Robotics

Week 2 Practical: Getting Started with BrickPi; Accurate Robot Motion

Andrew Davison
ajd@doc.ic.ac.uk

1 Introduction

Please divide yourselves into groups (listen in the lecture for how big the groups should be), and come and see me and the lab assistants at the start of the lab. We will ask you to fill in a form with the names and login IDs of the group members, and give you a box which contains a Raspberry Pi + BrickPi kit which you will be able to keep throughout term and return to us at the end of the course. You will be given the key to a locker in the lab which you can use to store your kit in. One member of the group will be formally responsible for keeping the kit safe throughout term.

In this practical you will first 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 essential and optional 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 AT THE START OF THE PRACTICAL SESSION NEXT WEEK. 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 every week by email. A dummy coursework has been created on CATE. You don't need to do anything about this yet, but at the end of term we will ask you to use this to register the members of your group and we will assign your final coursework marks here.

2 Hardware and Software Components

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

- Raspberry Pi and Brickpi: the single board computer and an interface board to Lego Mindstorms sensors and motors; these are already assembled together in a plastic case,
- Rechargable battery,
- Battery charger,
- Battery connection Y-cable,
- HDMI to DVI cable: to connect to a monitor,
- SD card: this is for data storage, and it comes prepared with and ready-to-use image with all the libraries and software tools you will need,

- Wifi USB dongle
- 3 Lego Mindstorms motors, 1 ultrasonic (sonar) sensor, 2 light sensors and 2 touch sensors plus 3 cables.

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 lab assistants 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 oppotise 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 Piazza 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 Pi**, 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 network connector; desktop graphics output via HDMI; and two USB ports for keyboard, mouse or WiFi dongle. 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 **BrickPi** is a 3rd party 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 NXT robotics kits with Raspberry Pi. BrickPi sits on top of the Raspberry Pi's GPIO pins and provides 5 sensor ports and 4 motor ports to which Lego NXT parts can be connected. Dexter Industries also provide software libraries for Python, C and Scratch which have easy to use APIs for interfacing with the Lego hardware.

We will be distributing BrickPi/Raspberry Pi units which are already assembled in the BrickPi perspex case which has holes for attaching Lego. There should also be coloured stickers which label each sensor and motor port on the BrickPi. See

- `www.dexterindustries.com/BrickPi/getting-started/`, and
- `www.dexterindustries.com/BrickPi/getting-started/attaching-lego/`

for more information on BrickPi, and instructions in case you need to assemble a kit yourself.

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 wall charger. You can charge the battery alone or via the Y-cable that connects battery and Pi. You can thus hot-plug/unplug the charger when plugging into the Y-cable. Note that powering the Raspberry Pi directly via a standard micro-USB power jack 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.

2.3 SD Card and WiFi Configuration

A Raspberry Pi uses a standard SD card as its long term storage, and Raspbian is a variant of Debian Linux supported by the Raspberry Pi Foundation which is the most widely used operating system. Dexter Industries have their own version of Raspbian which builds in support for the BrickPi hardware and interface libraries. In the Department of Computing, thanks to Duncan White and CSG, we have made our own somewhat modified version of BrickPi Raspbian which you should work with. In particular, this version has been set up to provide WiFi networking via the USB dongles you have with each kit, connecting to the college network.

We distribute a pre-burned SD card with our DoC/BrickPi Raspbian image ready to go. Note that our distribution is set up to boot into the text console, rather than launching the X-Windows desktop. The Raspberry Pi is quite capable of running X-Windows, and you can type `startx` if you want it. The main current reason not to, besides keeping a lightweight approach which leaves the full resources of the Pi's processor available for robotics, is that the standard Raspberry Pi has only 2 USB sockets. If we have a WiFi dongle and keyboard plugged in there is no space for a mouse unless you use a USB hub, and in general we have found that a hub may need to be externally powered to guarantee good WiFi operation.

Set up your Raspberry Pi (including WiFi) by following the detailed instructions in the second handout, or here: `www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/wifi_setup.txt` .

Once you boot up your Raspberry Pi successfully, you will find a folder called `BrickPi/` in your home directory. This contains examples and can be used as workspace. You will also find a folder `Imperial/` which contains the C++ project and boost.python interface used to compile our custom interface. You should not have to do anything with it. Furthermore, you'll also find the folder `DexterInd/` that contains the C and Python interfaces provided by the BrickPi manufacturer. These are less accurate than our custom ones and you should not have to use them.

If for some reason you need to burn a completely new DoC/BrickPi image to your SD Card (this will erase everything on the card!) you can obtain the image from the directory: `http://www.doc.ic.ac.uk/~ajd/files/2018.10.12_DoCBrick.Pi_4G.tar.gz`

3 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 7–18 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!

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. Each group will be given the key to their own locker (with a number which matches your box) to keep your equipment safe.

- Using the Lego kit, build a wheeled robot which you expect to be able to move and turn accurately — almost certainly a robot with the differential drive configuration and two motors. This week your robot will not need sensors, and only two motors. A great 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 in particular has a neat place at the back to carry a battery pack (though this will probably need some modification to carry our new rechargeable battery packs). 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.

- 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 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).
- 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, between the two drive wheels. Try to mount the pencil as close to this point as possible, or at least on the central front/back axis of the robot.

4 Programming using Python

The main programming language we plan to use in the course is Python, which is excellent for robot programming due to its ease of use and the wide range of modules which can easily be imported. It is very well supported on Raspberry Pi. The general Python Tutorial at `http://docs.python.org/2/tutorial/` is well worth going through.

4.1 Using Our Custom Python Interface

A Python API is provided by Dexter Industries which can be used to easily access the sensors and motors from Raspberry Pi (in the `DexterInd/` directory); but when we used this last year we found it was difficult to control the motors and sensors accurately.

In order to provide more accurate closed-loop (rotational) velocity and angle (position) control, we have developed a custom library in C++ that will run a background process which constantly gets sensor values and sets motor outputs. You can interface it through Python. We strongly recommend only using our interface instead of Dexter's. Our interface is loaded by Python with the command `import brickpi`.

The best way to understand the API is via the example programs `motor_velocity_example.py`, `motor_position_example.py`, and `ultrasonic_example.py`.

You can run Python programs as e.g. `python myprog.py`. But Python also features an interactive mode from the command line — just type `python`. The interactive mode lets you access our API documentaion: you can use `help(func)` for help on a function or module, e.g. `help(brickpi)` or `help(brickpi.Interface.setMotorAngleReference)`.

4.2 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 in console mode. Good for testing very basic capabilities with a non-moving robot.
- Work on the Raspberry Pi using X Windows and a keyboard and mouse, e.g. using the `geany` editor. You will need to disconnect your WiFi dongle to plug the mouse in.
- Work from a terminal on a PC which is ssh'd into the Pi over WiFi. Good for changing and debugging a program interactively.
- Edit programs using the editor/IDE of your choice on a PC and scp them across to the Pi for testing. For significant programming this is surely the best option.

`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 recommend the use of source code management software like git to share and safely backup your code among group members.

5 Practical Assessed Objectives

5.1 Show us a Built and Working Pi/BrickPi Robot with Working WiFi and Motors (4 Marks)

We will expect you to demonstrate your working robot, ideally programmed to carry out the square trajectory explained below.

5.2 Controller Tuning (8 Marks)

Use the `motors_position_example.py` program to tune better gains of the feedback controller. Using `brickpi.Interface.startLogging(...)`, a logfile is created that you can use to quantitatively assess the performance of the individual controllers, e.g. using Python. Make sure to also stop the logging with `brickpi.Interface.stopLogging(...)`. Perform the performance assessment in a standardised test. This means that your robot should always use the same angle reference sequence and actually drive on the floor. The logfile format is:

```
time[nsec] referenceAngle[0] angle[0] referenceAngle[1] angle[1]
```

Note that the reference angles that are logged are pre-filtered by application of your velocity and acceleration limits. Tune the PID as well as feed-forward and minPWM (for the deadband compensation) parameters for the motors individually, so you achieve equivalent performance with both motors. In an ideal world, the parameter sets would be equal; but in reality, the motors will always behave slightly differently. You will also want to set reasonable values for the velocity and acceleration limits (for the reference filtering) — but these need to be equal for both motors, as they determine the shape of the angle reference. Make sure not to mix tuning of these limits with tuning of the other parameters.

Warning: think before running a script with your parameters. Unreasonable parameters can lead to excessive current draw with Pi shutdowns due to voltage drop or even burning of motor controller chips on the BrickPi. Specifically, you should tune the P-gain values by altering them in steps of maximum 100, and should never exceed a total of 1000.

We will be able to get a good idea of whether your controller is well tuned by observing your robot running `motor_position_example.py` after tuning. However for maximum marks on this section

you should be able to show and explain to us, either printed or on screen, graphs you have plotted of the data from the logfile before and after tuning. Your graphs should ideally show reference angle, actual angle, and angle error against time for a reasonable number of timesteps. Of course these graphs, where it is easy to see phenomena such as undershoot or oscillation, will help you in tuning and understanding your controller.

You can use the tools of your choice on a PC to make plots from the logfile data, such as matlab or Python. In Python, matplotlib is a very easy to use tool for plotting graphs: see `http://matplotlib.org/users/pyplot_tutorial.html` for a tutorial.

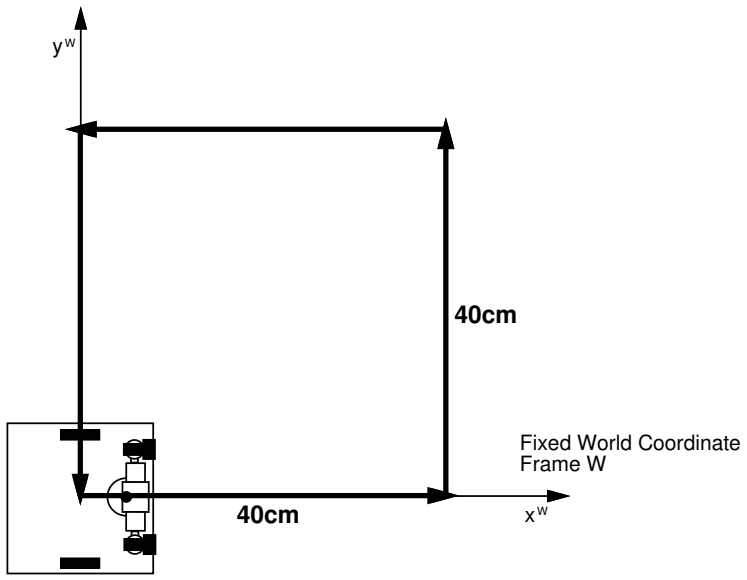
5.3 Distance and Rotation Calibration (5 Marks)

Calibrate your robot's drive system. Mark out a range of 40cm on the paper (we have tape measures and scissors to borrow, and a big roll of paper) and adjust the motor angle increase/decrease within your program to values such that it covers this distance as accurately as possible. If you achieved proper tuning of your controllers, you should be able to get nice, straight and repeatable motion patterns.

Next calibrate the robot's on-the-spot rotation. Work out how to to achieve a 90° rotation accurately and define `Left90deg()` and `Right90deg()` subroutines.

Write a simple test Python test program which demonstrates forward and backward motions of 40cm and left and right turns on-the-spot of 90° with pauses in between.

5.4 Driving Accurately in a Square and Observing Drift (9 Marks)



Using a pen, **on your paper 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**. Based on your earlier calibration, write a program to send the robot around one complete circuit, returning to its original location and orientation.

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. Draw a big 'X' at the final location of the robot after each run, hopefully back near the origin, and measure its (x, y) coordinates. 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)? SAVE THE PIECE OF PAPER YOUR ROBOT DRAWS ON TO SHOW AT THE ASSESSMENT. It is more important that you do the experiment with a focus on equal conditions every iteration rather than worry too much about getting extremely precise movement.

5.5 Calculating a Covariance Matrix (4 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.

6 For Further Thought

- Which causes a larger effect in your robot, uncertainty 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?

Acknowledgements

Thank you to Stefan Leutenegger; Duncan White, Geoff Bruce and the rest of CSG; Maria Valera-Espina; and Jan Jachnik, Robert Lukierski, Lukas Platinsky, Jacek Zienkiewicz, Tristan Laidlow, Andrea Nicastro and Jan Czarnowski for their support.