[ELE00118M] MSc Practical Robotics (PRAR)

[ELE00103M] Year 4 Robotics module

Lab Session 4 Handout

Lab 4: PWM and Motor Control

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

Technician: Mike Angus

1. Aims and Objectives

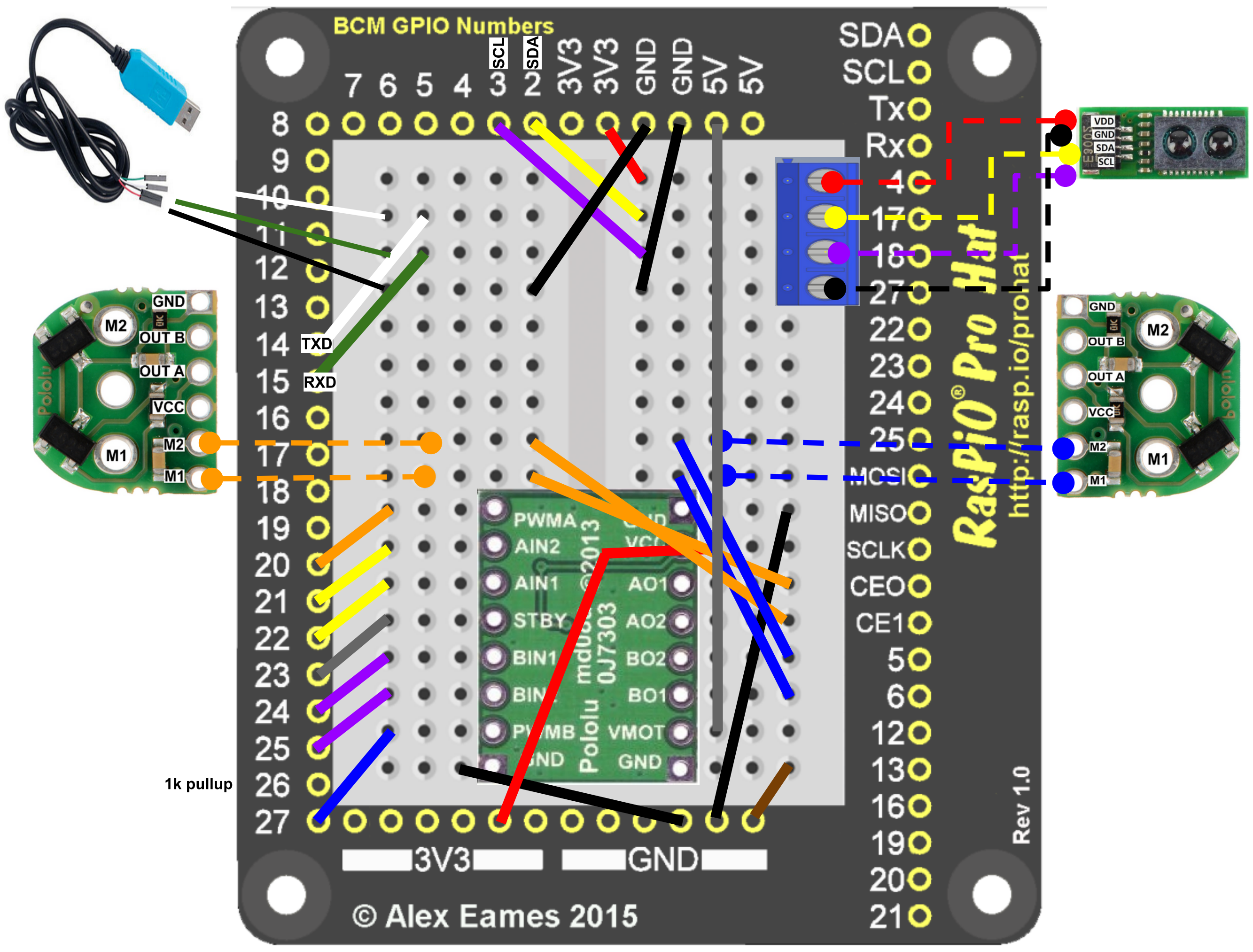
This laboratory will detail the use of bridge circuits to control motors and other actuators using GPIO pins from microcontrollers.

2. Learning outcomes

- Understand and use an H-bridge Integrated Circuit (IC) for motor control
- Move and brake a motor using a microcontroller

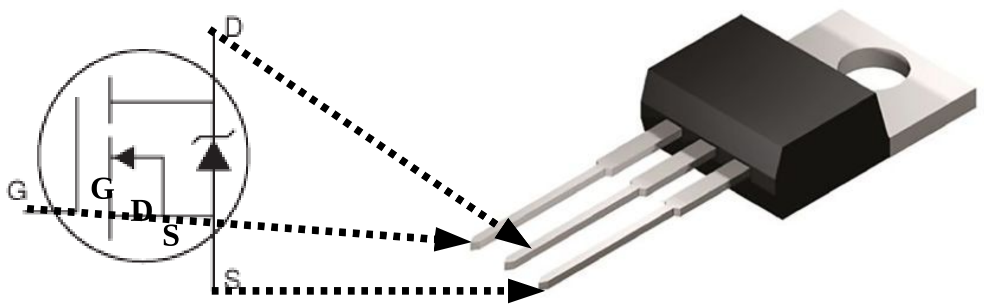
3. Software and hardware

You will use C++ programs using the Raspberry Pi from Lab Session 1. Similarly to Lab Session 2 you will use GPIO pins to control the movement of a motor. The GPIO pins on the Raspberry Pi will be used to communicate with the sensors. To reduce the current draw from the pins (which have a maximum of 50mA), we will rely on the resistors on the pins of the RasPiO Pro Hat and use an H-bridge IC that uses MOSFET transistors which do not draw current from their gate after switching like BJTs do. While assembling parts on your robot please refer to the “Wiring Instructions” section starting on page 50 of the document “**Home Robotics Kit 2020**” for guidance and tips on how to build a good quality robot.


BCM GPIO Numbers
7 6 5 4 3 2 3V3 3V3 GND GND 5V 5V
SCL SDA
8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
TXD
RXD
1k pullup
SDA SCL Tx Rx 4 17 18 27 22 23 24 25 MOSI MISO SCLK CE0 CE1 5 6 12 13 16 19 20 21
PWMA AIN2 AIN1 STBY BIN1 BIN2 PWMB GND
GND VCC AO1 AO2 BO2 BO1 VMOT GND
Pololu md08a ©2013 0J7303
VDD GND SDA SCL
GND OUT B OUT A VCC M2 M1
3V3
GND
© Alex Eames 2015
RasPiO® Pro Hat
http://rasp.io/prohat
Rev 1.0

To control your robot's motors, you will be using the TB6612FNG Dual Motor Driver IC. This chip is a DC motor driver that can control two motors at a time for most kinds of DC motors up to 15V and 1.2A in continuous operation. The figure above is a guide to recommended connections from the Raspberry Pi to the motor driver.

The IC incorporates N-channel Metal Oxide Semiconductor Field Effect Transistors (MOSFETs) with internal circuitry that makes them easy to turn on with 3.3V signals. The pinout for MOSFETS in a TO-220 package is below:



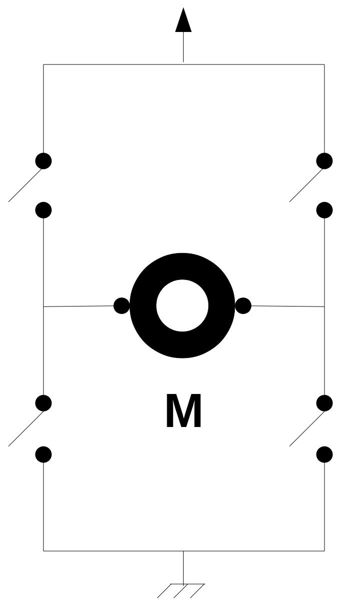
4. Pre-Lab Preparation

- Look up the datasheet for the TB6612FNG Dual Motor Driver IC and familiarize yourself with its control method, parameters, pinout and maximum ratings

5. Tasks

TASK 1: CONNECT THE MOTOR DRIVER IC

1. In this lab, you will use the TB6612FNG motor driver IC for motor control. This IC uses an H-bridge is to control the current in both directions through a DC device such as a brush DC motor by using four transistors as electronic switches. The basic diagram of an H-bridge is to the right, and you can see how it got its name (see also the TB6612 datasheet)



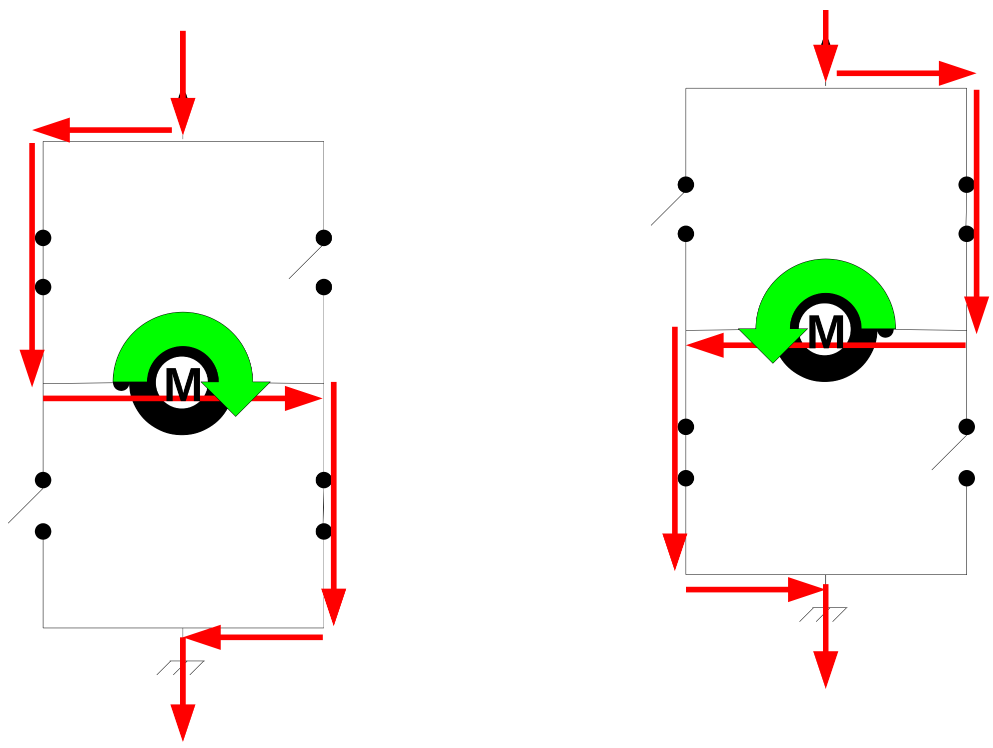
To connect the TB6612 to your Raspberry Pi, read through the datasheet and familiarize yourself with the signals on the IC and how they connect to the H-Bridge.

You will then have to connect up the pin signals to pins on your Raspberry Pi, You can use your own wiring layout and pins if you like but if you get confused, you can follow the above recommended wiring layout on the RasPiO Pro Hat.

TASK 2: CONTROL YOUR MOTOR WITH THE MBED

To allow current forward through the motor (denoted “M”), One switch on the “high side” is closed, and the opposite switch on the “low side” is closed as shown below. Speed control can be accomplished by rapid on-off switching of the “high side”, “low side”, or both.

To reverse the motor, current is directed opposite the previous direction by opening the previously-closed switches and closing their high and low-side counterparts. When the motor is suddenly reversed, note that back-EMF may occur from the previous inertia of the motor (voltage is generated opposite the driving voltage)



Once you have connected up all signals to the IC, make note of which Raspberry Pi pins are connected to which IC pins on the TB6612FNG and create a C (more challenging) or Python (easier) program to make the motors turn forward, reverse and stop. It is recommended to use the pigpio library as in Lab 1 to turn on and off the GPIO pins.

NOTE: if you are using Python remember to start the daemon with "`sudo pigpiod`" and in your code to set the pins as outputs before writing to them.

Some examples in C follow (they should be easy to convert to Python if desired) As before it is suggested to define macros or variables in your program for the pins to make them easy to remember. The example uses the following mapping:

```
#define STBY 23
#define PWMA 20
#define PWMB 27
#define AIN1 22
#define AIN2 21
#define BIN1 24
#define BIN2 25
```

All these pins need to be configured as outputs:

```
gpioSetMode(STBY, PI_OUTPUT);
gpioSetMode(PWMA, PI_OUTPUT);
```

```
  gpioSetMode(PWMB, PI_OUTPUT);
  gpioSetMode(AIN1, PI_OUTPUT);
  gpioSetMode(AIN2, PI_OUTPUT);
  gpioSetMode(BIN1, PI_OUTPUT);
  gpioSetMode(BIN2, PI_OUTPUT);
```

To make the motors move, you will first have to turn the standby mode of the IC off. This is accomplished by making the state of the *STBY* pin high:

```
  gpioWrite(STBY, 1);
```

To set the direction of the motor, you will need to follow the truth table on page 4 of the TB6612 datasheet. For example, to make motor A turn counter-clockwise (CCW), you need to set *AIN1* to low and *AIN2* to high.

```
  gpioWrite(AIN1, 0);
  gpioWrite(AIN2, 1);
```

Finally, the motors will not run unless the PWM inputs are high. We will use these inputs to control the motor speed later in the lab but for now you can just set them high to turn the motor on continuously.

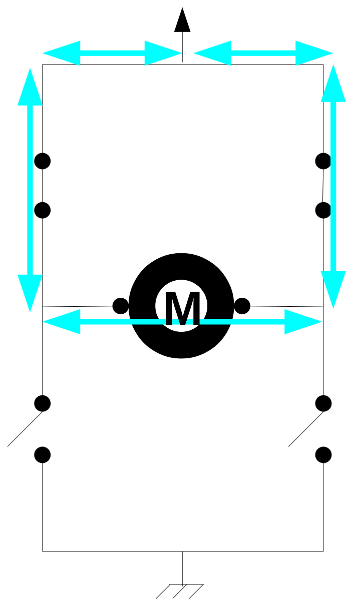
```
  gpioWrite(PWMA, 1);
```

Using these examples and the truth table, program the motors to (for some period of time, say 5 seconds each) firs t run CCW, then run CW, then pull the PWM pins low to stop.

NOTE: If you have your robot assembled and sitting on its wheels, make sure to prop it up on something that prevents the wheels from touching the floor or it may run away.

TASK 3: IMPLEMENT BRAKING FOR YOUR MOTOR

To stop a DC motor, you can very easily use the back-EMF generated by external forces to create a current that opposes further motion. Simply connect the terminals of the motor together, and if the shaft is turned it will create a current (like a generator) that in turn will cause a torque that *opposes* the motion of the motor.



Modify your program to "short brake" both motors after the first change of motor direction (CCW) using the truth table. Then try to "high impedance" or "stop" brake the motor after the second change of direction (CW). Do you notice a difference between the two kinds of braking?

TASK 4: SET UP PULSE-WIDTH MODULATION (PWM)

In robots, it is generally necessary to control the speed of a motor as well as it's direction. This could be accomplished by changing the input voltage to a DC motor and therefore changing the motor current and hence the applied motor torque, but in practice, variable voltage supplies are complex and limited in efficiency. A much simpler and more compact method is to directly control the "average" current into the motor by switching it on and off rapidly. This is called pulse-width modulation (PWM). If switching is done fast enough, the motor's internal inductance and mechanical momentum will keep the rotational speed constant and back-EMF generated during the off-cycle of the H-bridge will be absorbed through the flyback diodes in the H-bridge MOSFETs.

The Raspberry Pi 4 is capable of providing hardware-generated PWM on every GPIO pin, and this is accessible using the pigpio library. The C functions are described with examples at http://abyz.me.uk/rpi/pigpio/cif.html and the Python functions at http://abyz.me.uk/rpi/pigpio/python.html. Examples given in C here can be easily translated to Python.

The following line in C defines a pin output as PWM with a given duty cycle *dutyA*:

```
gpioPWM(PWMA, dutyA);
```

The equivalent in Python is:

```
pi.set_PWM_dutycycle(PWMA, 255)
```

Duty cycle is the fraction of time the pin is on relative to the total period, and it is expressed as a value from 0 to 255 as if it was controlling an 8-bit timer/counter on a microcontroller. For example if you want to have 50% duty cycle (half-speed) you would set *dutyA = 127*.

NOTE: Remember to define dutyA as "int dutyA = 0" before your main program loop if you are programming in C.

Aside from duty cycle (the percentage of time the PWM is on), you also have control of frequency (the speed at which pulses are repeatedly sent. You can change the PWM repetition frequency to 50Hz using the following line in C:

```
gpioSetPWMfrequency(PWMA, 50)
```

The equivalent in Python is:

```
pi.set_PWM_frequency(PWMA, 50)
```

The value to use for a given frequency is given in the table that you can find at http://abyz.me.uk/rpi/pigpio/cif.html#gpioSetPWMfrequency or http://abyz.me.uk/rpi/pigpio/python.html#set_PWM_frequency. Note that pigpio cannot reproduce any given frequency due to limitations of the hardware. The function will return the frequency selected so that you can check what frequency is actually used.

- Set up your program to first set the PWM duty cycle of your motors as 127 (50%) and then try different PWM repetition frequencies to see which ones work with the TB6612 motor driver and which ones do not. What is the lowest frequency at which the motor can be reasonably run? What is the highest that works with the TB6612?
- Then, set up your program to use 5 microseconds PWM repetition frequency and have it first increase the PWM duty cycle of your motors from 0 to 255 in 1 second increments, then have it decrease to 0 again. This will familiarize you with the movements of your motors. What is the minimum PWM value below which the motor does not turn? What is the implication of this for a robot?

TASK 5: CONTROL THE ROBOT WITH PULSE-WIDTH MODULATION

You now have the capability to control your robot using the motor driver (as always though, be careful to test it with the wheels off the ground).

Write a program using all the knowledge you have gained that reads keystrokes from the serial terminal, and changes the speed of the motors to allow you to drive your robot under keyboard control. It is recommended to use a “gaming” style keyboard mapping where you can set the direction using the *w,a,s,d* keys and scale the maximum motor speed using the number keys *1...9*.

Practice driving your robot under keyboard control and try to improve the smoothness and controllability by gradually changing the PWM values rather than changing them all at once. You can use conditional statements to smoothly change values such as:

```
pwma_current = 0

pwma_set = 10

if(pwma_current < pwma_set)

    pwma_current = pwma_current + 1

if(pwma_current > pwma_set)

    pwma_current = pwma_current – 1
```

EXTRA TASK: IMPLEMENT A ROBUST MOTOR STOP

One often-overlooked but absolutely critical feature in a robot is the ability to stop it if something goes wrong. Because PWM signals and pins are have their states stored asynchronously from running program code, if your program encounters an error or is stopped manually, the motors will continue running.

If you are using Python, you can implement a way to stop your motors if anything goes wrong in your program by enclosing your main program code with a `try` statement. If anything goes wrong within this indented `try` block, the program will immediately go to the code under a following `except` block, where you should put statements that will stop your motors immediately (e.g. set PWM and STBY pins low). The following code example illustrates how to use `try` and `except`:

```
try:
    while True:
        print("in main loop")
        time.sleep(1)
except:
    print("put motor stop code here")
```

You can also implement an external emergency stop using the light sensor described in previous labs or another kind of switch that is easy to press from outside the robot. You could program this sensor to break you out of the main loop with a `break` statement, or you could also manually trigger the exception handler described above by using a `raise()` statement such as `raise(RuntimeError)`.

There is no native exception handling in the C language, and you can just use a simple `goto` statement to break out of the loop to a motor stop routine if desired. If you are using C++ though you can use the C++ `try` and `catch` statements for exception handling in the same way. Some information on using C++ `try` and `catch` is here: https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm