Worksheet 2

MATLAB and the Raspberry PI

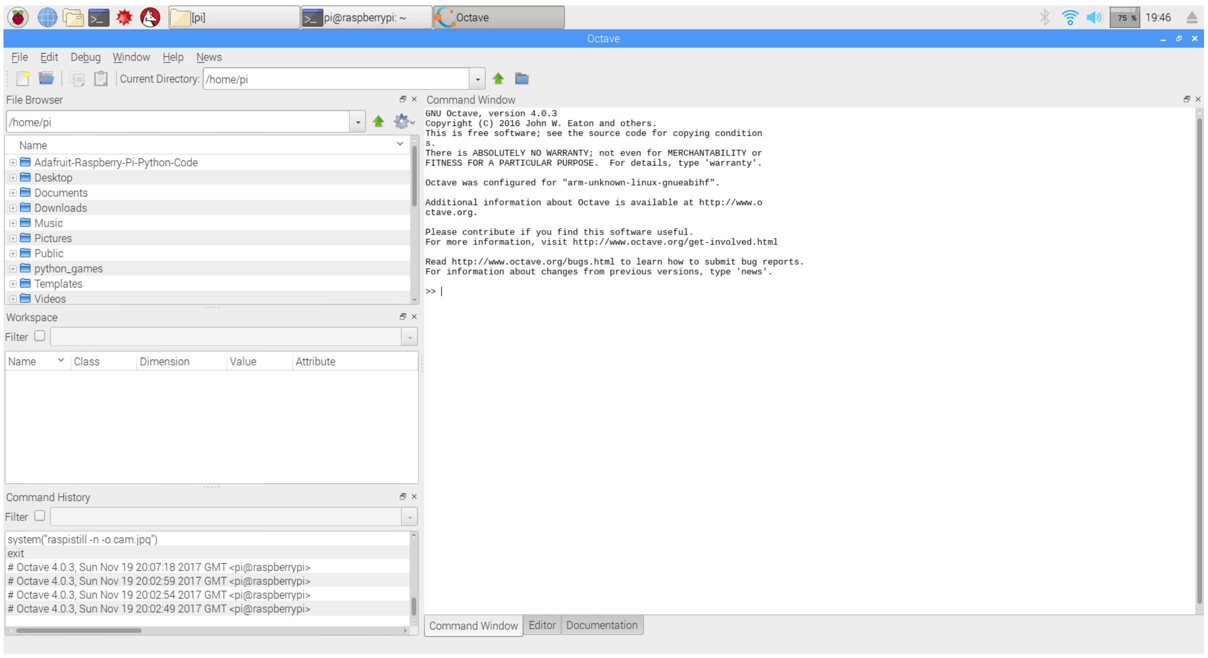
Until now you have only programmed in the MATLAB programming language, although MATLAB it's self is quite useful, it has some drawback:

1. It's really really expensive, and to make sure the developers get their money for it, every time MATLAB starts, it checks a license server to see if you are allowed to use it (and if you have paid your money!). This is non ideal for a non internet connected buggy.

2. It's really memory/disk intensive, and although the PI is quite powerful, it would struggle with MATLAB. Even my i7-Intel laptop sometimes struggles with MATLAB.

Luckily for you, there is an alternative to MATLAB which is free and does not have any of the above drawbacks. It is called Octave, it's is much less memory intensive and will run easily on our Raspberry PIs just fine. It works just like just like MATLAB, and you will feel quite at home using it.

To start Octave, click on the Raspberry, in the top left hand corner of the screen, go to Education and click on Octave, you should get something looking like Figure 1. Think as Octave as free/open source MATLAB.



```
GNU Octave, version 4.0.3
Copyright (C) 2016 John W. Eaton and others.
This is free software; see the source code for copying condition
s.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "arm-unknown-linux-gnueabihf".

Additional information about Octave is available at http://www.o
ctave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>> |
```

```
system("raspistill -n -o cam.jpg")
exit
# Octave 4.0.3, Sun Nov 19 20:07:18 2017 GMT <pi@raspberrypi>
# Octave 4.0.3, Sun Nov 19 20:02:59 2017 GMT <pi@raspberrypi>
# Octave 4.0.3, Sun Nov 19 20:02:54 2017 GMT <pi@raspberrypi>
# Octave 4.0.3, Sun Nov 19 20:02:49 2017 GMT <pi@raspberrypi>
```

Figure 1: Octave running on the PI. Personally, I really like Octave and prefer using it to MATLAB.

Basic use of Octave

Just to make you feel at home with Octave, we will now use it to write some basic computer code, just as we did when we first started learning MATLAB.

a) Generate a 10x10 random array of numbers.

b) Define an array of an array called student_marks equal to [10 20 30 40 50 60 70 80 90]

c) Set variable 'b' equal to the 2nd element of student_marks, and the variable ‘a’ equal to the third element of the array student_marks.

d) Calculate the average of the variables a and b.

e) Replace the 5th element of the array with 80.

See it’s just like MATLAB. :)

Assessment: Before we continue, I should say that today's lab will be assessed as a group exercise it will be worth 10% of the buggy lab in total. I expect a collection of .m files to be submitted by each of you to moodle by the end of the lab. Really, I am doing this just to make sure your group has gone through the lab sheet and understood it – i.e. it’s to help you learn. The marking scheme will be as follows:

*No file submitted: 0%
*Poor effort, an attempt at less than half of it: 40%
*Reasonable effort, some parts missing: 60%
*Good effort, with everything done: 100%

If I ask you to save a file, then it will form part of the assessment, and I expect it handed in at the end of the lab.

I expect most groups to get 100% :)

Scripts in Octave

Q1. Just like MATLAB, you can make scripts in Octave. To make a new script, click file→New→New script. You will see a script appear on the right hand side of the screen. A new menu has appeared looking like figure 2.

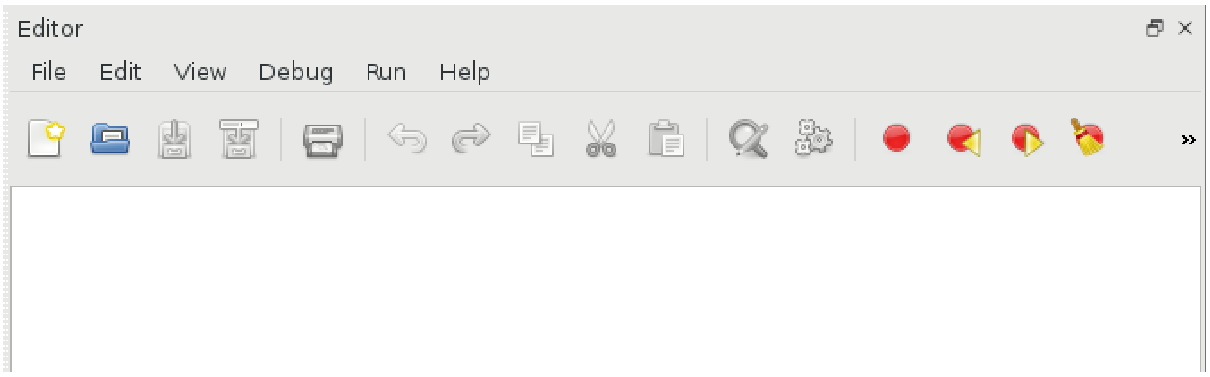
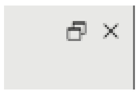


Figure 2: The script editor

Click, *File→Save as* and then save the file as ‘q1.m’ under /home/pi/ . On the PI, all users home directories are stored under /home/, and you are logged into the system under user ‘pi’, so you save all your work in /home/pi/ Now in the script use the disp command to print ‘I love programming the PI!’ to the screen. Then once you have done this, click on the cog icon, with the little yellow play button embedded in it. Your script will then run. The output will appear in the command window. If you can’t see the command window click on the, little window icon to break the script editor out of the main interface:



Q2) Make a new script called q2.m. Now edit your script so that it will sum the numbers from 1 to 100, using a for loop.

Q3) Make a new script q3.m. Using a while loop make it count from -10.0 to 0.0.

Q4) Write a script which sorts an array of 10 random numbers. Hint: The answer’s in the lecture notes. :), save it in q4.m

Q5) Write a script to integrate, the function sin(x)+0.1*sin(x) between -pi and pi. and save it in q5.m.

The above questions, are really just to get you back into thinking about programming again.

<u>The echo sensor</u>

We are now going to start using the PI to control hardware. Let’s first start off with the echo sensor as we already have it installed. Make a new script called echo_test.m, and save this under /home/pi/. To control the hardware of the buggy, we are going to have to use functions, which know how to talk to the hardware. These functions are stored in /home/pi/lib. Add the line

addpath('/home/pi/lib')

to the top of your script, this will tell Octave where the functions are stored.

Now under that add the line:

echo_sensor()

and run the script.

Try placing your hand in front of the echo sensor and then moving it away, how does the number change? If the number does not change, or you get a minus number check the wiring on your echo sensor very carefully. If all else fails call a demonstrator over for help.

Q1) Using a while loop and the sleep command, make Octave print out a distance measurement every second. Something funny happens, when you run this script. It will only show you the output once the script ends. Octave does this, to make the program quicker to run [MATLAB does not do this], printing text to the screen is very slow. However, sometimes it is useful to see the text generated by our program as it runs. To force Octave to print out text, as the program runs add the command

fflush(stdout)

just after the command used to print the distance.

Q2) Edit your program so that it prints out the words "Too close!", when you hold less than 5 cm front of the sensor, "I miss you!", when your hand is not there, and "Just right", when your hand is 20-25 cm away. Save this all under echo_test.m

Motor control

The buggy has two motors. These can be used to drive the buggy forwards, backwards or to turn it, by running one motor forwards and one backwards. By the end of this section you will be able to move the buggy in any direction.

The Octave/MATLAB command to drive the motors is

motors(power1,power2,delay)

Power1 and Power2 control the power going to the motors, the power must be a number from 0 to 100. The delay is the time for which the motors run. So for example if you wanted to move the buggy forwards for one second, you would use the command

motors(100,100,1)

If you want to run a motor in reverse simply put a minus in front of the motor power. For example,

motors(-100,-100,1)

To stop the motors, use the command:

motors(0,0,1)

Q1) Now, make a new script called my_motor_script.m and save it under /home/pi/. Make sure you add ***addpath('/home/pi/lib')*** to the top of the script so that, Octave can find the buggy functions. [I'm not going to tell you to do this again, I'm going to assume you know this has to be done by you automaticly.] Now try out the commands above to move the motors. Did you wire them up correctly? If they spin in different directions, just swap the wires around in the motor driver board.

Hint 1: You will need to use the **addpath('/home/pi/lib/')** command before these commands will work though.

Hint 2: Position your buggy so the wheels are off the ground, you can use the box your PI came in to do this. If you don't do this your buggy will fly off the table.

Q2) We are going to write a script to drive the buggy forward then to stop for 5 seconds when it sees an object. Edit the script my_motor_script.m, so that it contains a while loop, which will run for ever (hint: while(1) ….. end). In the while loop, just as you did before print out the values from the echo sensor. Now using an ***if*** statement, run the motors for 1 second if the distance detected is more than 45cm, if a distance of less than 45 cm, is detected make the code wait for 5 second using the **sleep** command.

Q3) We are now going to edit the script to reverse the direction of the buggy if, an object is detected. Define a variable outside the while loop called 'direction' set it to 1.0. Now, if a distance below 45 cm is detected make the script multiply direction by -1.0 and wait for one second. Now, edit the script by using an 'if-else' statement, to drive the buggy forwards if direction is set to 1.0 and backwards if direction is set to -1.0.

Q4) Now, assuming your script works, re-save you script as /home/pi/autorun.m . Unplug all the cables from the buggy, and put the buggy on the floor. Turn the buggy on with the power switch, and it should run of batteries. It will take about 30 seconds to boot, but when it does it will run the file autorun.m and your buggy should start going forwards!

- Note 1: autorun.m will only be executed, when no keyboard or mouse are present. If you want to test autorun.m, while the screen is still connected to you PC, just unplug the keyboard and mouse, and it will be executed.
- Note 2: Do remember to turn off the buggy before connecting it to the PC again. If you don't there is a risk power will be fed from the batteries to the USB port of the PC, which won't have a good outcome. [https://www.youtube.com/watch?v=2SopsQEfoc4]

Q5) By setting one motor to a much lower power level than the other, it is possible to make the buggy turn. Edit your script, so that when the buggy detects an object, it will:

- Reverse for 2 seconds
- Start going forwards again, but it run one motor slower than the other for a 3 seconds to make it turn.
- Go forwards on full power with both motors for 3 seconds.
- Then run the other motor slow for 3 seconds to straighten up the buggy.

This sect of actions should enable your buggy to move around one of your team mates stood in the way. Pay with the code until it works, you may have to adjust the timings, and use a bit of creativity. Also it may or not work so well on carpet, feel free to go and find a surface which is not carpet – like the ESLC.

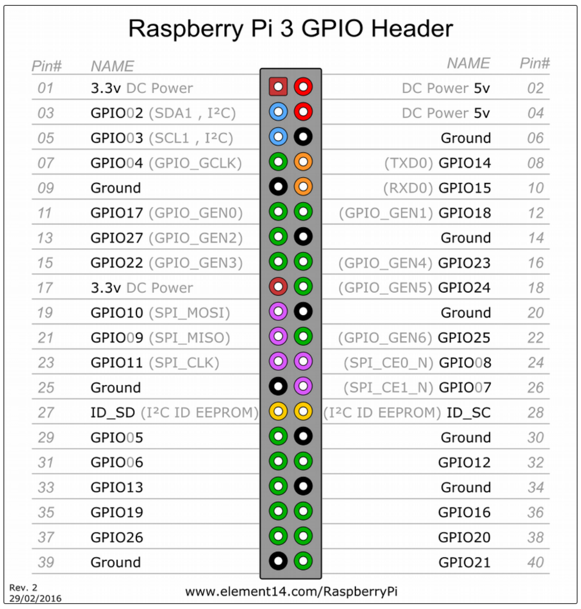
Using the GPIO pins

Now you have a grasp of the Octave software, you can start with some basic input and output commands. In this section you will use some pre-defined function in order to turn on LEDs and to check the state of a switch.

The Raspberry Pi has a set of GPIO pins that can act as inputs or outputs. The labels for the GPIO pins can be seen in figure 3. If you hold your PI, with the USB ports towards you and the SD card slot pointing away from you, the GPIO pins will be orientated the same as they are in the picture.

Output

Question 1: Which GPIO pins are the motors connected to? Save your answer as a comment in the file *gpio_question.m*.



Raspberry Pi 3 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016
www.element14.com/RaspberryPi

Figure 3: Raspberry Pi GPIO pin labels

Question 2: We are now going to have a play with driving some LEDs with the GPIO pins. Once you have mastered driving LEDs, you will be able to drive any external real world device using these pins. Think, car breaks, fans, lighting, ignition systems to rockets, anything which takes a yes/no signal to do something. So, although driving LEDs may seem pretty simple, it gives you the power to interface your computer with any real world device. In order to turn the LEDs off and on they must be connected to a output pins of the PI, using a 330 ohm resistor [brown, black,black, orange,orange]. The resistor just limits the current the LED draws from the PI. The wiring diagram for the LED can be seen in figure 4, the yellow blobs are the 330ohm resistors, they are connected to the negative power rail. The red wires go to the PI, use male female jumper leads to do this.

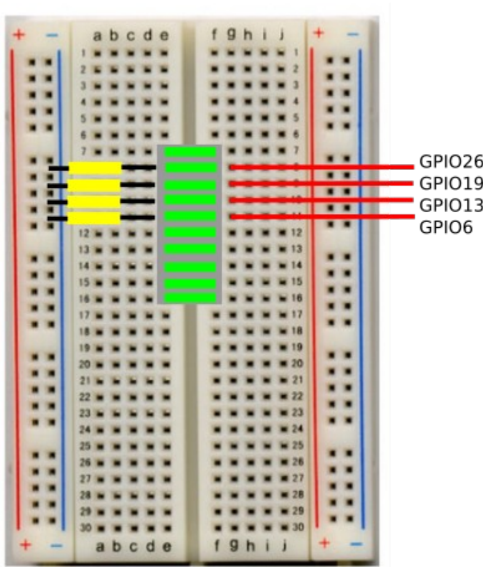


Figure 4: Wiring diagram for the LED [You may have to slightly rearrange the wiring for the echo sensor to get the LEDs into your bread board, don't rearrange the wiring for the power supply.]

Now the LEDs have been connected, the all the pins can be turned on for one second using the octave function

pin_out("1111",1.0)

Make a new script called led_test.m and see if the command works. If it does not work, you have probably connected your LED block the wrong way around, just lift it off the board rotate it through 180 degrees and plug it back in. [LEDs only work one way around, I did not tell you this before, because there was a 50% chance of you plugging it in the right way :)]. Now add the command pin_out("1010",1.0), to your script. What does it do? Save you script.

Question 3: Write a script to make your LEDs turn on and off randomly. With a one second wait between each random selection of LEDs. Hint, first pick a random number between 1 and 4, then use an if-elseif-end statement, to turn the on a given pattern of LEDs depending on which random number was chosen. Save this in the script led_test.m .

Question 4: Make a new script called knight_rider.m and make the active LED bounce backwards and forwards along the display, as shown in this video :):

https://www.youtube.com/watch?v=hG44lIO_bss

This can be done with a while loop, the pin_out command and the wait command. [If you've not seen the TV program…. you've missed nothing. :)]

Question 5: Write a program to turn on the LEDs when the distance detected from the echo sensor is less than 10cm, and to turn them off when the the distance is larger than 10cm. Save this as echo_led.m

Input from the outside world

Ask a demonstrator (or me :)) for a 'block of red switches', I have not put these in the kits as the pins get damaged very easily. In this final section to the work sheet, we are going to be using the switches to get input from the outside world. Again, just like with the LEDs, these switches could be replaced with any type of sensors, gas sensors, light sensors, heat sensors, you name it you can connect it to the PI. However, to keep things simple we are first going to play with simple off on switches first.

The bread board is getting pretty crowed now, but if you squeeze you resisters for your echo sensor right to the end, and then push the LEDs just up against them, there is enough room for the red block of switches. This can be seen in figure 5.

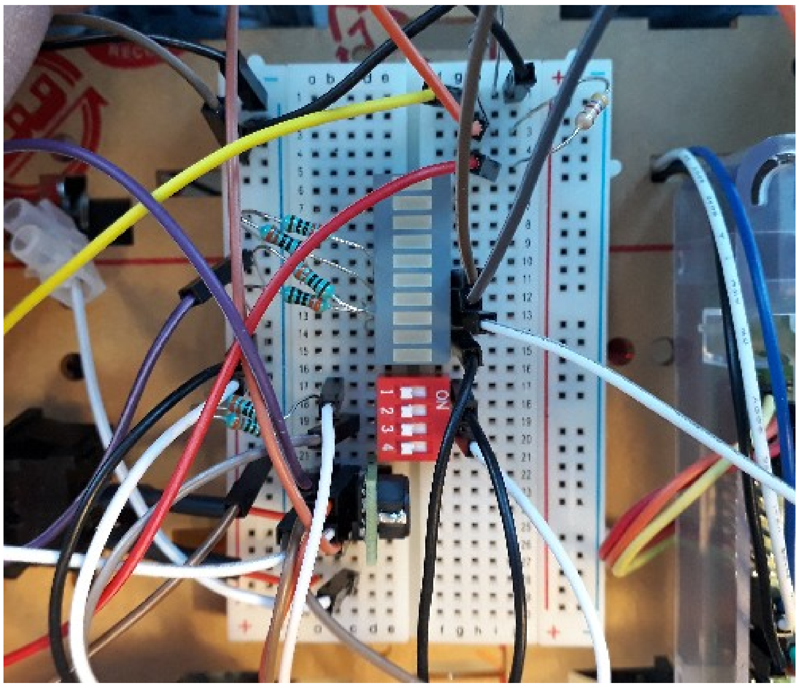


Figure 5: Adding the switches to the bread board.

Wire up the switches according to the wiring diagram in figure 6, if you get lost ask a demonstrator for help.

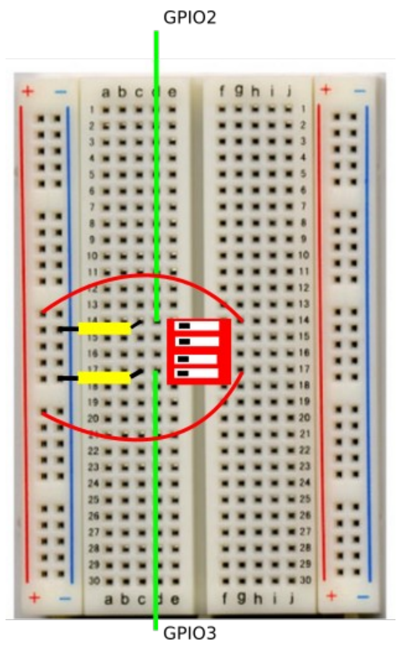


Figure 6: Adding the switches to the bread board. The yellow boxes are resistors, they are the same as the 330 Ohm ones you used for the LEDs.

Once the switches are installed you can read their state using the pin_in() function in octave, this returns a 1D array containing the position of the switches. Try playing with the switches and see what pin_in() returns. The switches don't stick to the bread board very well because they have short legs. To turn the switches on and off I suggest you hold the red switch block down with one fingure, and toggle the switches with the a screwdriver.

Question 1: We are now going to write a script to make the buggy drive in a big circle. If switch 1 is on, then your buggy will drive in a clockwise circle, if switch 1 is off it will drive anti clockwise circle. When the buggy meets an object, it will stop for one second, then display the following sequences on the LEDs 1111,1110,1101,1100,1011,1010, 1001,1000,0111,0110,0101,0100,0011,0010,0001, 0000, with a 0.1 second wait between each combination of numbers. These numbers are actually binary numbers counting from 15 to 0, (we will learn about binary numbers in second year). Once the buggy has counted from 1111 to 0000, it should resume driving in a circle. The task has been broken down into a series of steps below.

- **Step 1:** Make a new script called circle.m and save it. Then, use the pin_in() function, to determine if switch 1 is on or off.

- **Step 2:** Using an if statement, if switch 1 is on drive the left motor faster than the right motor for 1 second. If switch 1 is off, drive the right motor faster than the left motor for 1 second. Test your script.
- **Step 3:** After the commands for running the motors, use the echo sensor to test for an object closer than 20 cm. If an object is detected, make the buggy code wait for 5 seconds.
- **Step 4:** Use a while loop to (hint while(1)…...end) to make your loop run forever. Now, using a series of pin_out("xxxx",0.1) statements, and sleep commands make the buggy display the series of binary numbers given in the above question, if an object is detected. As soon as the number 0000 is displayed on the LEDs, the buggy should resume driving in a circle.
- **Step 5:** Once your code is working take your buggy down to the Atrium of the ESLC, and test it in the big open space.

Once you have completed all the tasks, use a usb stick to extract the .m files from the buggy, copy them onto your PC, zip them, then upload them to moodle. Each group member must do this individually although it's a group exercise. Next week's session will not be so group orientated.