[ELE00118M] MSc Practical Robotics (PRAR)
[ELE00103M] Year 4 Robotics Module

Lab Session 1 Handout

Lab 1: Robot Kit Assembly and RPi Programming

Lecturer: Mark A Post (mark.post@york.ac.uk)

Demonstrator: Robert Woolley (rw1445@york.ac.uk)

Technician: Mike Angus

1. Aims and Objectives

To assemble the basic chassis of a robot using the parts provided in the York Robotics Kit and be introduced to C++ and Python programming on an embedded programming platform, the Raspberry Pi.

2. Learning outcomes

- Familiarization with the parts of a robot and how they can be assembled
- Construction of the platform on which the labs this term will be based
- Understanding the programming and functions of microcontrollers
- Familiarity with the Raspberry Pi platform and tools for future labs
- Practice using GPIO pins and a serial port for simple communications

3. Hardware and Software

You have been provided with a Robotics Kit that contains these parts:

- Raspberry Pi 4
- Raspberry Pi Camera
- Printed ArUco tags and shapes for vision recognition
- Micro SD card with Raspberry Pi OS loaded
- USB to Serial UART cable
- USB-C Power Bank battery
- USB-C Raspberry Pi mains power supply and adapters

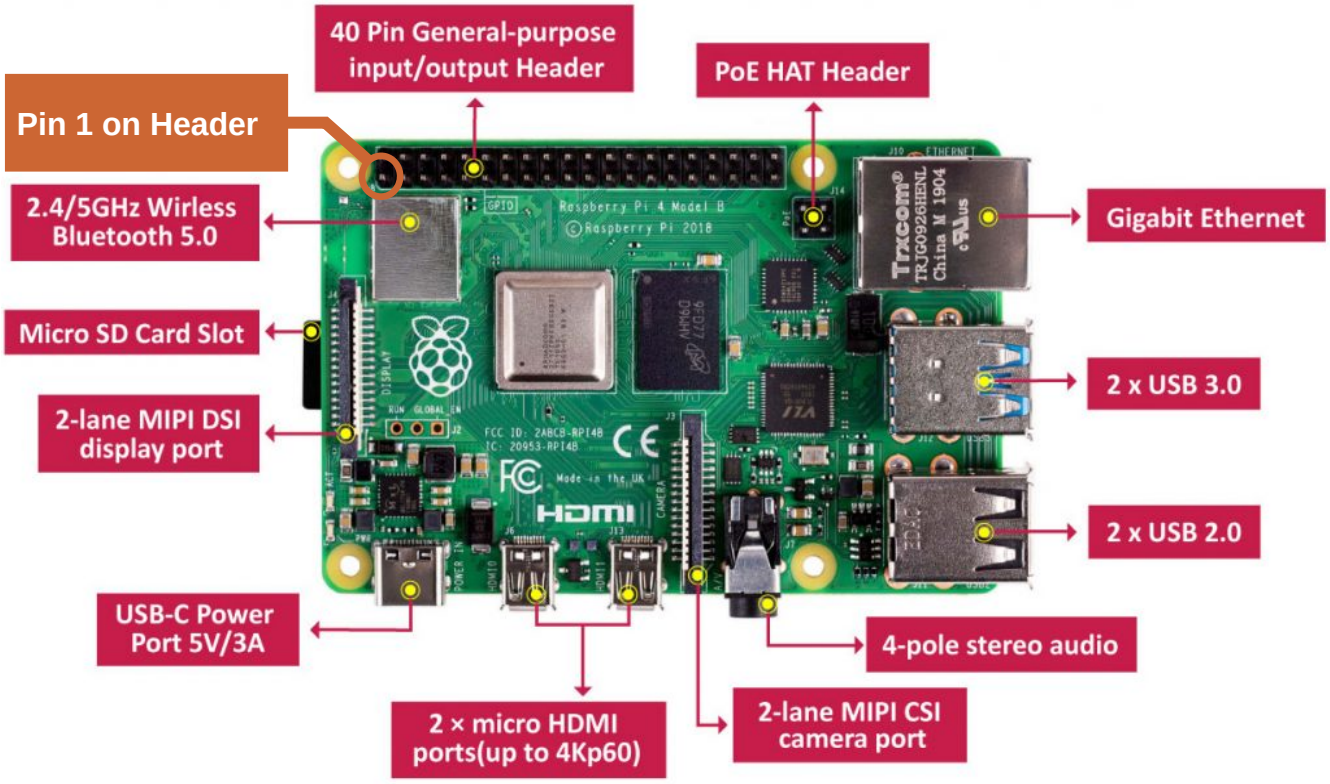
- A 2m Ethernet cable
- 2 micro-metal gear motors
- 2 “Moon Buggy” wheels
- Ball castor
- RasPiO Pro Hat
- 40x male-male breadboard jumper wires
- Sharp GP2Y0E02B I2C IR Range Sensor
- TB6612FNG Dual Motor Driver
- 2x 6-way 2mm to 2.54mm cables
- 2x 6-way breadboard pin adaptors for motor cables
- 1x 3-way breadboard pin adaptor for USB serial cable
- 3D-printed stand for camera + IR sensor
- 3D-printed camera bracket
- 2x 3D-printed motor housings
- 200mm Velcro strip (for battery)
- 2x 60mm Velcro strips (for cables)
- 5mm acrylic spacer (for castor)
- 2x 3mm acrylic braces (for chassis)
- A 3mm acrylic robot chassis in two parts
- 4x 11mm M2.5 standoffs (for ProHat)
- 4x 5mm M2.5 standoffs (for Raspberry Pi)
- 6x 12mm M3 screws (for camera stand and motor mounts) [+1 spare]
- 6x M3 nuts [+1 spare]
- 4x 6mm M2.5 screws (for Raspberry Pi mounting) [+1 spare]
- 4x M2.5 nuts [+1 spare]
- 2x 12mm M2 screws (for camera bracket on camera stand) [+1 spare]
- 2x 6mm M2 screws (for camera on camera bracket) [+1 spare]
- 4x M2 nuts [+1 spare]

To assemble your robot, you will be following the guidance provided in the document “**MSc/MEng Robotics Kit**”, which is included on the module webpage. This document will guide you step-by-step with detailed instructions for assembling the included components as you perform your labs this term. Keep it nearby and refer back to it for help whenever you are assembling part of your robot in a lab.

THE RASPBERRY PI PLATFORM

You will be working to develop some simple programs using the Raspberry Pi 4. You have a choice for many of the labs whether to use the C language or the Python

language, or a combination of them. Both are essential for modern roboticists to know. C examples are given in the earlier labs because it allows lower-level and more efficient programming in Linux, but Python is favoured in the later vision and ROS labs for its simplicity and user-friendliness, but there is online documentation available for OpenCV and ROS that allows you to do equivalent tasks with both languages and you are welcome to use whichever language you are most comfortable with. The Raspberry Pi 4 is based on a Broadcom BCM2711 SoC with a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor. The features of the Raspberry Pi 4 are shown below:



The Raspberry Pi 4 combines the benefits of an embedded processor board and a desktop computer in a low-cost package. The board includes a USB-C port used to power it (high-current 3A or more power supply recommended), two micro-HDMI ports that can be connected to display monitors, Wifi with Bluetooth for wireless networking, native Gigabit Ethernet for wired networking, and both USB 2.0 and USB 3.0 ports for connecting general PC peripherals. At the same time, a 40-pin header allows you to send and receive 3.3V digital electronic signals directly using the pins as General Purpose Input-Output (GPIO) or other peripherals such as:

- Serial Universal Asynchronous Receiver/Transmitter (UART) ports (*TXD*, *RXD*)
- Inter-Integrated Circuit (I^2C) bus ports (*SDA*, *SCL*)
- Serial Peripheral Interface (SPI) bus ports (*MOSI*, *MISO*, *SCK*, *CE*).

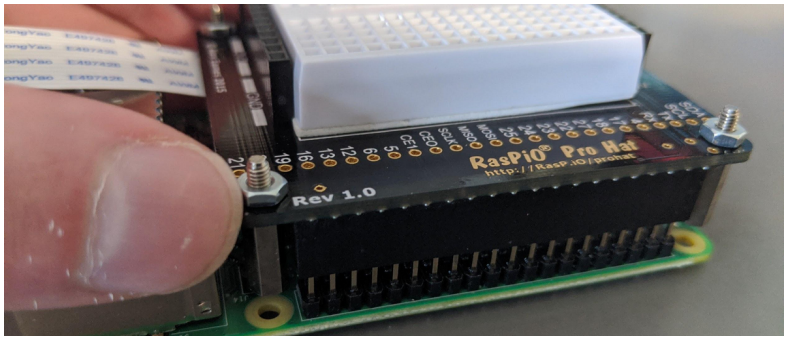
The 40-pin header connects directly to the pins on the SoC chip, and therefore are **very sensitive to voltages above 3.3V and electrostatic discharge**. This includes the 5V power pins on the board (pins 2 and 4) – surprising as it may seem, they **will** permanently damage any of the other pins on the header if connected directly. Keep this in mind and take care when wiring circuits not to connect Power pins incorrectly or to touch any pins without first grounding yourself (touching with your fingers a grounded piece of metal such as a bare water pipe or a USB port on a plugged-in computer). The complete pinout is shown below (pin 1 is to the left on the diagram above).


3v3 Power 1
2 5v Power
GPIO 2 4 SPI3 MOSI
SDA1 4 SDA3
3
4 5v Power
GPIO 3 4 SPI3 SCLK
SCL1 4 SCL3
5
6 Ground
GPIO 4 4 SPI4 CE0 N
4 TXD3 4 SDA3
7
8 TXD0 4 TXD1 GPIO 14
4 SPI5 MOSI
Ground 9
10 RXD0 4 RXD1 GPIO 15
4 SPI5 SCLK
GPIO 17
SPI1 CE1 N
11
12 GPIO 18
SPI1 CEO N 4 SPI6 CEO N
GPIO 27
4 SPI6 CE1 N
13
14 Ground
GPIO 22
4 SDA 6
15
16 GPIO 23
4 SCL6
3v3 Power 17
18 GPIO 24
4 SPI3 CE1 N
GPIO 10 SPI0 MOSI
4 SDA5
19
20 Ground
GPIO 9 SPI0 MISO
4 RXD4 4 SCL4
21
22 GPIO 25
4 SPI4 CE1 N
GPIO 11 SPI0 SCLK
4 SCL5
23
24 4 SDA4 GPIO 8
SPI0 CE0 N 4 TXD4
Ground 25
26 4 SCL4 GPIO 7
SPI0 CE1 4 SPI4 SCLK
GPIO 0 4 SPI3 CE0 N
4 TXD2 SDA0 4 SDA6
27
28 4 SPI3 MISO GPIO 1
SCL0 4 SCL6 4 RXD2
GPIO 5 4 SPI4 MISO
4 RXD3 4 SCL3
29
30 Ground
GPIO 6 4 SPI4 MOSI
4 SDA4
31
32 4 SDA5 GPIO 12
4 SPI5 CE0 N 4 TXD5
GPIO 13 4 SPI5 MISO
4 RXD5 4 SCL5
33
34 Ground
GPIO 19
SPI1 MISO 4 SPI6 MISO
35
36 GPIO 16
4 SPI1 CE2 N
GPIO 26
4 SPI5 CE1 N
37
38 GPIO 20
SPI1 MOSI 4 SPI6 MOSI
Ground 39
40 GPIO 21
SPI1 SCLK 4 SPI6 SCLK
General Purpose Input Output
SPI (Serial Peripheral Interface)
I2C (Inter-Integrated Circuit)
UART (Universal Asynchronous Receiver / Transmitter)
Ground (GND)
5v Power
3v Power
2 Physical Pin Number
4 Pi 4 Only

4. Pre-Lab Preparation

- Have at least a cursory read through this lab script from beginning to end to familiarize yourself with what you will be doing in this practical exercise.
- If you are using your own PC for lab work, download and install PuTTY for your platform (https://www.putty.org/) which is recommended for Serial and SSH connections. Other alternatives include HTerm for Windows, the Serial app for Mac, or the picocom package for Linux.
- Look through the provided robot kit materials to familiarize yourself
- Plug the USB power bank battery from the robot kit in to a USB port or standard USB charger to charge it in preparation for use
- Assemble the Raspberry Pi and the RasPiO Pro Hat as described below, and read its manual at https://rasp.io/wp-content/uploads/2016/04/Pro-Hat-GPIO-Zero.pdf

To make it easier to connect to electronic circuits, and to prevent accidental damage to the delicate Raspberry Pi pins, we connect the RasPiO Pro Hat shown below to the top of the Raspberry Pi. Take care to ensure that the Hat is aligned over top of the Raspberry Pi board and that the connector on the bottom of the Hat is evenly lined up with the 40-pin header on the Raspberry Pi as shown below before powering it on.



The RasPiO Pro Hat has a solderless prototyping board on top surrounded by a row of pin sockets. The pin sockets connect to each of the the Raspberry Pi pins on the 40-pin header though a protection circuit with a Zener diode in parallel to GND (to prevent over-voltage if more than 3.3V is applied) and a 330Ω resistor in series with the pin (to prevent over-current, limiting the current out of a pin to just 10mA). While assembling parts on your robot please refer to the “Wiring Instructions” section starting on page 50 of the document **"MSc/MEng Robotics Kit" (available on the module webpage)** for assembly guidance and tips.

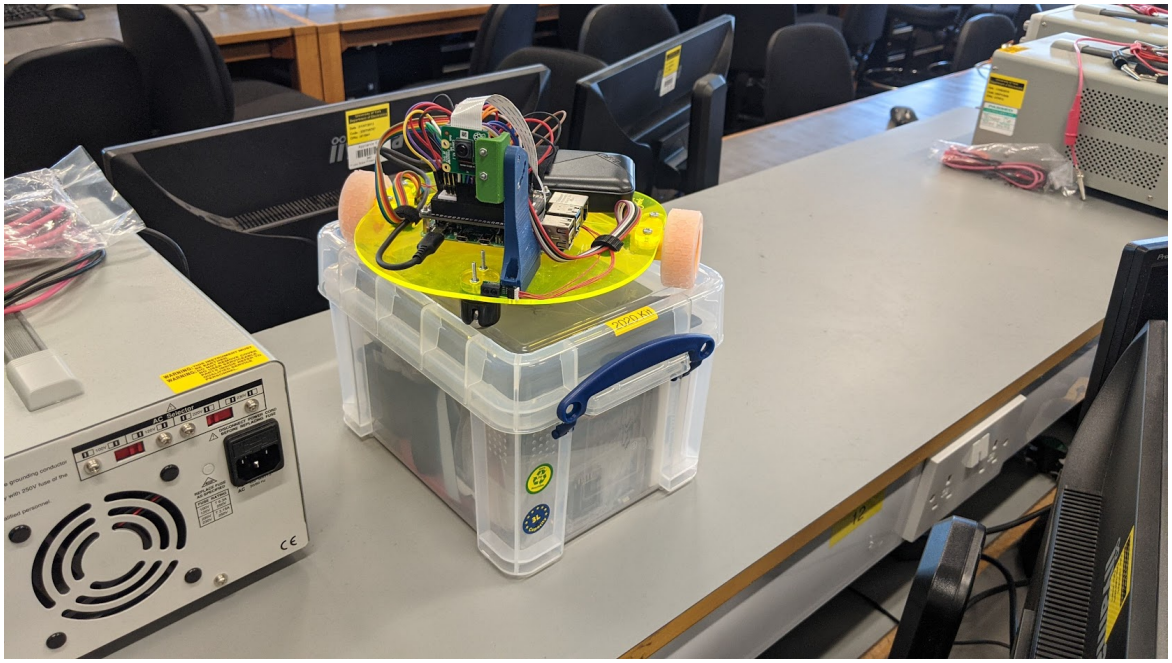
NOTE: Make sure to back up your work frequently from the Raspberry Pi, at least every lab, as small errors can make the SD card unreadable and destroy files!

5. Tasks

TASK 1: ASSEMBLE THE CHASSIS OF YOUR ROBOT

Using the Robotics Kit instructional document, complete assembly steps 1-43 that are from the start of the document to page 49, just before “Wiring Instructions”. This will allow you to have the basic chassis of your robot assembled. The following labs will refer back to the relevant content in “Wiring Instructions” as they focus on wiring and programming your robot.

Below is a photo of how you can store your robot in the lab between sessions. Place your assembled robot on the box lid as shown, with your name clearly visible (each box will have a label with your name on it). **Please make sure to store any cables, power supplies, tools, and parts not currently used on the robot inside your box** as the workspace must be clear between lab sessions.

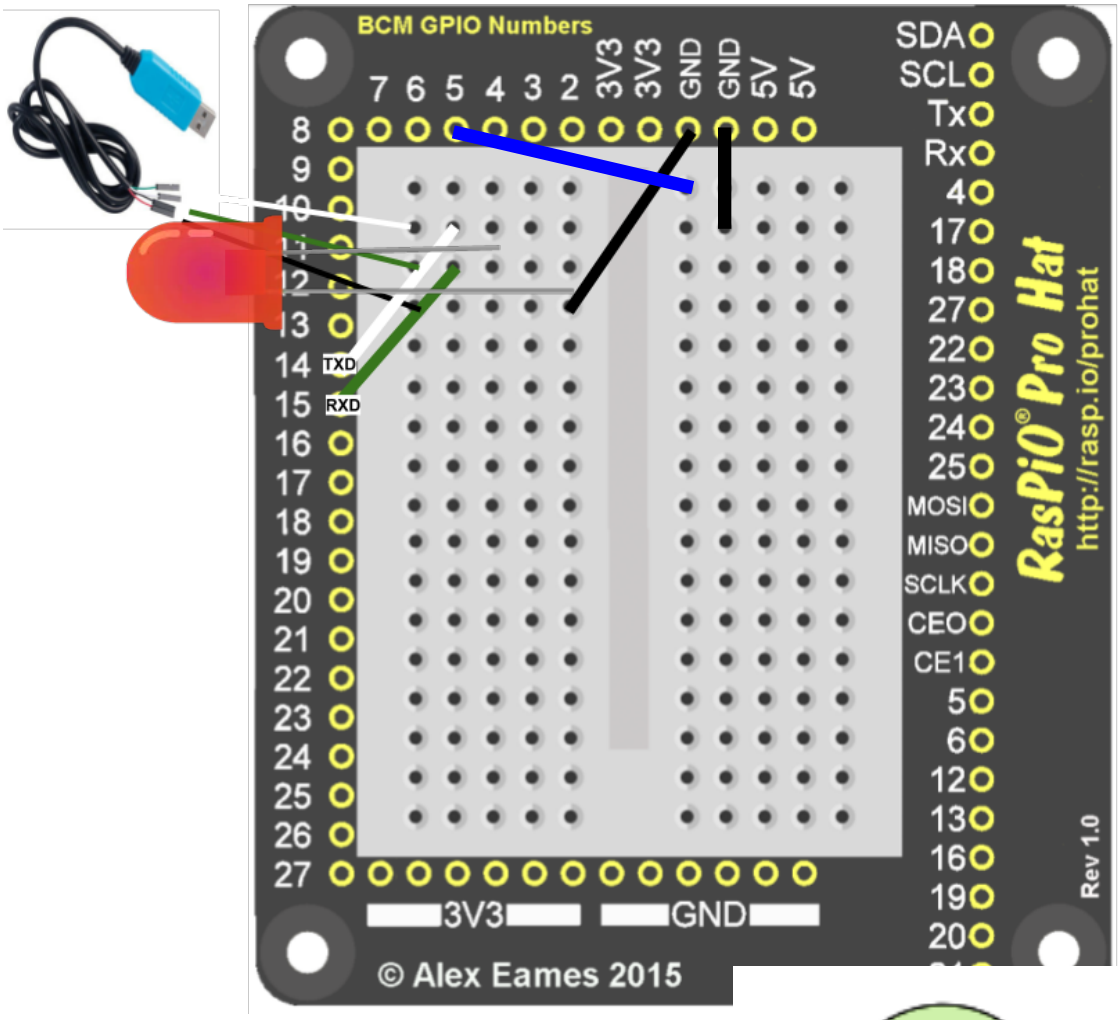


TASK 2: WIRE TOGETHER SERIAL COMMUNICATIONS AND AN LED LIGHT

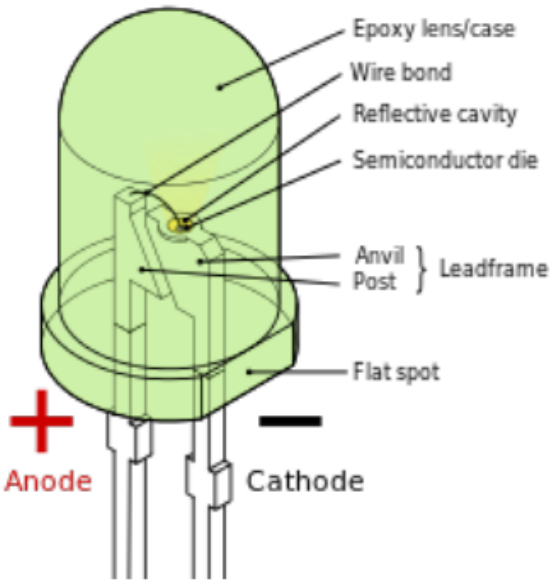
Robots rarely use Keyboard, Video, and Mouse (KVM) interfaces. You are welcome to connect these peripherals to the Raspberry Pi and enable the graphical desktop with the `raspi-config` command line utility, where you will be able to use the variety of graphical applications available with the Raspberry Pi OS SD card image provided.

However, in the labs for this module we will be connecting to the Raspberry Pi the way one would normally use a computer to program a robot – using your PC as a host that communicates to the robot over a serial or network link.

First, connect a USB to TTL Serial converter to the UART pins on the RasPiO Hat. This will let you communicate with the Pi using a serial terminal program such as picocom or hterm. The black wire is GND, the green connects to RXD (pin 15), and the blue to TXD (pin 15). It is recommended to use wire jumpers in between as shown below.



Also, take a red LED and connect it with the longer lead (+) connected to pin 5 and the shorter lead (-) connected to GND. Again using jumpers are recommended as in the diagram.



Note about LEDs: *Through-hole LEDs are semiconductor devices with an anode (positive+) and cathode (negative-). The anode normally has a longer lead, and the cathode*

usually has matching flat spot on the plastic casing. They have a rated forward-voltage (typically 1.8V for red, 2.2V for green) and current (for most LEDs up to about 20mA, except for high-brightness types). We need a resistor in series with the LED to provide this current, using the formula: As the GPIO pins are 3.3V, the 330Ω resistor already included in the RasPiO hat is usually a good choice – it will typically limit the current to around 5mA with a red LED. It is very important we carefully limit the currents through GPIO pins on microcontrollers as if limits are exceeded either the LED or microcontroller pin can very quickly be damaged beyond repair – do NOT attempt to connect an LED directly to a microcontroller pin without a resistor in series!

TASK 3: COMMUNICATE WITH THE RASPBERRY PI

Log on to your PC or a lab PC. If you are in the lab, all the PCs are dual-boot and you are usually able to use either Ubuntu Linux or Windows for simple programming tasks (though Linux is preferred in general by roboticists for its power and flexibility). If you have not used Linux before this is probably a good time to give it a go as the Raspberry Pi runs Linux and you will likely need to use it quite a bit later in the course – on a lab PC you need to restart the machine and select Linux from the boot menu.

If you are using your own PC at home, you can install the software recommended for your labs yourself. If you have a PC running Ubuntu Linux or another Debian-based Linux distribution you can install all the software needed using the apt utility. On Windows you will need to download and install the packages separately, or you can use the VirtualBox hypervisor software to run a pre-prepared Ubuntu installation that has all the software you might need for the course already with no changes to your PC.

Next, connect the USB to TTL Serial converter to a USB port on your PC. If you have a wired home network you might also want to connect the Ethernet port to your home network as it is faster and more efficient than using Wifi. If you put your Pi on a network you will have the option of connecting to it over the network with Secure Shell (SSH) which is faster and more reliable than serial.

Plug the MicroSD card loaded with an image of the Debian Linux-based Raspberry Pi OS into the MicroSD slot on the bottom of the Raspberry Pi. The metal pins should be facing upwards, against the board. Do not force the card in if it doesn't fit at first.

Finally, connect the power bank battery to the USB-C port on the Raspberry Pi with a USB-C cable. The red light on the Raspberry Pi should light up and the green light adjacent to it should start flashing to indicate CPU activity. In a few seconds you should see a login prompt in the serial terminal. Enter the username "pi" and default password

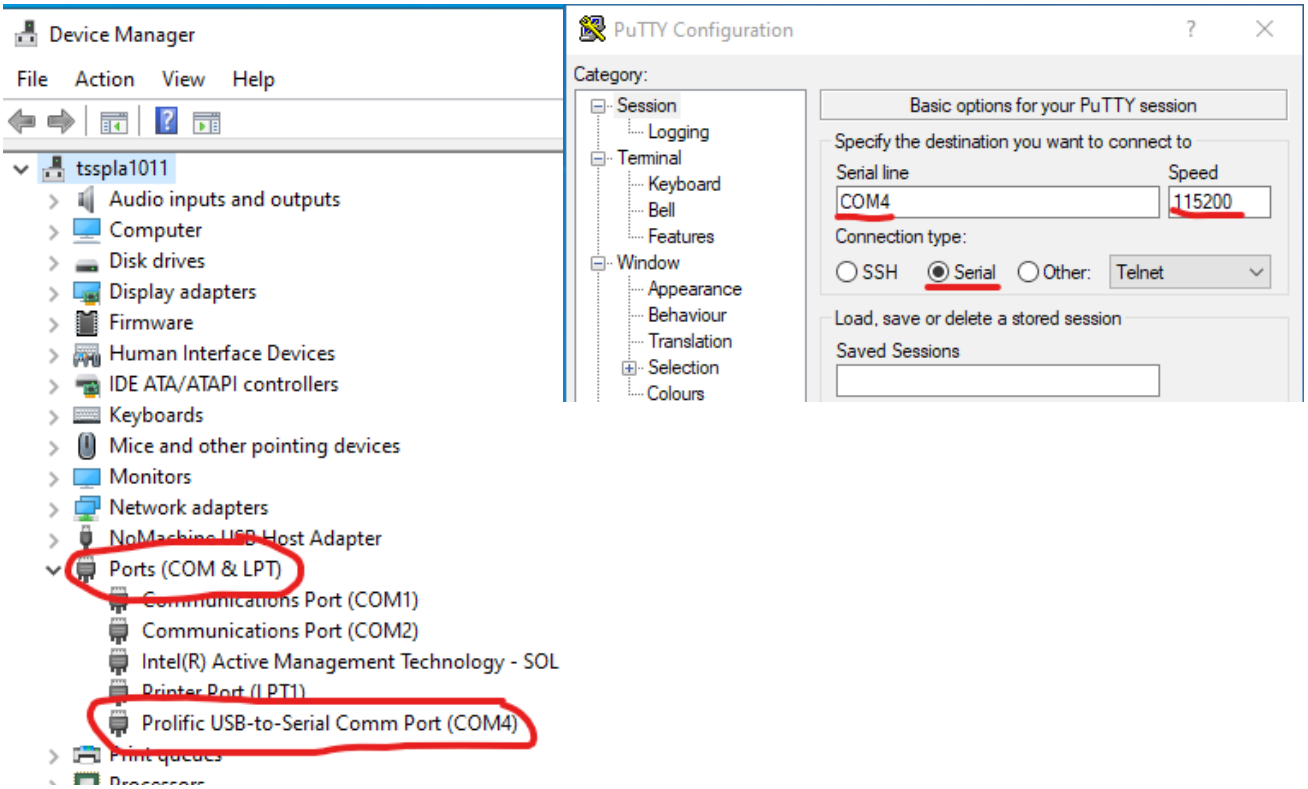
“raspberry” to log in. If you do not see anything in the serial terminal, you may have the TX and RX wires crossed. Check your wiring and change the connections, then press *enter* a few times in your serial terminal to see if it is working.

On a Windows PC, we need to install serial-port terminal software to read/write the serial port. Many programs are available for this, but a program called H-Term works well (available here: http://www.der-hammer.info/terminal/). You will need to find which COM port is the correct one by looking in Device Manager under COM ports, and set `hterm` to read newlines correctly by changing the 'Newline at' option to 'LF'. You may instead prefer to use PuTTY (https://www.putty.org/) which is newer, more fully featured, and works on both SSH and Serial connections. PuTTY is already installed on the lab workstations, so simply use this if you are in the lab.

On Linux/Mac systems no driver is needed: the USB to Serial converter will usually appear as `/dev/ttyUSBx` or `/dev/ttyACMx` – use `dmesg` to check the name of the last plugged in device. You can easily watch output from the serial port by setting the baud rate with `stty -F /dev/ttyUSB0 115200 cs8 -cstopb -parenb` and then concatenating the output with `cat /dev/ttyUSB0` This is less convenient than a terminal, so two easy to use Linux terminals are minicom and picocom which can both be installed on a Raspberry Pi or Ubuntu PC with `sudo apt install minicom picocom`. Picocom is started with `picocom -b 115200 /dev/ttyUSB0` and exiting is a little complicated – hold <ctrl> and type ‘a’ then ‘x’.

Serial connection on the lab computers

On the lab computers, PuTTY is installed on the Windows partition, and you can use this for both UART (aka ‘serial’) and SSH connections. To connect using your UART cable, identify the correct COM port using Control Panel->Device Manager, and initiate a serial connection in PuTTY using a baudrate of 115200 as shown.



Note:In this example, the COM port is COM4, but it may be different on your PC, so you must check which is the correct port on your machine for the USB-to-Serial adaptor.

When you first connect in this way, the terminal may be blank, even if it is properly connected. If this is the case, you should be able to just press ‘enter’ to reach the login prompt. The default login username is ‘pi’, and the password is ‘raspberry’. Once you are connected, it should look like this:

TASK 4: SET UP NETWORKING TO THE RASPBERRY PI

Home network instructions

You will need networking to a PC in order to do some of the later labs. The easiest way to connect to the Raspberry Pi is to plug it in to the modular jack of a home network with an Ethernet cable. You can also set the Raspberry Pi up to use Wifi by following the instructions at:

https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

To use a network connection, you need to first log in to the serial terminal and then enter `ifconfig` to get the network address assigned by your DHCP server or router. Next to "eth0" in the output, look for the text "inet xxx.xxx.xxx.xxx", where x's are numbers – this is your IPv4 address and usually starts with 192.168.0.xxx for a home network. Make note of this address and then on a terminal program such as PuTTY select a Secure Shell (SSH) connection and enter the host to connect to as "pi@xxx.xxx.xxx.xxx" using this IP address. If you are on a Linux command line use:

```
ssh -X pi@xxx.xxx.xxx.xxx
```

Then log in using the username "pi" and default password "Raspberry" as above. You may afterwards want to change your password with `passwd` for better login security. The "-X" switch turns on Xorg display forwarding so that programs you run on the Raspberry Pi can conveniently open a window over the network to your host PC.

Once you have networking running we also recommend you use the WinSCP program to move files to and from your Raspberry Pi.

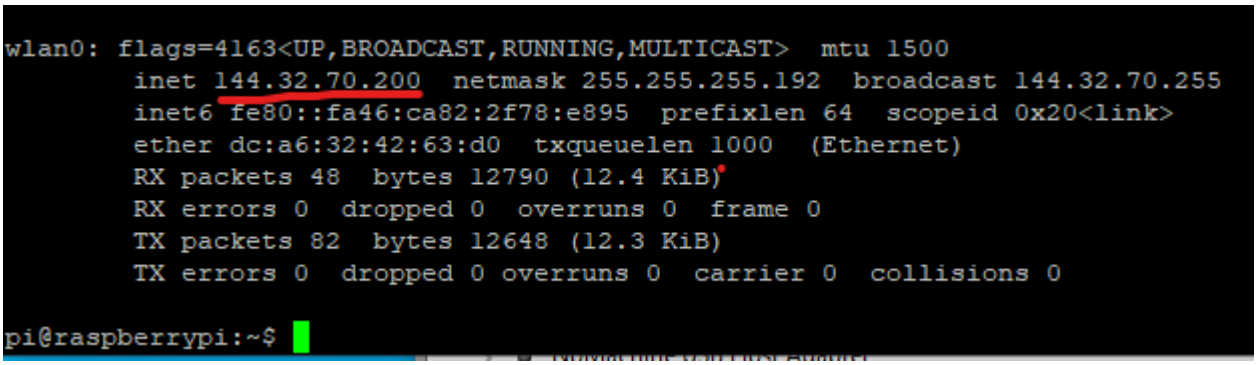
Connecting to the robot lab network

A special wi-fi node called 'robotlab' has been provided for use with robots in the lab. You can connect to this network as follows. After connecting to your robot with the serial terminal, type `sudo raspi-config` to open the configuration screen, go to "2: Network Options" and then "N2 Wireless LAN" to enter an SSID and passphrase.

The SSID is: **robotlab**

The passphrase is: **vertzlentath**

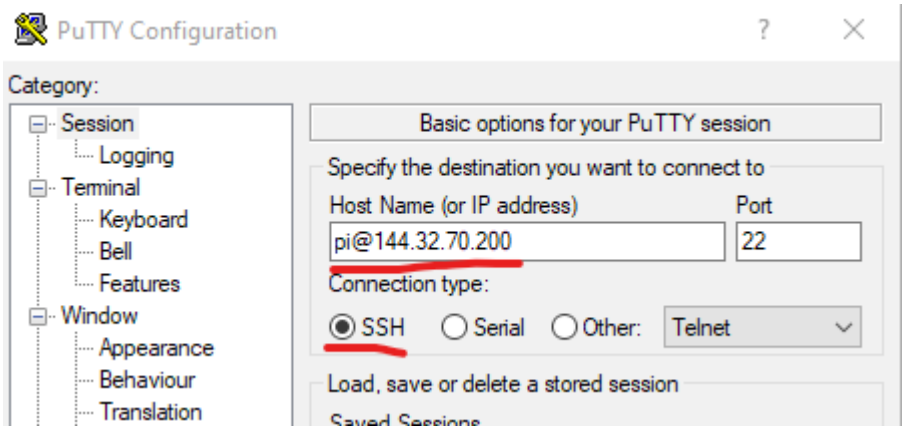
Once you have done this, go to ‘Finish’. After a few moments, the robot should connect to the robotlab network. You will need to check this and find out your IP address on that network. This can be achieved by typing `ifconfig` in the terminal. If you look for the section called `wlan0`, you should be able to see an IP address similar to this one.



You can use this address to connect to your robot over the wireless network. There are two important uses for this ability, SSH and file transfer.

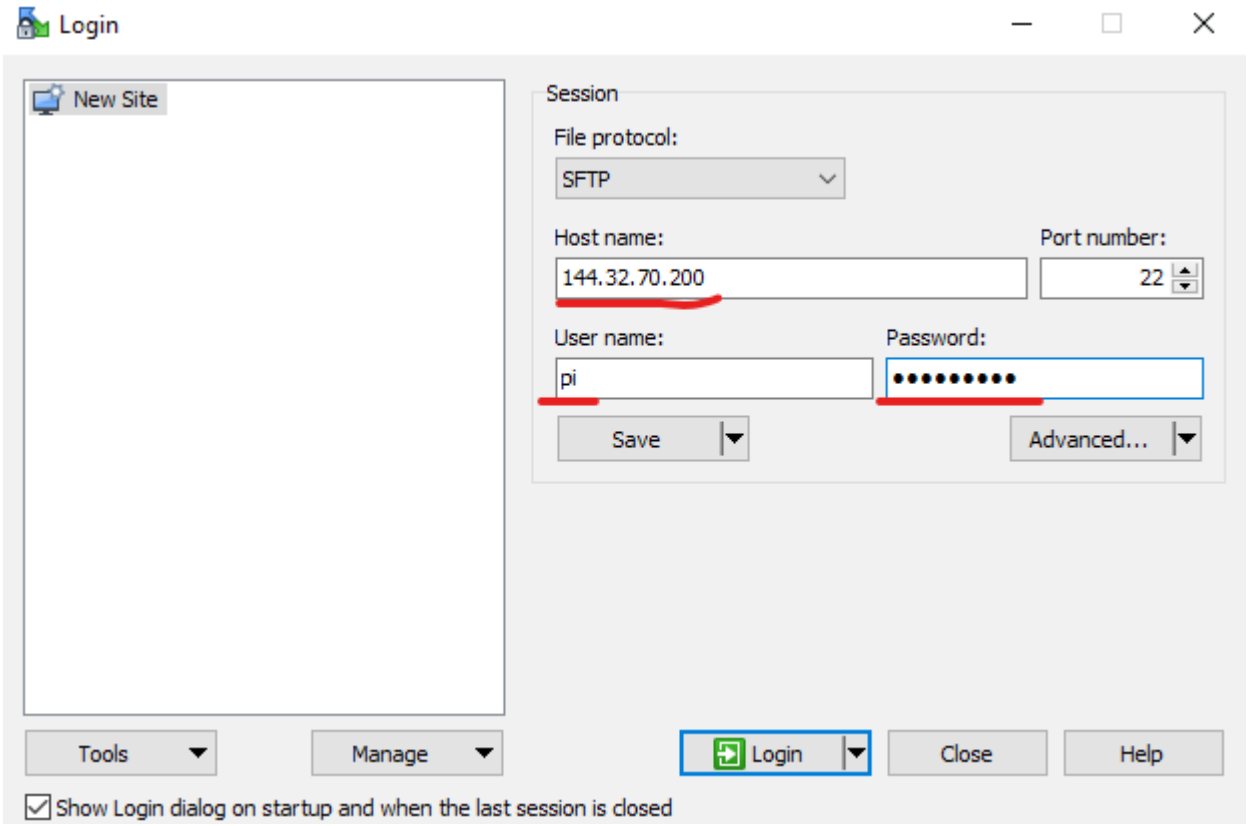
SSH using PuTTY

Using the IP address you have found, you can connect over SSH with PuTTY using pi@<your_ip_address> as shown. This will give you a terminal similar to the serial terminal, but wirelessly connected and supporting more features.

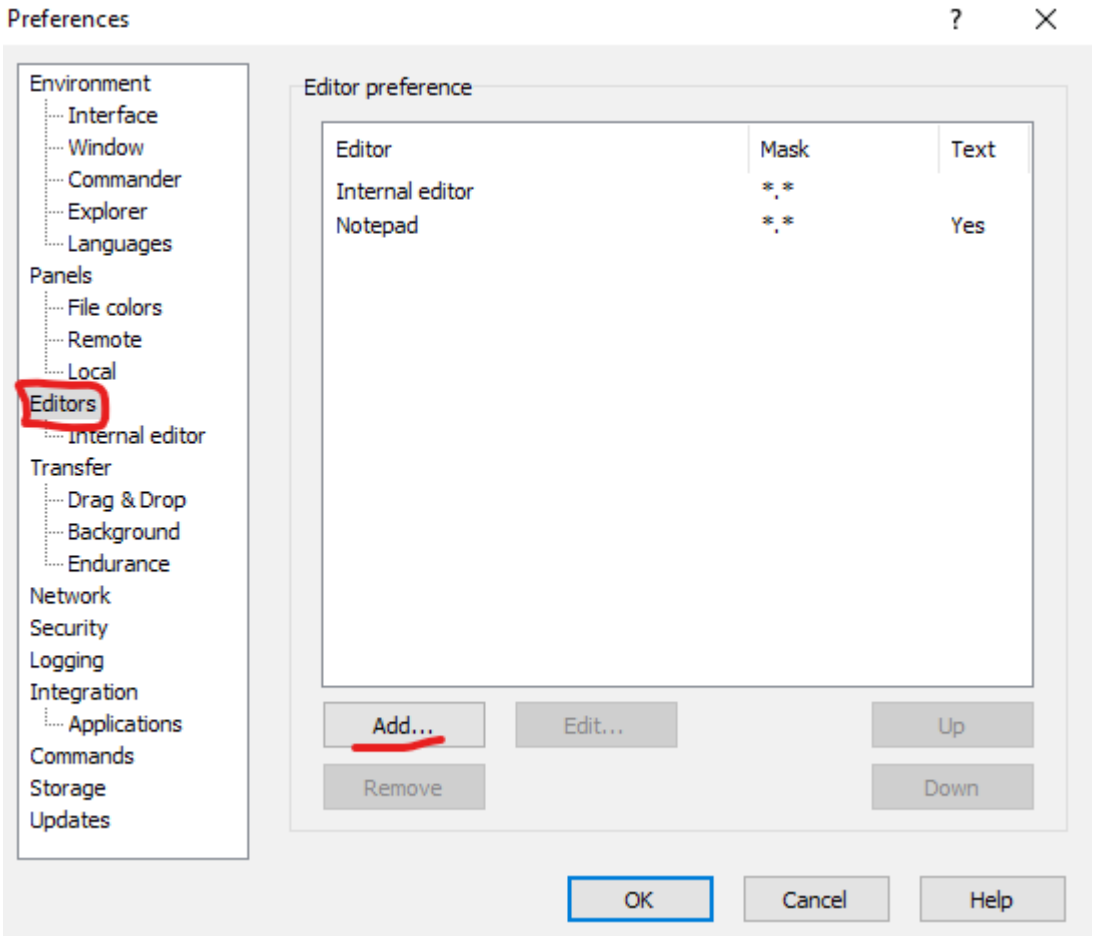


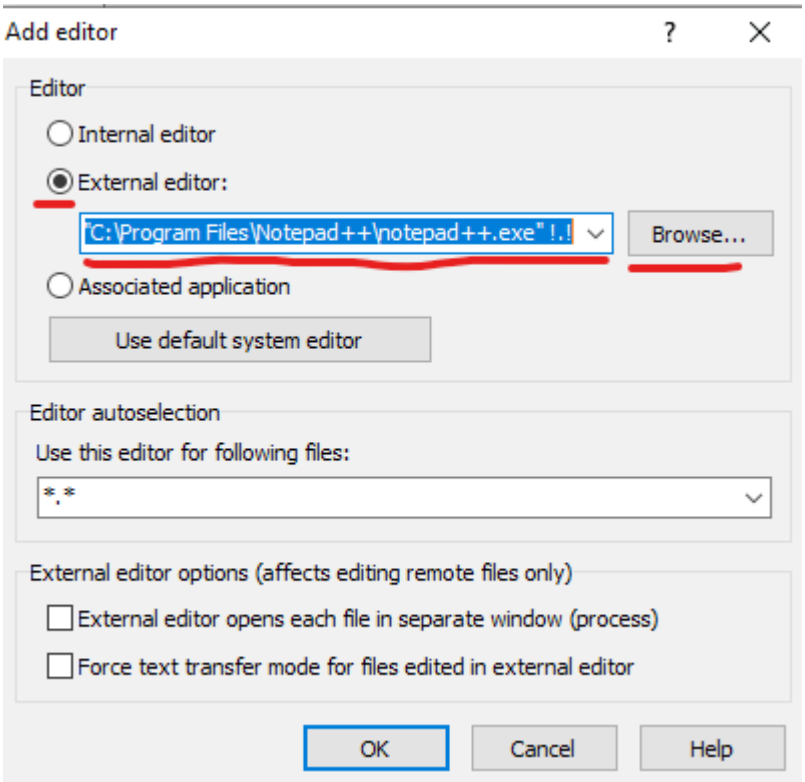
File Transfer/Editing using WinSCP

You can connect using WinSCP in a similar way, allowing you to browse the folder structure of your Pi and (very usefully) directly edit code there using the text editor of your choice. The configuration looks very similar to what we used for PuTTY, except that we can enter the username and password directly in the configuration screen:

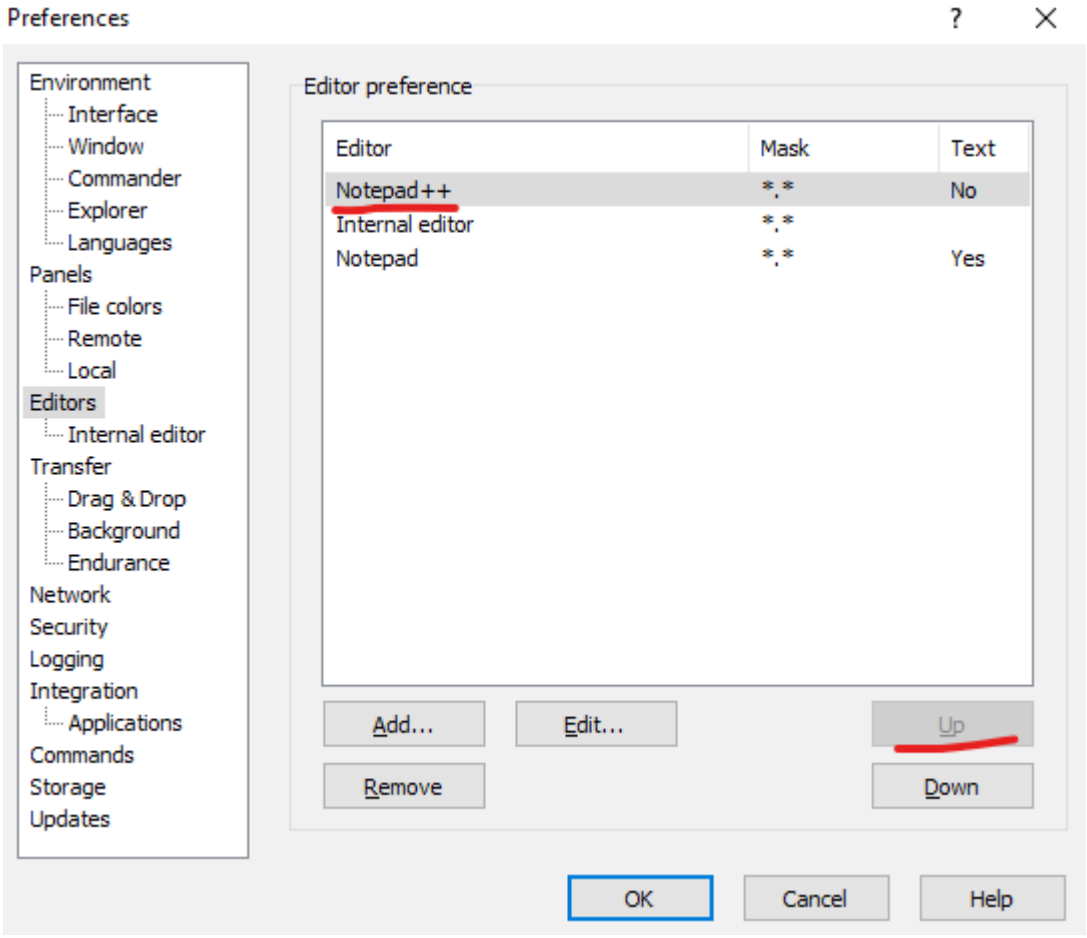


This will open a file explorer, showing your local computer on the left, and the Pi on the right. You can browse the files on the Pi, copy files between locations, and if you open a code file on the Pi, it will be opened local as a local copy in a text editor. You can make changes to the code, and when you save it, the modified file will be written back to the Pi. By default this will use Notepad – you may want to change this to a more appropriate editor for code, such as Notepad++, which is pre-installed on the lab PCs (you can also use a text editor of your own choice, if it has a portable version that you can link to from this dialogue – Sublime Text is one good example of this)


Preferences
Environment
Interface
Window
Commander
Explorer
Languages
Panels
File colors
Remote
Local
Editors
Internal editor
Transfer
Drag & Drop
Background
Endurance
Network
Security
Logging
Integration
Applications
Commands
Storage
Updates
Editor preference
Editor
Mask
Text
Internal editor
.
Notepad
.
Yes
Add...
Edit...
Up
Remove
Down
OK
Cancel
Help


Add editor
Editor
Internal editor
External editor:
"C:\Program Files\Notepad++\notepad++.exe" !.!
Browse...
Associated application
Use default system editor
Editor autoselection
Use this editor for following files:
.
External editor options (affects editing remote files only)
External editor opens each file in separate window (process)
Force text transfer mode for files edited in external editor
OK
Cancel
Help

To make your choice the preferred editor, you will need to move it up to the top of the editor preference using the ‘up’ button.



Now you can edit code files on your Pi simply by double-clicking them, and they will open in your chosen text editor. After saving your changes, you can run your code from the command line using your SSH connection, thereby allowing completely wireless software development on your robot!

TASK 5: WRITE A C PROGRAM TO BLINK THE LED USING PIGPIO

You can edit the C and Python files for these labs using a command-line text editor called `nano`. If you’re familiar with another command-line editor that happens to be installed on the Pi (such as `vi`) then you may use this instead. It’s also possible to edit over the network using a local text editor on your workstation as described earlier, which is particularly useful when working with larger projects, but for now we recommend not to use a graphical text editor or a programming IDE for editing. Part of the learning experience is editing files using only the command line, because this replicates what you will sometimes have to use if you use a Raspberry Pi mounted on a robot.

There are several libraries and frameworks that can be used on the Raspberry Pi for controlling GPIO pins and their functions. In these labs we will use the most recent and full-featured of these – the *pigpio* library. The *pigpio* library has documentation at http://abyz.me.uk/rpi/pigpio/ and has bindings both for C/C++ and for Python. To practice using a low-level interface to GPIO pins similar to that used by Arduino and Mbed frameworks, we first illustrate the use of C programming to blink the LED.

Find the *pigpio_example.c* file in the labs directory (from the command prompt use `cd labs`, and return to the home directory with just `cd`. Use the editor *nano* to view the file with `nano pigpio_example.c`. (you are welcome to install and use any other editor you are familiar with as well) In the *nano* editor the commands are *ctrl-?* where *?* is one of the keys listed at the bottom of the screen. The most useful commands are:

- cut line(s): `ctrl-k`
- paste (un-cut) the last cut line(s): `ctrl-u`
- save the file: `ctrl-s`
- quit the editor: `ctrl-x`

We can look at each line of the code in this program to understand how it works.

```
#include <stdio.h>
#include <pigpio.h>
#include <math.h>
```

#include statements tell the compiler that we wish to include different code files before compiling this program. Here we are including stdio.h and pigpio.h – 'header' files, and math.h for later use of math functions. These include definitions of functions and other header files we need to make the GPIO pins on the Pi work. We can also use #define for convenience to map text strings to other strings/numbers/etc. that we want to use in our program. In this case we tell the compiler to replace the term "LED" with the pin number 5 with:

```
#define LED 5
```

We then start the program with:

```
int main() {
```

A block of code in C++ is called a function – and every program must have at least one function which is called main(). This function is needed as it is the code that is run when

the program starts. Blocks of code are contained within pairs of { } curly braces; it is conventional to indent code within pairs to make nested blocks easier to ready, as can be seen in the code example.

```
if (gpioInitialise() < 0)
```

This is a conditional *if* statement that serves two purposes: it initializes the pigpio library and it is checked to make sure initialization is successful. If it is not (the return value is <0) then the code within the curly braces is run, in this case a `printf` is used to print an error text string and then the program is exited with `return 1;`.

```
gpioSetMode(LED, PI_OUTPUT);
```

This statement sets the pin we defined as "LED" (pin 5) as an output pin, meaning the voltage on it will be changed from 0 to 3.3V when commanded.

```
for(i=0; i<10; i++)
```

Here we start a 'for' loop. Code within the curly braces of a *for* loop is run until the condition in the middle part of the statement *i<10* is false. Before the first loop is run, *i=0* is set. Then after each loop, *i++* is run which increments the value of *i*.

You could also use a *while* statement like:

```
while(1)
```

Code within the curly braces of a *while* loop as long as the statement within the round braces is true. Here the statement is simply "1" which is the same as 'true' in binary – meaning that the code will be always run. It is the simplest way of creating an endless loop in C++. However you will have to exit the program manually with *Ctrl-C*.

```
gpioWrite(LED, 1);
```

We defined LED as 5, and pin 5 to be a *PI_OUTPUT* above. When we set a pin value with *gpioWrite()* to 1, it means we turn on the output pin – which in this case will turn on the LED in your circuit. Setting the value to 0 turns off the output pin and LED.

```
time_sleep(0.5);
```

The time_sleep statement simply makes the program pause for the amount of time (in seconds) expressed by the number in the braces. The next two lines are very similar to these, except we are turning off the LED and waiting 0.5 seconds again.

Compile the program with:

```
gcc -o pigpio_example pigpio_example.c -lpigpio -lm
```

NOTE: if you are using geany or another IDE that uses a pre-defined command to compile the source code you will need to add *-lpigpio* (link to pigpio library) and *-lm* (link to math library) to the linker stage of the build commands to compile your programs, as well as the *#include <pigpio.h>* and *#include <math.h>* program code.

Then run it with:

```
sudo ./pigpio_example
```

You should see the LED blink 10 times and then stop. Note that we need to run C programs using the pigpio library with *sudo* because the library writes directly to the memory space of the Raspberry Pi, which is protected for security. *sudo* executes a command as if you are the root (system) user, which has access to the memory space.

When you have the program running, try editing the code to make it do the following:

1. Get the Pi to blink the LEDs at a faster or slower rate, and a different number of times, then recompile your code and test it
2. Adapt your code so that it runs continuously until *Ctrl-C* is pressed

TASK 6: BLINK THE LED USING PYTHON AND THE PIGPIOD DAEMON

It is advantageous if we can use a high-level language like Python to control GPIO pins as well, and also without having to use *sudo* for security reasons. This task will accomplish the same thing as the previous one, but using Python and communicating with the *pigpiod* daemon (a program the runs in the background and provides services). To use *pigpiod* you need to first start it as the root user with:

```
sudo pigpiod
```

You can find the complete list of Python functions that you can use with examples at http://abyz.co.uk/rpi/pigpio/python.html. Find the *pigpio_example.py* script in the *labs* directory. We will go through it here as before:

```
#!/usr/bin/python3
```

This line starts with a *shebang* or *crunchbang* (#!) tells the shell (bash) that this script can be executed with the python3 executive. There is no compilation since Python can be a run-time interpreted language (you can also compile files to byte code though).

```
import time
import pigpio
```

There are also no *#include* statements. Instead, libraries are imported as above. Here we are importing the *time* library for the *sleep()*function and the *pigpio* library to control the GPIO pins.

```
LED = 5
```

Instead of *#define* we simply create a variable “LED” with the value of the pin number 5

```
  pi = pigpio.pi()
```

Python is an object-oriented language, so this creates a *pi* object from the *pigpio* library that represents all the GPIO pins on your device that can be accessed.

```
  pi.set_mode(LED, pigpio.OUTPUT)
```

This statement sets the pin we defined as “LED” (pin 5) as an output pin, meaning the voltage on it will be changed from 0 to 3.3V when commanded.

```
  for i in range(0,10):
```

Instead of for loops, Python uses *iterators*. This statement sets *i* to each of the values between 0 and 10 (ending at 9). Instead of curly braces, Python uses **indentation** to denote what statements are part of a loop. Unlike C, you **must** indent each statement within a loop exactly the same way (you can use any number of spaces, tabs, etc. as long as they are consistent)

The equivalent *while* statement in Python to let the program run endlessly would be:

```
  while True:
```

However you will have to exit the program manually with *Ctrl-C*.

```
  pi.write(LED, 1)
```

We defined LED as 5, and pin 5 to be an output above. Setting a pin with *pi.write()* to 1, means we turn on the output pin. Setting the value to 0 turns off the output pin.

```
  time.sleep(0.5);
```

The time.sleep from the time library simply makes the program pause for the amount of time (in seconds) expressed by the number in the braces. The next two lines are very similar to these, except we are turning off the LED and waiting 0.5 seconds again.

Run the program with:

```
  python3 pigpio_example.py
```

Or alternately (because of the *#!/usr/bin/python3* line and that the file has executable permissions set) you can execute it like a program with:

```
./pigpio_example.py
```

You should see the LED blink 10 times and then stop. Note that *sudo* is no longer needed because the library uses the *pigpio* daemon as an intermediary to write to the memory.

When you have the program running, try editing the code to make it do the following:

3. Get the Pi to blink the LEDs at a faster or slower rate, and a different number of times, then recompile your code and test it
4. Adapt your code so that it runs continuously until *Ctrl-C* is pressed

TASK 7: CONTROL FROM THE PC SERIAL TERMINAL CONSOLE

A vital part of robotic communications is being able to send commands in real-time to a robot as well as receiving information from it. Characters can be read and sent from the Raspberry Pi console easily using C and Python functions while you are logged in to the serial port or to a SSH console over a network.

If you are writing code in the C language:

Printing a line of text to the console in C is simple: *#include <stdio.h>* and use the *printf()* function. The function prototype is :

```
int printf(const char *format, ...)
```

The "format" string is a combination of text and formatting characters prefixed with '%' that print the values of variables that are given in the space where the "..." are. For example, to print an character, integer, and floating point value you could write:

```
printf("mychar = %c myint = %d myfloat = %f\r\n", mychar, myint, myfloat);
```

if you have declared variables *char mychar; int myint; float myfloat;* See https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm for details.

The *"\r\n"* at the end represent control characters for a "carriage return" that returns the cursor the the left side of the console, and a "newline" that scrolls the console down by one line. The way these are interpreted depends on the terminal emulation software that your system is using. UNIX/Linux uses '\n' as both a return and newline, Mac OS X uses '\r' as both a return and newline, and DOS/Windows separates '\r' and '\n'.

Reading a single line of text in a C program is also, easy, as you can use the *scanf()* function that has similar syntax, with a format string that specifies what format that text should be converted into. Two examples would be:

```
scanf("%c", &mychar);
scanf("%d", &myint);
```

See https://www.tutorialspoint.com/c_standard_library/c_function_scanf.htm for details.

If you only want to read in one character (useful for controlling your robot with keys) you can also use the getchar() function as follows:

```
mychar = getchar();
```

See also: https://www.tutorialspoint.com/c_standard_library/c_function_getchar.htm

If you are writing code in the Python language:

Python uses the *print()* function, which is very flexible in use. You can print a string (delimited with either single or double quotes), and combine it with other strings using the '+' operator to concatenate them. Note that you have to enclose non-character variables with the *str()* function to convert them to a printable string:

```
print("mychar = "+str(mychar)+" myint = "+str(myint)+"
myfloat = "+str(myfloat)+"\r\n")
```

For more control over formatting, you can use the *%* and *.format* formatters in Python in a similar manner to *printf()* in C. Detailed examples are given at https://pyformat.info and an example of formatting the same string as above is given here:

```
print("mychar = %c myint = %d myfloat = %f\r\n" %(mychar, myint, myfloat))
```

Input from the console can be read using the *input()* function in Python, which reads in lines of text as strings after printing a optional prompt string which can be omitted by just calling `x = input()` alone. If you want to read in non-character numbers, you will need to convert them to numbers with the *int()* and *float()* functions, for example:

```
mychar = input("enter a character: ")
myint = int(input("enter an integer: "))
myfloat = float(input("enter a floating-point number: "))
```

EXTRA TASK: NON-BLOCKING KEYBOARD INPUT

The disadvantage of the console functions above is that they require you to push the *Enter* key before any text is read, after which **all** the text you have typed will be read in at once (a limitation of the way consoles are designed). Non-blocking keyboard input and game-like keyboard input that does not wait for *Enter* is a bit more complex and not covered here. However, you are strongly encouraged to find a solution online on your own as it will be good practice for solving software problems later on in the course.

EXTRA TASK: ADD AN OPTICAL SWITCH TO THE RASPBERRY PI

Add an external switch to the board to make it toggle the LED on and off. This can be used also as a switch to help control the robot while it is moving (e.g. an emergency stop). You can use this switch as you wish to extend later exercises.

The RasPiO Pro Hat includes a cadmium sulfide (CdS) cell Light Dependent Resistor (LDR) that can be used as a light-sensitive switch. For instructions on use see page 16 of the manual at https://rasp.io/wp-content/uploads/2016/04/Pro-Hat-GPIO-Zero.pdf. Add this sensor to your circuit as a light-sensitive switch and have it turn the LED on and off.

NOTE: Whilst adding a switch is relatively straightforward, most mechanical switches suffer from 'bounce'. What are the effects of this – and how can they be overcome? Dr. Andy Greensted – who used to work at York – has a really good page on switch debouncing that is well worth a read
http://www.labbookpages.co.uk/electronics/debounce.html