TAs who will give you the robotics kit box with the number that matches your group number. Your group will keep this kit throughout term and you must return to us complete at the end of the course so please look after it! You will build robots using the contents of this kit together with Lego pieces which you can find in the open drawers in the lab — see more details below.

In this first practical you will learn how to assemble and use a Raspberry Pi based robotics kit; and then examine how wheeled robots move and in particular the uncertainty in their motion.

This practical and several others this term will be ASSESSED. There is a lot to do this week, so there are 30 marks to be gained for completing the objectives defined for today's practical, out of a total of 100 for the coursework mark for Robotics over the whole term. Assessments will take place via short demonstrations and discussion of your robots and other results to us or the lab assistants NEXT WEEK, AT THE START OF THE PRACTICAL SESSION ON THURSDAY 30th JANUARY. No submission of reports or other materials is required. We will assign marks based on our judgement of whether each objective has been successfully achieved, and we will confirm these marks to you face to face.

2 Hardware and Software Components

You are provided with a box that contains the following items:

- Raspberry Pi3 and BrickPi3: the single board computer and an interface board for Lego Mindstorms sensors and motors; these are already assembled together in a plastic case,
- Rechargable battery,
- Mains battery charger,
- Battery connection Y-cable,
- HDMI to DVI cable: to connect to a monitor.
- MicroSD card: this is for long-term storage, and plays the role of a disk drive on the RaspberryPi. It comes prepared with a ready-to-use Linux Raspbian image with the libraries and software tools you will need,
- 3 Lego Mindstorms motors, 1 ultrasonic (sonar) sensor, 2 touch sensors.

- Four cables that are used to connect either Lego motors or sensors to the BrickPi.

It is quite possible that at some moment during term, parts of this kit may be faulty (as happens sometimes with all real-world hardware!) If you have any problems, contact me or the TAs either in the lab sessions or during the week (we are all based in the Dyson Robotics Lab, which is in the William Penney Building, on the main walkway opposite the Junior Common Room and upstairs from the Data Science Institute — ring the buzzer which says Dyson Robotics Lab and someone will let you in). We have spare parts and will usually be able to help you quickly. Also check EdStem for general information and solutions to problems that others have found; the lab assistants and I will be happy to answer questions there.

In the following we describe the main components and how to set them up in detail.

2.1 Raspberry Pi and BrickPi

For those that are not familiar with **Raspberry Pi3**, it is a low-cost single board computer which runs a full Linux operating system, designed and built by the Raspberry Pi Foundation in the UK with the primary aim of promoting computing education. The Pi is based around an ARM processor within a system-on-chip architecture similar to recent generation mobile phones. It uses an SD card for long-term storage; has a WiFi chip; desktop graphics output via HDMI; and four USB ports for keyboard, mouse and other devices. It also has a general purpose hardware interface via a GPIO connector which people across the world have already used for a vast range of projects. See the huge amount of documentation online, starting here: `http://www.raspberrypi.org/faqs`.

The **BrickPi3** is an add-on for Raspberry Pi from a company in the US called Dexter Industries. Its aim is to make it easy to use the sensors and motors from Lego's Mindstorms robotics kits with Raspberry Pi. BrickPi sits on top of the Raspberry Pi's GPIO pins and provides 4 sensor ports and 4 motor ports to which Lego NXT parts can be connected (the sensor ports are labelled S1, S2, S3, S4 and the motor ports MA, MB, MC, MD). Dexter Industries also provide software libraries for Python, C and Scratch which have easy to use APIs for interfacing with the Lego hardware and this is pre-installed on our robots. See

- `https://www.dexterindustries.com/BrickPi/brickpi3-getting-started/`

for more information on BrickPi3.

You have a BrickPi/Raspberry Pi unit which is already assembled in the BrickPi perspex case.

2.2 Power and Batteries

We have rechargeable battery packs which you can use to power the BrickPi/Raspberry Pi units. Each of these comes with its own mains charger. You can charge the battery alone or via the Y-cable that connects battery and BrickPi. You can thus hot-plug/unplug the charger when plugging into the Y-cable, or charge the battery while also running the BrickPi on mains power. (Note that you can alternatively power the Raspberry Pi directly via its standard USB-C power jack, but this does not power the BrickPi, and therefore won't allow you to use sensors or motors.)

Note: the battery must be turned on for it to be charging. If you hook up the battery to the charger but don't turn it on then the red light will come on, but the battery won't be charging.

3 WiFi and Software Configuration

The first thing you should do is to put the MicroSD card into your Raspberry Pi and connect it to mains power via the Y-cable, with your rechargeable battery also connected and turned on so that it starts to charge. Use the HDMI cable we have provided to connect your RPi to a monitor in the lab, and connect a keyboard and mouse to the USB ports on the RPi, then power up using the power switch on the BrickPi and you should see Raspbian Linux booting up on the screen.

At the end of the practical, please make sure you plug the monitor, keyboard and mouse you have used back into the lab machine they came from, as a courtesy to the students that will use it next!

Set up your Raspberry Pi's WiFi and our custom interface by following the detailed instructions in this handout: `www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/wifi_setup.txt` .

If you have any trouble with these parts, please ask us for help.

Some general points:

- Make sure that whenever you turn off your robot, you properly shut down the Raspberry Pi (using the command `sudo poweroff`) before disconnecting the power supply. A number of groups neglected this last year, resulting in a lot of corrupted SD cards (and lost work).
- Back up your files regularly! Remember that your RPi has a general Linux distribution, so you can use any tools that you would normally use to do this. **Any files that you store only on your Rpi's SD card will be lost if your SD card gets corrupted or lost.** A very convenient workflow is to have a git repository with all of your robotics code. This way, you can easily edit your code on either your own or a lab machine and then just pull the edited code to your robot from the repository. With this workflow, you won't need to use ssh for code editing and you will always have a backup of your latest work.

4 Building a First Robot

Each group is given a Raspberry Pi, BrickPi, motors, sensors, batteries and SD Card, but we will not distribute Lego parts. Each group should come to get the parts they need for their robots from the drawers in the teaching labs which contain a huge amount of (mixed up) NXT Lego parts. We will leave these unlocked during term. Please help yourself to the parts you need, but be mindful that there are are many groups sharing this Lego. There should be plenty to go around but don't hoard parts you don't need, and please be cooperative if we come around to ask if we can give your spare parts for other groups! At the end of term, please put all of the Lego you have used back into the drawers.

Between tutorials, you can keep the Lego which makes up your robot along with your Pi kit and work on it in the lab out of hours or outside of the lab if you need to.

- Using the Lego kit, build a wheeled robot which you expect to be able to move and turn accurately. You should use the differential drive configuration explained in lectures, with two motors most likely directly connected to driving wheels. This week your robot will not need sensors, and only two motors. A starting design is the SimpleBot robot which is detailed as one of the projects on the BrickPi pages at: `www.dexterindustries.com/BrickPi/projects/simplebot/`. This is a small and neat differential drive robot which carries the BrickPi case comfortably and has a neat place at the back to carry a battery pack (though this will probably need some modification to carry our rechargeable battery which has a different shape). It should

be quite easy to build the SimpleBot just by looking at the pictures; and if you can't find exactly the right pieces then there are many ways to make adaptations and you are free to design your robot in a completely different way if you wish. You should certainly use a differential drive design though. Besides the driving wheels which are attached to the motors, you will need another undriven caster (like the spinning wheels on a chair) wheel or similar to support the robot's weight. Some teams use a ball in some way for this, or even a smooth Lego part which slides over the surface can work well. Your robot will move better if most of the weight is over the driving wheels. Usually within a few hours the robots from different groups already look very different from each other and you are free to follow whatever design you want.

- The robot design should include a mount for a downward-pointing pencil which can trace the robot's path (ideally this should be lightly spring-loaded so that downward pressure can be maintained — tape or rubber bands can be used to attach the pencil if required). We have spare pencils, tape and rubber bands for mounting. In order to observe its detailed motion, we will run the robot on a large sheet of stiff paper taped to the desk or floor (join two or more sheets together if necessary for more space). We will hand out large sheets of paper for this.

- Think about where the 'centre' of your robot should be defined — this will probably be at the centre of rotation when the robot turns on the spot, half-way between the two drive wheels. Try to mount the pencil to draw as close to this point as possible.

5 Programming using Python

Python is very well supported on Raspberry Pi. The general Python Tutorial at `http://docs.python.org/3/tutorial/` is well worth going through if you are new to. We will use Python3.

A Python API is provided by Dexter Industries which can be used to easily access the sensors and motors from Raspberry. It is installed in the `Dexter/BrickPi3/Software/Python` directory on your RPi. You will see the `brickpi3.py` file which contains the API we will be using, and the `Examples` directory where there are a number of example programs you can try. This first week, the `LEGO-Motor` examples are relevant. Later on, we will be using the Touch and NXT-Ultrasonic sensors.

To use the BrickPi3 robot interface from your own Python programs, you should include these statements at the top of your program:

```
import brickpi3
BP = brickpi3.BrickPi3()
```

As explained in lectures, the most important functions to control a motor connected to Port A are as follows. For each of these, change `PORT_A` to `PORT_B`, etc. for motors connected to the other ports.

- `BP.set_motor_power(BP.PORT_A, power)`: set a raw power level (in the range -100 to 100) which will make the motor move without PID control.

- `BP.set_motor_power(BP.PORT_A, BP.MOTOR_FLOAT)`: set the motor to 'float' without power, such that it can be turned by hand (the encoder can still be read so this is an interesting way to interface with a robot).

- `BP.get_motor_encoder(BP.PORT_A)`: returns the current encoder position in degrees.

- `BP.offset_motor_encoder(BP.PORT_A, BP.get_motor_encoder(BP.PORT_A))`: resets the encoder count to zero.
- `BP.set_motor_position(BP.PORT_A, degrees)`: set a position demand for the motor in degrees, and start PID control to reach it.
- `BP.set_motor_dps(BP.PORT_A, dps)`: set a velocity demand for the motor in degrees per second, and start PID control to achieve it.
- `BP.get_motor_status(BP.PORT_A)`: return a tuple of four values which indicate the current status flag, power in percent, encoder position in degrees and current velocity in degrees per second.
- `BP.set_motor_limits(BP.PORT_A, power, dps)`: set limits on the power and degrees per second that will be used in PID control. These are useful to protect your BrickPi and motors from overloading (we would recommend usually staying below 70% power).
- `BP.set_motor_position_kp(BP.PORT_A, kp)`: set PID proportional gain constant; default is 25.
- `BP.set_motor_position_kd(BP.PORT_A, kd)`: set PID differential gain constant; default is 70.
- `BP.reset_all()`: disable all motors and sensors.

Note that once a PID control demand is sent to a motor, the motor will keep running to satisfy the demand using BrickPi's control chips, while your Python program on Raspberry Pi moves on. This is good because you can get on with other processing while your robot is moving. However, it means that if you want your program to wait until the PID action is completed, you will need to run a loop to monitor the motor and check for the finish condition. For instance, you can use a `while(True):` loop which keeps reading `BP.get_motor_status(BP.PORT_A)` and checks for when the motor position and velocity have reached the values you want. Note that `BP.set_motor_position(BP.PORT_A, degrees)` may never reach exactly the position you request so you may need to use a threshold in your test.

Another important implication of this is that if your program exits without stopping the motors then they will keep moving. (This could even happen if your Raspberry Pi reboots due to low power.) A good idea is to at least ensure that if you use Control-C to stop your Python program then your robot will stop. This can be achieved by using a `try: except KeyboardInterrupt:` structure around your program to ensure that `BP.reset_all()` is called if the program is interrupted. This is done in all of the example programs provided by Dexter so you can check the exact syntax.

5.1 Programming Workflow

The main options for programming are the following.

- Work directly on the Raspberry Pi using a plugged in keyboard and screen using an editor such as `nano` in console mode. Good for testing very basic capabilities with a non-moving robot.
- Work on the Raspberry Pi using the X Windows graphical environment (start it using `startx`) and a keyboard and mouse, e.g. using the `geany` editor.

- Work from a terminal on a PC which is ssh'd into the Pi over WiFi (using `nano`. Good for changing and debugging a program interactively.
- Edit programs using the editor/IDE of your choice on a PC and move them across to the Pi for testing. Using git is probably the best way to do this, and this is the best plan for significant programming.
- Using our web interface which includes a built-in Python editor: `https://www.doc.ic.ac.uk/~ajd/robotics/index.cgi`
- Using vscode on your local machine, you can SSH into the PI via the Remote-SSH plugin, which allows you to create/edit remote files and directories on the Pi. The terminal you get within vscode will be pointing to the Pi's shell.

`nano` is a simple console-based text editor which is pre-installed on Raspberry Pi. You can easily install other editors that you might prefer (emacs, vi, etc.) using `sudo apt-get`. `geany` is a nice simple editor which is good for Python within X Windows and runs well on RPi.

As in all software development, and particularly since you are working in groups, we strongly recommend the use of source code management software like git to share and safely backup your code among group members. Again, we remind you that it is risky to store any important programs only on your Pi's SD card in case it gets corrupted and the files are lost. If your SD card does get corrupted, you can come and get a freshly installed one from us.

6 Practical Assessed Objectives

6.1 Show us a Built and Working Pi/BrickPi Robot with Working WiFi and Motors (6 Marks)

We will expect you to demonstrate a fully built and working robot running your programs; probably we can assess this part while watching your robot driving in the square trajectory that comes later.

6.2 Investigating And Understanding Motor Control (4 Marks)

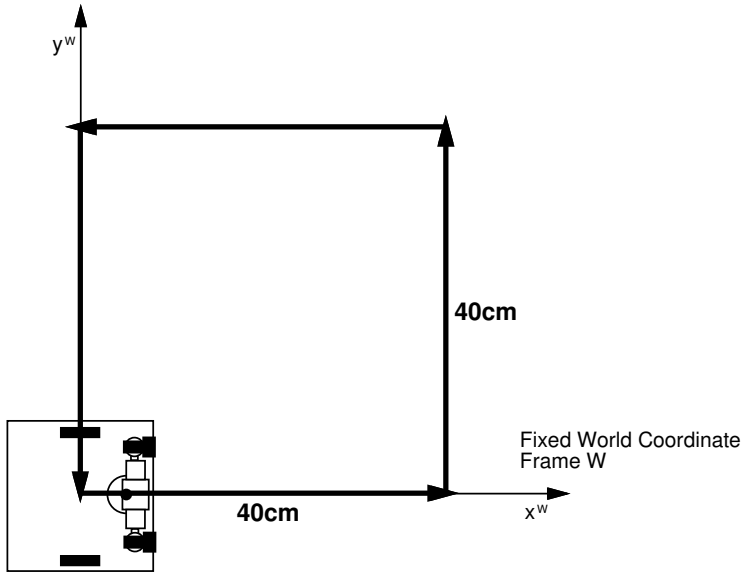
Copy the `LEGO-Motor_Power.py`, `LEGO-Motor_Position.py` and `LEGO-Motor_DPS.py` example programs from `Dexter/BrickPi3/Software/Python/` to your `prac-files` directory so that you can edit them without changing the originals. In each of these programs, two motors are connected to the BrickPi. One motor is set to float without power, but its encoder is continually read by the program and you can turn it by hand to set a demand for the other motor, either in terms of a raw power value, a PID position demand or a PID velocity demand (note that in the programs as they were written by Dexter, different choices of the motor ports A, B, C, D are used in the different programs, but this is all explained in the programs' comments — it is easy to change the programs such that they all use the same motors for control and demand and you should do this).

You will see that the `LEGO-Motor_DPS.py` has a loop which repeatedly calls `BP.get_motor_status(BP.PORT_A)` and prints the output. This is very interesting because you can observe the relationship between power, position (degrees) and velocity (degrees per second). Add the same functionality to the other two programs so that they continually print the motor status as they run. Then experiment informally with each of the programs to get a feel for how they each work, and make a few notes on each to discuss with us in the assessment. This should be done with your robot not driving on the ground, but held or upside down so that the wheels don't touch the ground. What

happens in each case if you use your hand to try to gently slow down the running motor? What can you observe in the power and velocity of the motor? Why?

We will assess this part by discussing it with your group, and perhaps by asking you to run these programs to show us.

6.3 Distance and Rotation Calibration for Accurate Driving (10 Marks)



We have tape measures and scissors to borrow, and a big roll of stiff paper. Using a pen, **on your paper measure and mark out an accurate square course of side length 40cm for the robot to follow, starting at one corner, defining a coordinate frame as shown above**.

Write a program to send the robot around one complete circuit, returning to its original location and orientation, by moving forward 40cm and turning left on the spot 90° four times. The aim is that the centre of the robot, the point between its wheels, should ideally exactly trace out the square. You can choose whichever control method you think will work best, but probably simple position control of your motors is sufficient to get quite accurate motion.

You will need to calibrate your robot's motion in a straight line and rotation on the spot so that rotations of the motors turn precisely into translation and rotation of the robots. As explained in lectures, we recommend doing this by trial and error using your whole robot, rather than using calculations. A well calibrated robot should be able to get all round the square and back to where it started to within a couple of centimetres.

We will ask you to demonstrate your robot driving in an accurate square to us.

6.4 Experiment to Measure Robot Accuracy (8 Marks)

Now mount the pencil on your robot so that the robot draws a visible line as it drives. The pencil should ideally draw exactly under the robot centre between the two wheels so you may need to use tape or a rubber band to mount it nicely.

Place the robot on top of your paper where you have an accurately drawn square in pen. Carry out the square motion 10 times, with the pencil down to draw a trace on the paper. Each time pick up the robot and replace it carefully at the start position. You will get 10 drawn trajectories, each one a little bit different. Draw a big 'X' at the final location of the robot after each run, which should be back near the

origin, and measure its (x, y) coordinates in cm. You will see that the robot is very unlikely to follow exactly the same path every time! What is the approximate scatter range of the different outcomes? Is there a *systematic error* (meaning that the final locations are consistently different from the ideal result with an error in the same direction)? Is there much scatter, such that the points are quite spread out from each other? SAVE THE PIECE OF PAPER YOUR ROBOT DRAWS ON TO SHOW AT THE ASSESSMENT.

6.5 Calculating a Covariance Matrix (2 Marks)

Manually, or via a short program in the programming environment and language of your choice, calculate the *covariance matrix*, which is an explicit measure of the scatter range, or uncertainty in the motion of the previous section. With $N = 10$ individual final location measurements (x_i, y_i), the covariance matrix is a matrix of four values as follows:

$$\mathtt{P} = \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 & \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}) \\ \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})(x_i - \bar{x}) & \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2 \end{bmatrix},$$

where $\bar{x} = \frac{1}{N}\sum_{i=1}^{N}(x_i)$ and $\bar{y} = \frac{1}{N}\sum_{i=1}^{N}(y_i)$ are the coordinates of the mean final location. **Use centimetre units.** Note that this and all covariance matrices are symmetric. As a sanity check, the square roots of the two values on the diagonal should represent the standard deviations of the scatter in the x and y directions respectively — i.e. we would expect a few centimetres after a 40cm square motion. We will check to see that the covariances you calculate tie up with your trajectories plotted on the paper.

7 For Further Thought

- Which causes a larger effect on your robot, imprecision in drive distance or rotation angle?
- Can you think of any robot designs which would be able to move more precisely?
- How should we go about equipping a robot to recover from the *motion drift* we have observed in this experiment?