ECE5725 Lab 3

Outline

In Lab 3, we will explore the following:

- Add output capability to the R-Pi through exploration of a number of attached components.
- Build a robot platform for the RPi
- Explore robot applications for the RPi and integrate these with piTFT feedback
- One overall goal will be the development of **reusable and modular code** that can be used in successive steps of the lab.
    - Consider software design before beginning code in order to map operations into python functions that may be used in future parts of the lab.
    - Plan to show design of software components, used to achieve some degree of code re-use, in your lab report

The goal of Lab 3 will be to complete an autonomous robot under control of an application to maneuver within the environment and to display status and control information. The robot will be self-contained and not be connected to any external devices

Lab safety

We have all been in lab before and handled electronic components. Please apply all the cautions that you have learned including the items below:
Integrated electronics are REALLY susceptible to static discharge.

- Avoid walking around while holding bare components
- Confine assembly to the grounded mats on the lab bench
- Make sure you ground yourself by staying in contact with the mat.

Personal safety

- Safety goggles are REQUIRED for soldering

Experimental Safety

- If something on your board is
    - Really hot
    - Smoking
    - Just doesn't look right
- Immediately unplug power

- Train yourself so that this is your first reaction – know where the power switch/cutoff is for your experiment.

Experimental assembly

- Before adding any hardware to the system, power should be OFF
- Before powering up the equipment, please have staff check your added circuit

GPIO Output

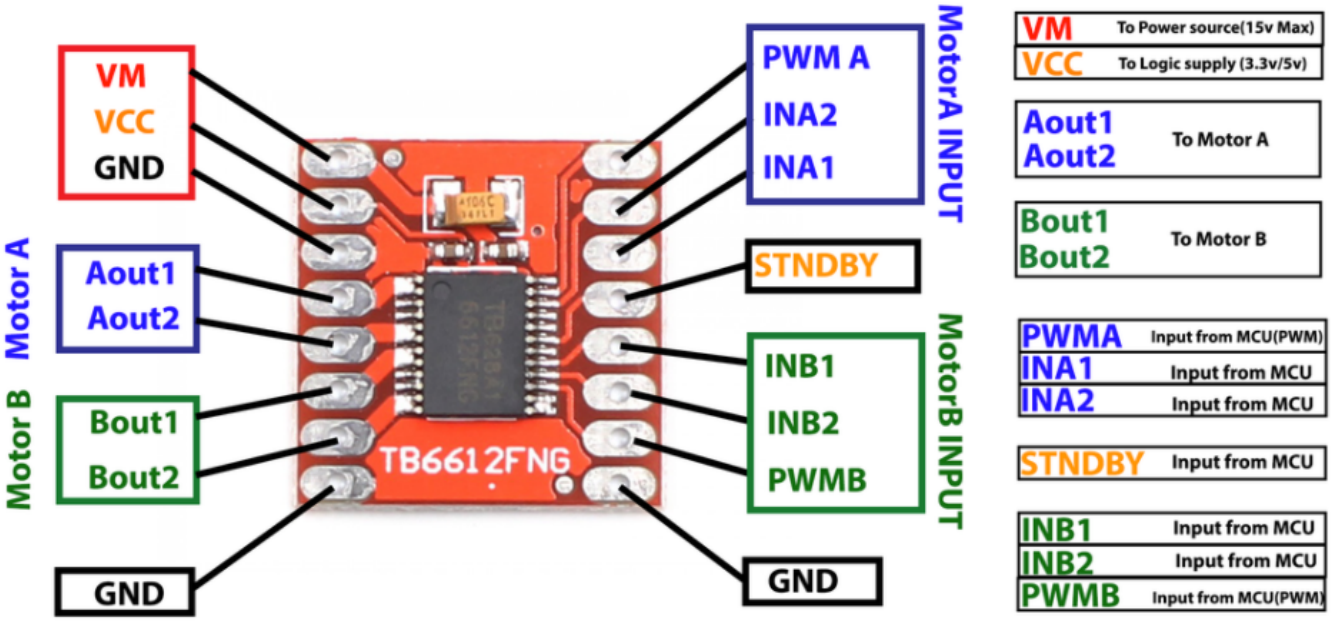
Lab 2 included experiments involving using Raspberry Pi GPIO pins as inputs. This experiment involves using GPIOs as outputs, driving a DC motor using Pulse Width Modulation (PWM) from a GPIO output.

From the information on Canvas, some information on the DC motor and controller is reproduced here. Please make sure to read through the data sheets for these components:

Technical Details for the yellow DC motor:

- Rated Voltage: 3~6V
- Continuous No-Load Current: 150mA +/- 10%
- Min. Operating Speed (3V): 90+/- 10% RPM
- Min. Operating Speed (6V): 200+/- 10% RPM
- Torque: 0.15Nm ~0.60Nm
- Stall Torque (6V): 0.8kg.cm
- Gear Ratio: 1:48
- Body Dimensions: 70 x 22 x 18mm
- Wires Length: 200mm & 28 AWG
- Weight: 30.6g [1]

A figure showing connections for the Sparkfun TB6612FNG dual-channel motor controller



[2]

Yet Another Reminder: Why protect the pins?

While the GPIO pins can provide lots of useful control and sensing ability to the Raspberry Pi, it is important to remember they are wired directly into the internal core of the system. This means that they provide a very easy way to introduce bad voltages and currents directly into the ARM chip on the Raspberry Pi (this is not good and means it is easy to break it without exercising a little care).

Things we need to protect:

1. Drawing excess current from the pins (or short-circuiting an output)

2. Driving over-voltage on an input pin (anything above 3.3V should be avoided). The RPi has protection diodes between the pin and 3.3V and GND, negative voltages are shorted to GND, but positive voltages greater than 3.3V + one "diode drop" (normally 0.5V) will be shorted to 5V, this means that if you put a 5V power supply on the GPIO pin you will "feed" the 3.3V supply with 4.5 Volt (5V - the diode drop) and that may damage 3.3V logic if the 5V source succeeds in lifting the RPi's 3.3V supply to an unsafe value. Note that if you limit the current (for example with a 1K resistor) the small amount of current flowing into the 3.3V supply will do no harm.

3. Static shocks, from touching pins without suitable grounding (Electrostatic Discharge or ESD)

4. Please be aware of maximum current draw from any devices you use (for example, 5 volts). Use this information to power devices accordingly.

5. In particular, consider how much current you will draw from the Raspberry Pi if you decide to connect 3.3V or 5V pins on the device. Remember, the maximum current from any Pin on the RPi is extremely low.

All of these can potentially damage Raspberry Pi, damage the GPIO circuits or weaken the RPi over time (reducing its overall life).

Step1: Using PWM in RPi.GPIO

You will be using the rpi.GPIO library to develop a control application for the motor. As in Lab 2, you will be required to include the rpi.GPIO library and, for this experiment, select a GPIO pin and configure it as an output. Some general tips on using a GPIO pin to drive the motor using PWM (from the link in Canvas 'References'):

To create a PWM instance:

```
p = GPIO.PWM(GPIO_pin, frequency)
```

To start PWM:

```
p.start(dc)   # where dc is the duty cycle (0.0 <= dc <= 100.0)
```

To change the frequency:

```
p.ChangeFrequency(freq)   # where freq is the new frequency in Hz
```

To change the duty cycle:

```
p.ChangeDutyCycle(dc)  # where 0.0 <= dc <= 100.0
```

To stop PWM:

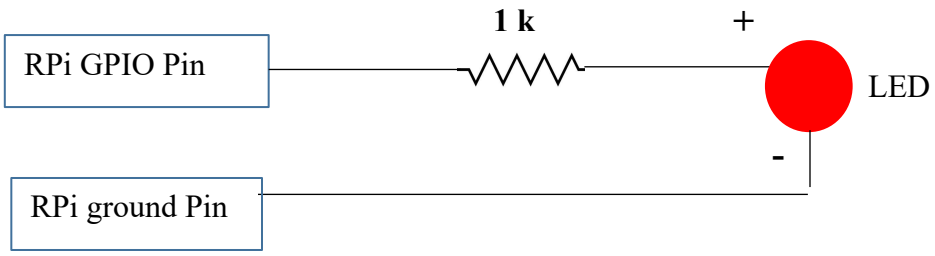
```
p.stop()
```

Note that PWM will also stop if the instance variable 'p' goes out of scope

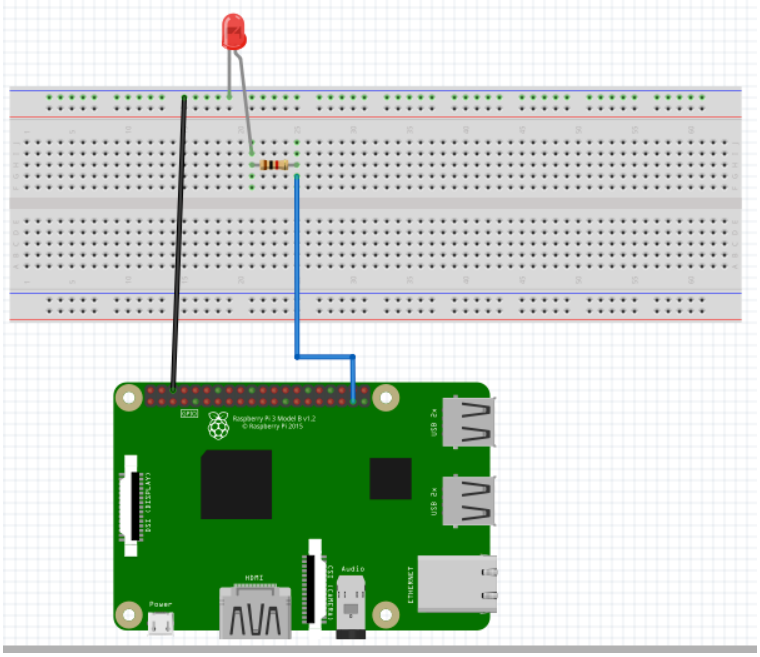
IMPORTANT NOTES:

1. **Please have the TA confirm your circuit before plugging in the power.**
2. Please read the data sheets for the DC motor and motor controller: There is a lot of information in these documents that will help you to progress on Lab3

First, implement an LED circuit on the selected output pin and experiment with blink rates using PWM settings. Design a python code, blink.py, to use rpi.GPIO PWM calls to blink an LED. You can use the blink rate of the LED to verify the PWM settings. simple schematic of an LED on a GPIO pin:



The following is a connection diagram for the LED on a RPi:



Pin 26 is used in this example. Note the use of a 1k current limiting resistor

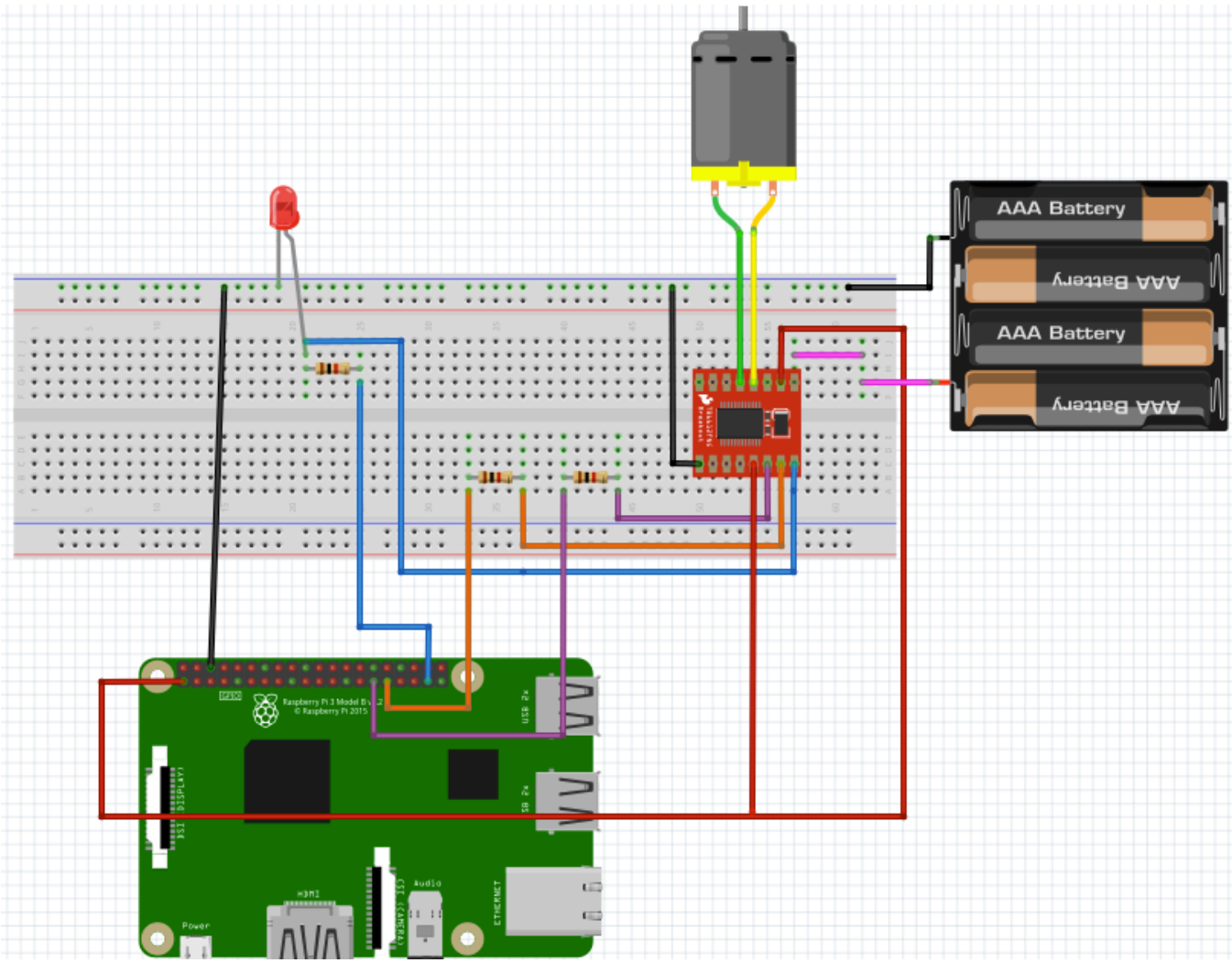
STOP: Please Have a TA confirm your circuit before applying power

Use blink.py to blink the LED on and off over a second. You should use PWM calls to initially set a blink frequency of 1 Hz with a duty cycle of 50%. This should allow for a visible confirmation of PWM operation as you will be able to see the LED blinking. The blink program should also include an integer argument which is used to adjust the blink frequency.

Please see the appendix for a nice tool you can use to check the GPIO PWM operation. Once you have checked the appendix, please plan to use the oscilloscope to display pwm signals. Note that you may also use piscope to display pwm signals (see the appendix). Record the signal for the blink program once you have it running.

Once you are satisfied that you understand the PWM parameters, connect a single DC motor and motor controller as defined in the data sheets. Below is a figure showing connections for:

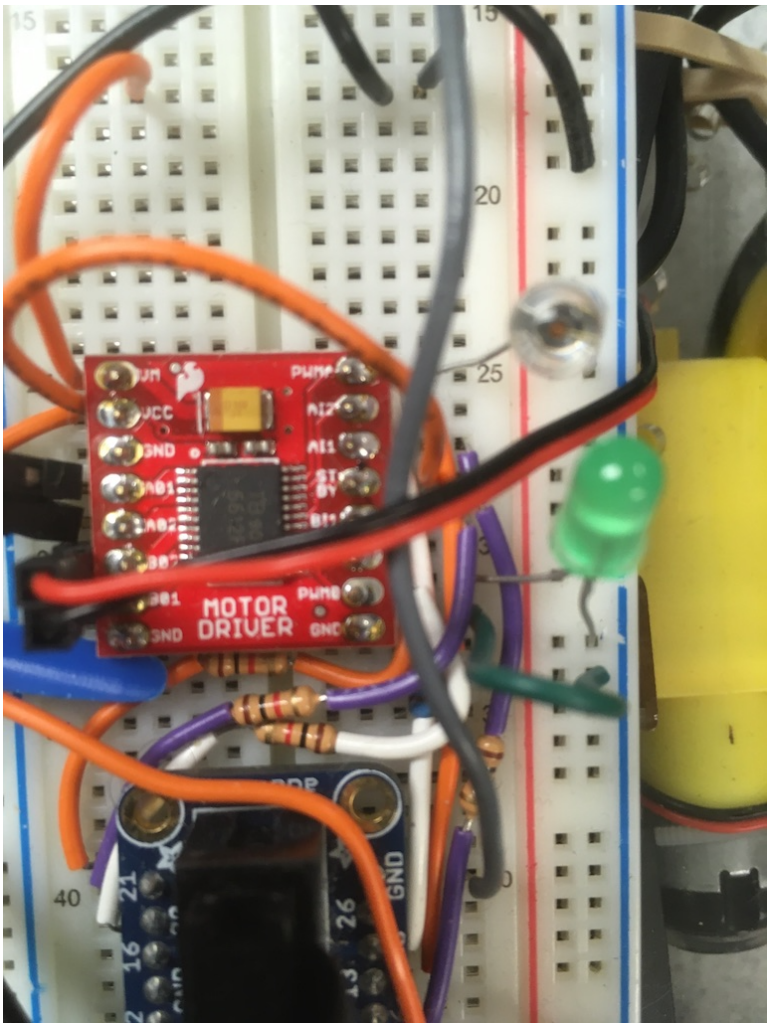
- Raspberry Pi
- LED to GPIO used for PWM Signal
- Motor Controller
- DC Motor
- 6V Battery Power



Notes:

- GPIO Pins 5 and 6 are shown connected to AIN1 and AIN2, used for direction control
- GPIO Pin 26 is again used for PWM (as in the previous PWM example)
- Note that the LED remains in the circuit; the LED provides a simple visual check of PWM output.
- 3.3 Volts from RPi is connected to motor controller VCC
- 6 Volts from battery is connected to VM, voltage for the DC motor
    - At this point in the lab, use the bench power supply to power the motors.
    - Note that a jumper is shown on the breadboard for this connection
- The DC Motor is connected to A01 and A02
- 1K current limiting resistors are used to protect GPIO pins
- Grounds for RPi, Motor controller, and 6V battery are all connected
- For connection details, check Lab3 videos on Cornell Box

The following photo shows and example wiring of the Motor Controller to the RPi and a DC Motor:



Notes:

- This photo focuses on control pins for motor B on the Motor Controller (MC).
- GPIO 20 and 21 are used for direction control and connect to BI1 and BI2 on the MC.
- GPIO 16 is connected to PWMB which is used for speed control
- Typical values for PWM are similar to those in the data sheet for the continuous rotation servo. Use a frequency of 50 Hz at 100% duty cycle for full speed rotation of the motor. Experiment with 50% to 75% duty cycle to get ½ motor speed

- Note that connections are made using the 1K resistors directly. The resistors have been protected using the insulation technique described in the Part Kit 2 unpacking video. This allows Resistors to route closely without danger of shorting wires.
- Placement of the MC relative to the GPIO breakout cable allows for wiring with resistors to be used

Stop: Please have the TA check your motor connections before applying power to the Raspberry Pi or to the motor.

Once the circuit is checked, develop a python application named 'motor_control.py' to perform the following functions:

- When the program starts, the motor should be stopped
- Range the speed of the motor through three speed steps (stopped, half-speed, full speed) in the clockwise direction
    - Each speed increment runs for 3 seconds
    - Print an indication on the screen for each speed increment
- Range the speed of the motor through three speed steps (stopped, half-speed, full speed) in the counterclockwise direction
    - Each speed increment runs for 3 seconds
    - Print an indication on the screen for each speed increment
- Return the servo to 'stopped' state

Notes:

- Before you apply power to the motors, you can verify that you have a PWM signal by observing the LED. You'll notice different LED brightness as you vary PWM duty cycle.
- You can also check PWM operation using either the oscilloscope or piscope. Please plan to record various settings of the GPIOs in use for your motor control operation.
- Pay special attention to settings for:
    - Clockwise and Counterclockwise, full speed
    - Clockwise and counterclockwise, approximately ½ speed
    - Stopped
- For each experimental step in each direction, record:
    - Frequency
    - Pulse_width
    - Duty_cycle

Step2: Left and Right Motors

Select unused GPIO pins for the attachment for two DC motors. Attach the motors for correct power and control using the RPi

STOP: Please have a TA check your circuit before proceeding

Implement two_wheel.py with the following functions:

- Provide a function for driving motors full speed clockwise and counter-clockwise. The function should also provide a command to idle (stop) the servo. Parameters to the function include servo number and direction (clockwise, counter-clockwise, stop)
- Design a test program to control the servos using buttons with the following functions:
    - Left servo, clockwise
    - Left servo, stop
    - Left servo, counter-clockwise
    - Right servo, clockwise
    - Right servo, stop
    - Right servo, counter-clockwise
- Note that these functions should be assigned to physical buttons (on the piTFT or externally connected to the RPi GPIOs). Hint: There may be a clever way to implement these 6 functions with fewer than 6 physical buttons.
- Note: Please make sure to implement correct commands for servo stop (according to the table in the connection guide)

Demonstrate the following python applications to the TA before proceeding:

- blink.py
- motor_control.py
- two_wheel.py
- Show saved screen shots from either the oscilloscope or PiScope demonstrating PWM signals for the LED blink program and for controlling motors (hint: Half-speed PWM for motors would be a good motor PWM example)

Please take a backup of your SD card at the end of your demo.

Week 2:

You completed steps 1 and 2 at the end of week 1 lab. For week3, we will continue with the step3, rolling control and move on to the build of the robot.

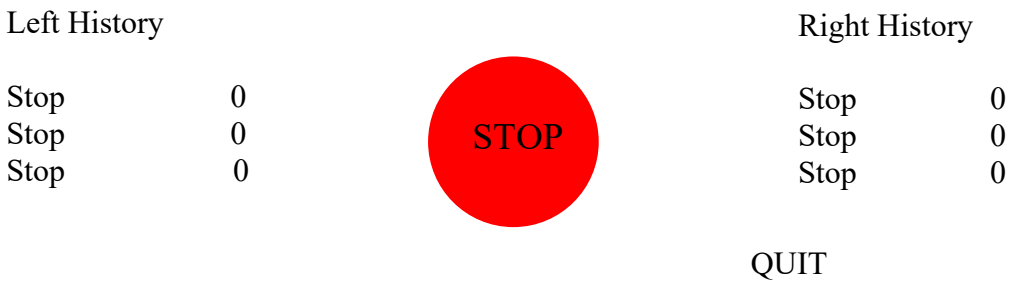
Step3: rolling control

Implement a python program, rolling_control.py, with the following functions

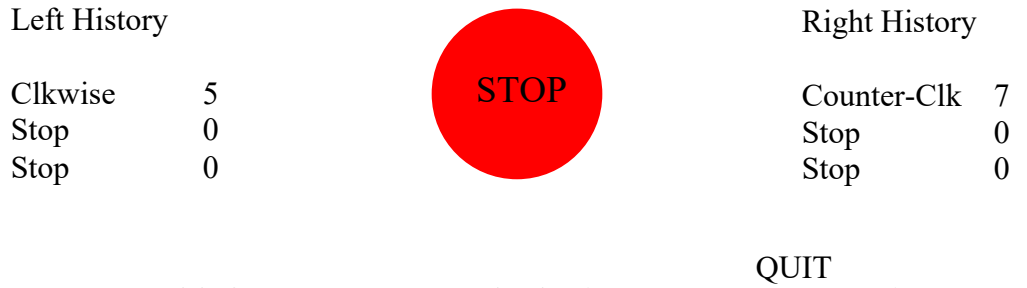
- On the piTFT screen, display direction (clockwise, counter-clockwise, stopped) for each motor
- Implement a single, red 'panic stop' button on the piTFT. If pressed, motors immediately stop and 'panic stop' changes to a green 'resume' button
- Implement a 'quit' button on the piTFT. When hit, quit causes the program to end and control returns to the Linux console screen.
- Record start-time/direction pairs for each motor and display a scrolling history of the most recent motion (include 3 past entries for each motor).
- Integrate the functions in rolling_control.py along with two_wheel.py button functions to insure correct display, on-screen button operation, physical button operation and correct motor operation.

One possible layout for the display would be:

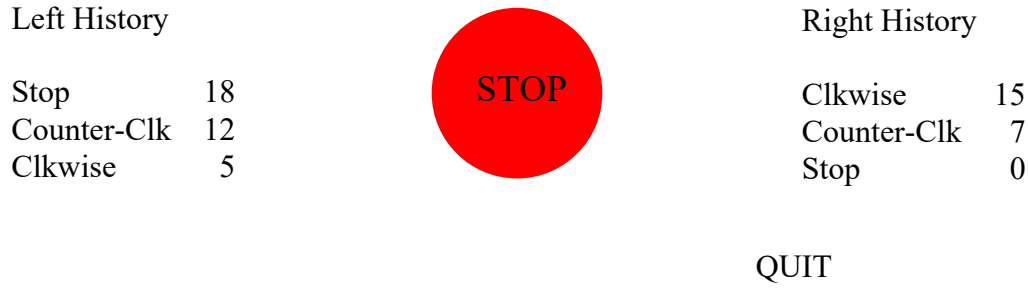
Initial state, motors not running:



At time 5, command left motor Clockwise. At time 7, command right motor Counter Clockwise:



At time 12, command left motor counter clockwise. At time 15, command right motor Clockwise. At time 18, command left motor stop:



Note that in this example, the "Left and Right History" entries would update when you hit a button to change speed and/or direction on either of the motors. The values represent time in seconds when each action started, as measured from time = 0 = start of program. These entries represent scrolling histories of the motor activity.

Once you have finished rolling control.py, please demonstrate the function of the program for one of the TAs.

In this portion of the lab, you will assemble a robot frame and include the Raspberry Pi, piTFT and servo motors to make a mobile system. Following the steps below, you will incrementally move towards an embedded, untethered system able to maneuver in its environment.

Step 4: Frame it: Robot assembly

Some ***Important*** notes on fasteners:

- Assembly will use machine screws. A machine screw is threaded for use with a threaded attachment or for the attachment of a washer and threaded nut.
- Always 'start' a machine screw by hand, not with a screwdriver or other tool. This will insure that the threads will not be damaged by stripping as a screw is forced using a tool.
- There may be some cases where sheet metal screws are used, which are self-tapping screws for metal, wood, or plastic.
- A self-tapping screw forms threads in the material as the screw is installed.
- Do not over tighten the self-tapping screws. Once they are fully installed, tighten the screw just enough for it to grab the material and hold the two pieces together.

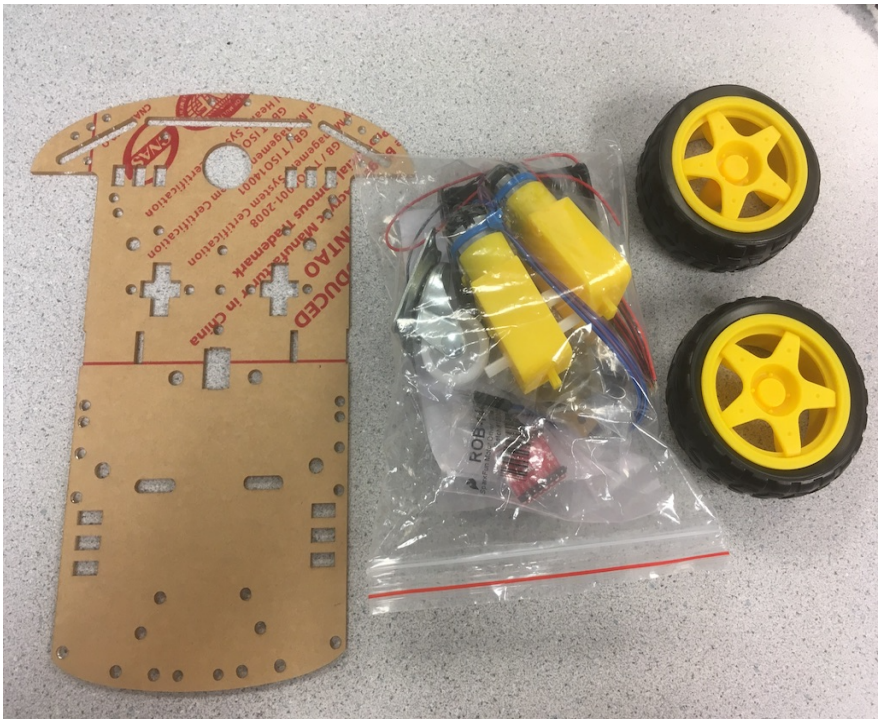
¼, 5/8, and ½ inch sheet metal screws

Machine screw, washer and nut

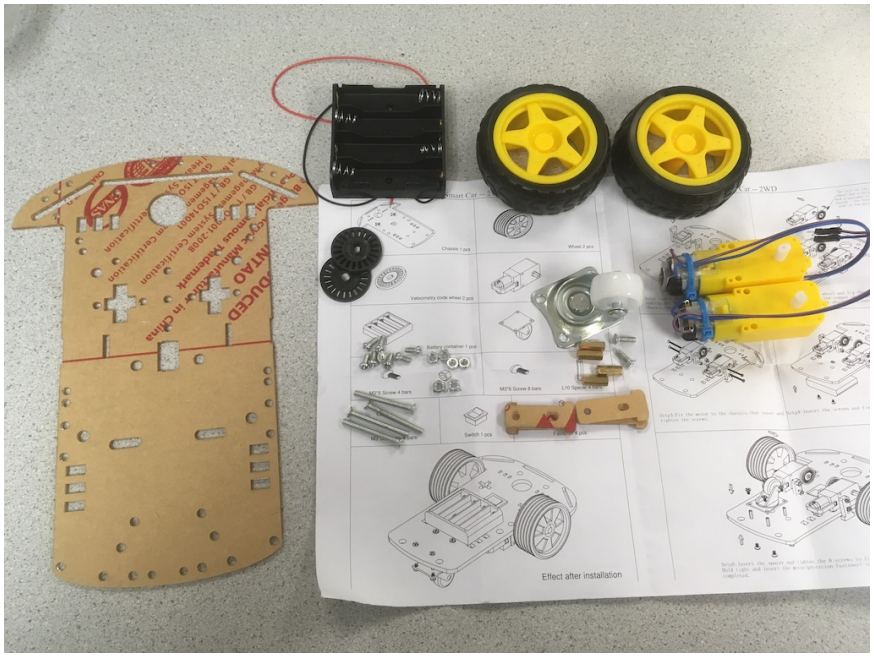
- Please talk to a staff member if you would like a demonstration of correct fastening procedures.

Build a robot frame as follows:

There are a set of step-by-step robot construction videos on Cornell Box at https://cornell.app.box.com/folder/131665112610 Please have a look. Below, are step-by-step instructions for robot assembly:

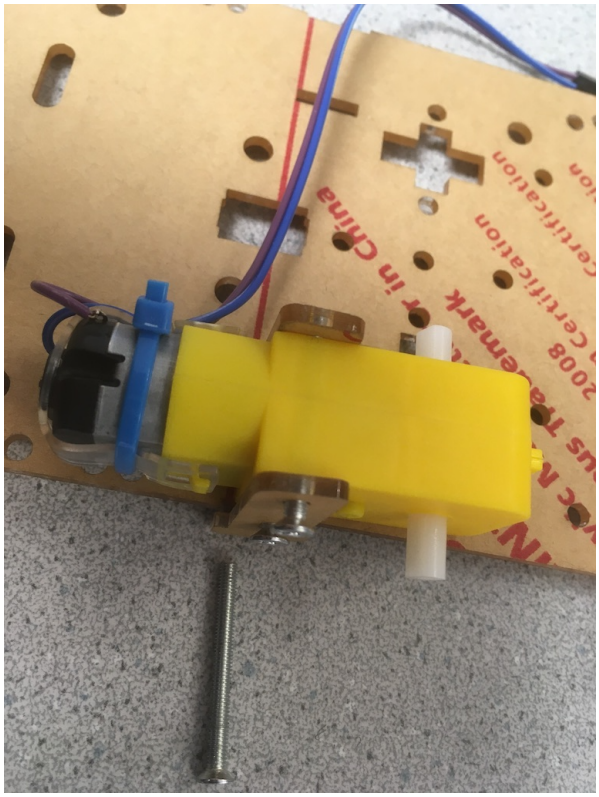


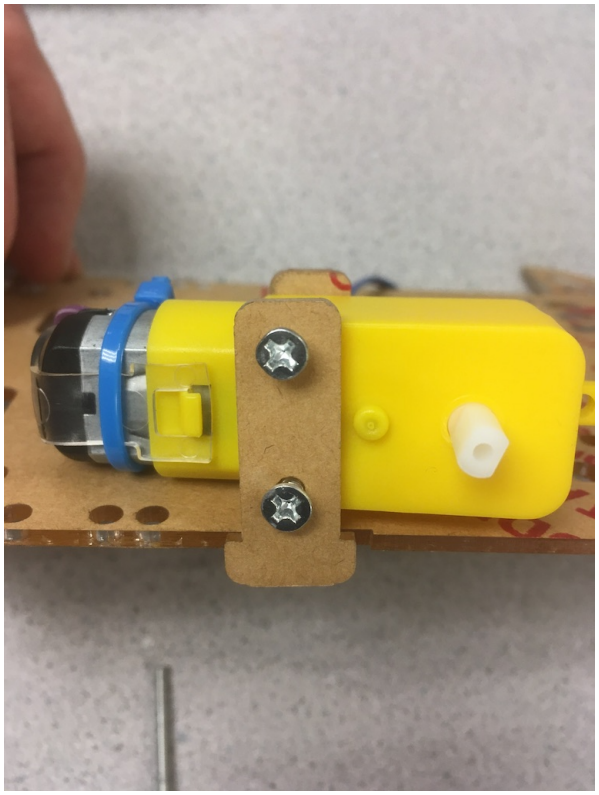
Initial Robot Parts



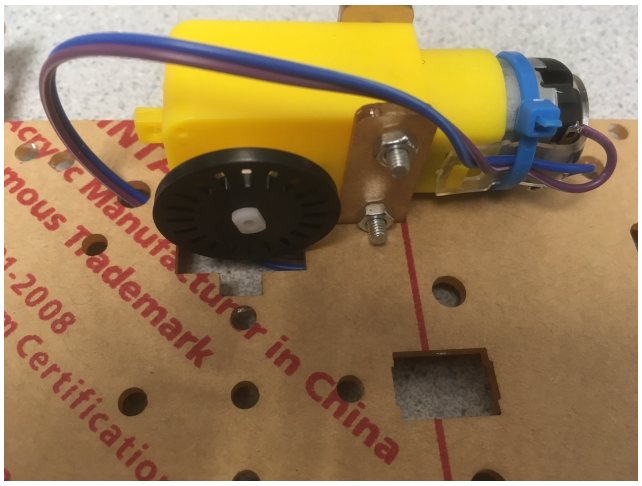
Robot Parts Sorted Using Instruction Sheet

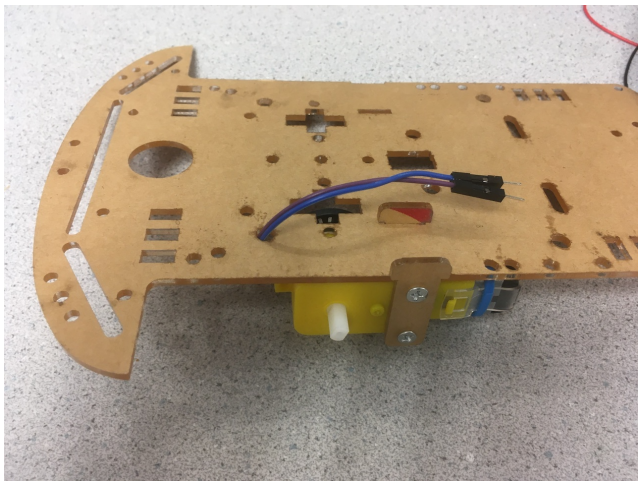
Install 1 motor to the main acrylic based using two of the acrylic fasteners, two M3x30 screws and two M3 nuts. The following Photos show some of this assembly detail:



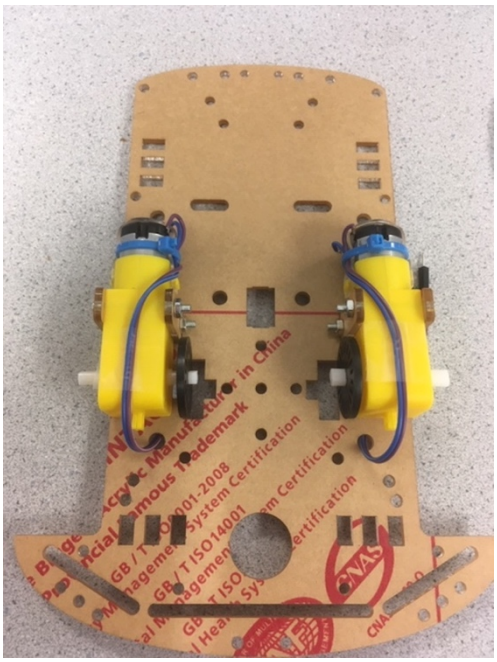


In the figure , above, the left photo shows the DC motor installed to the main base using 2 long bolts. An example M3x30 bolt is shown at the bottom of the photo. The bolts are not fully seated in order to illustrate installation position. Motor wires are mounted in-board rather than on the outside edge of the base. Note the position of the acrylic fasteners in this and the right photo. The right photo shows a side view of the motor. Note the position of the outer fastener relative to the base.

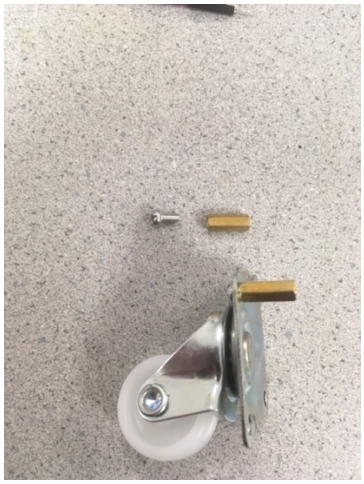




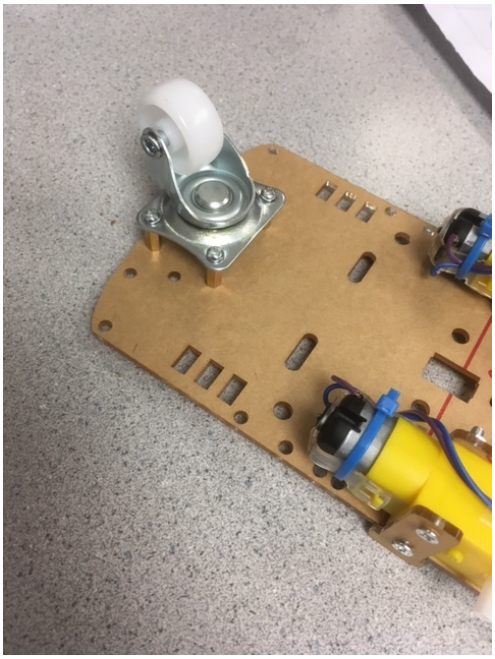
In the figure, above, the left photo shows the inner side view of the motor with the M3 nuts installed on the M3 bolts. Note the position on the inner acrylic fastener and the position of the black emitter disc on the inner motor axle. The right photo shows the top side of the base with the installed motor including wire routing.

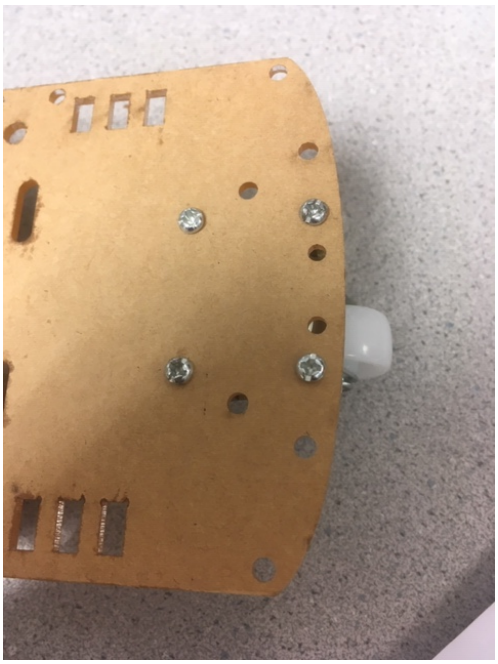


This Photo shows both motors installed on the base. Note the addition of a tiny piece of tape (on the yellow motor housing) to route the motor wires away from the emitter disks and spinning motor axle.

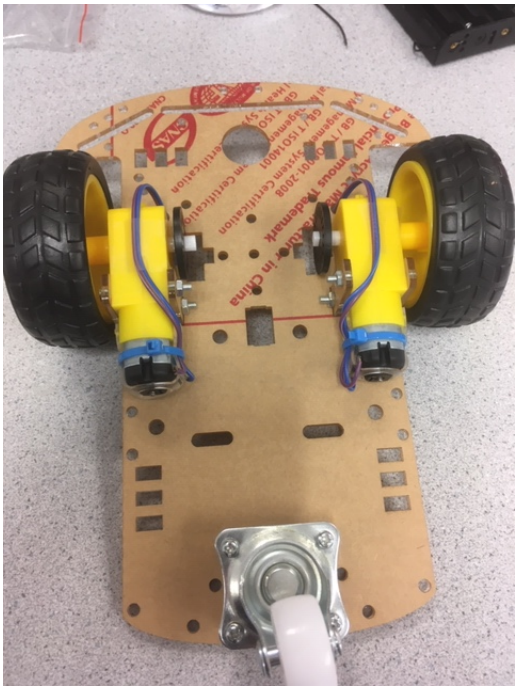


The photos, above, show the pre-assembly of the robot caster. The left photo shows the installation of the spacer to the caster using one M3x8 screw. The screw and spacer are also shown at the top of the photo. The right photo shows four spacers attached to the caster. At this point, screws are loosely installed to make the next step easier

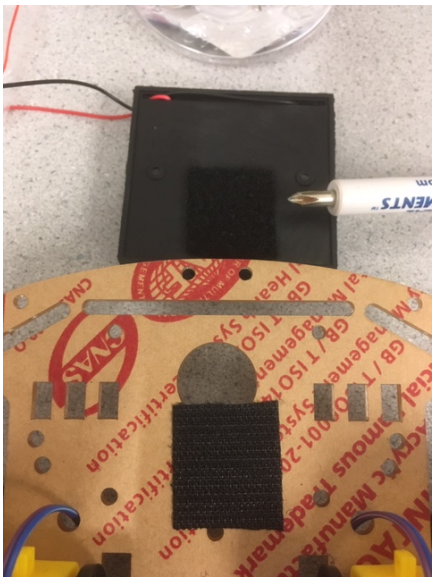


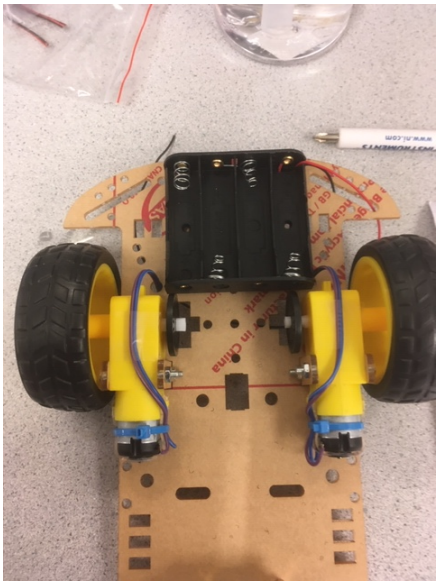


The Left Photo shows the caster installed on the base. Note that you may need to rotate the caster base 90 degrees to get the correct alignment with the robot base. The right photo shows the top side of the robot with the caster installed. Note the position of the four, M3x8 screws.

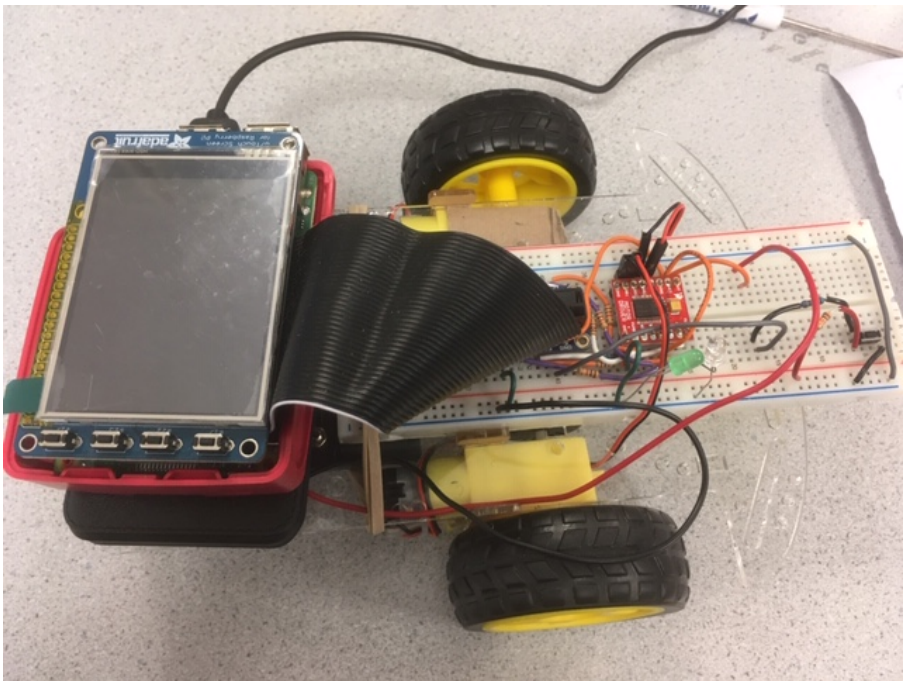


The left photo shows the wheel hub; note the hub shape matches the motor axle shape. The shapes should be aligned and the wheel will be easy to press onto the motor axle. The right photo shows both wheels installed





The left photo shows the hook (stiff) Velcro attached to the robot body. The loop (fuzzy) Velcro is installed to the back of the battery case; Note orientation of battery wires. The right photo shows the battery case installed on the underside of the robot base.



This Photo shows the protoboard attached to the robot base with Velcro (note DO NOT attach the board to the robot using the sticky foam on the back of the board. This destroys both the board and robot; use Velcro!). Over the caster, the RPi 5-volt battery is installed, also with Velcro. On top of the battery, the RPi is installed with a small Velcro pad between RPi case and battery.

Step 5: run_test code
Design a python code, run_test.py, that performs the following functions. Please run the motors at half speed for all functions:

- Move the robot forward about 1 foot
- Stop
- Move the robot backwards about 1 foot
- Pivot left
- stop
- Pivot right
- Stop
- …..return to top (loop continuously)

Use the piTFT display to display operation of the robot as defined in the rolling_control.py routine.

run_test.py should include the functions defined in rolling_control.py to establish the GUI displayed on the piTFT. This includes the left and right history displays, panic stop/resume and quit button. Please also make sure to implement a physical quit button and timeout function in your code.

run_test.py should also include a 'start' button. When run_test.py begins, the controls should be displayed on the piTFT but the robot will not move until the 'start' button is hit. The 'quit' button (designed in rolling_control.py) will exit the program, returning to a console window on the piTFT.

Note that the operation of run_test.py is automatic and does not require any physical button input.

Important: Backup your SD card before the next step! Critical! Do NOT skip this step.

Step 6: Configure the RPi to launch your application at boot

Setup run_test.py to start automatically once the system reboots. Select ONE of the methods below to run the application at boot

Method 1 (this is the preferred method):

- Add a call to run_test.py in /home/pi/.bashrc
- Make sure to use an absolute path in the call to run_test.py AND use sudo for the call
- Test this change by rebooting the system and logging in. After login, run_test.py should start.
- Using sudo raspi-config, configure the RPi to start without login
  - Select 'Boot Options'
  - Select 'Desktop/CLI'
  - Select 'Console Autologin'
- Test again to make sure run_test.py starts at power-up.

Method 2:

Method 2 has the advantage of running without login. However, modifying /etc/rc.local is a bit touchier than modifying .bashrc as in Method 1

- Make a copy of the file /etc/rc.local to preserve the original file (in case things fall apart later!)
- /etc/rc.local is a bash script which may be edited to include functions to run at system startup.
- There will be a trailing 'exit' in the /etc/rc.local file
- **BEFORE** the trailing exit, add a line in the bash script to start run_test.py after system reboot.
  - Use an absolute path to the script
  - Don't forget to include a leading sudo
  - Important: run_test.py should be set to run in the **BACKGROUND** so it will not interfere with the completion of the boot sequence on the RPi

  Once /etc/rc.local has been modified, reboot the RPi to make sure the application runs at startup.

Once one of the two methods have been tested and functions correctly, power the Rpi using a 5V cell phone battery and reboot the pi to start run_test.py. This will test the pi untethered from external power, keyboard/mouse and wired Ethernet.

Please plan to demonstrate the following software to the TA.

Rolling_control.py (demonstrated earlier in the lab; please confirm with TA at completion of lab)

run_test.py
For this demo, the robot should run untethered from

- Power
- Ethernet
- Keyboard
- Mouse
- Monitor

Run_test.py should launch at system startup

References:

[1] DC Gearbox Motor, Adafruit, https://www.adafruit.com/product/3777

[2] Driving Small Motors with the TB6612FNG, ArtisTech,

https://www.instructables.com/Driving-Small-Motors-With-the-TB6612FNG/

Appendix: Piscope

Piscope is a logic analyzer that runs on the RPi and can be used to check PWM operation. It is part of an excellent GPIO library, pigpio, which is already installed on the RPi. Piscope will run on either a monitor or VNC; if you have either capability working, please try piscope for debugging your PWM operations.

Getting Started:

Reference: http://abyz.me.uk/rpi/pigpio/piscope.html

Open a terminal window and run

```
sudo pigpiod  # to start the pigpio daemon
```

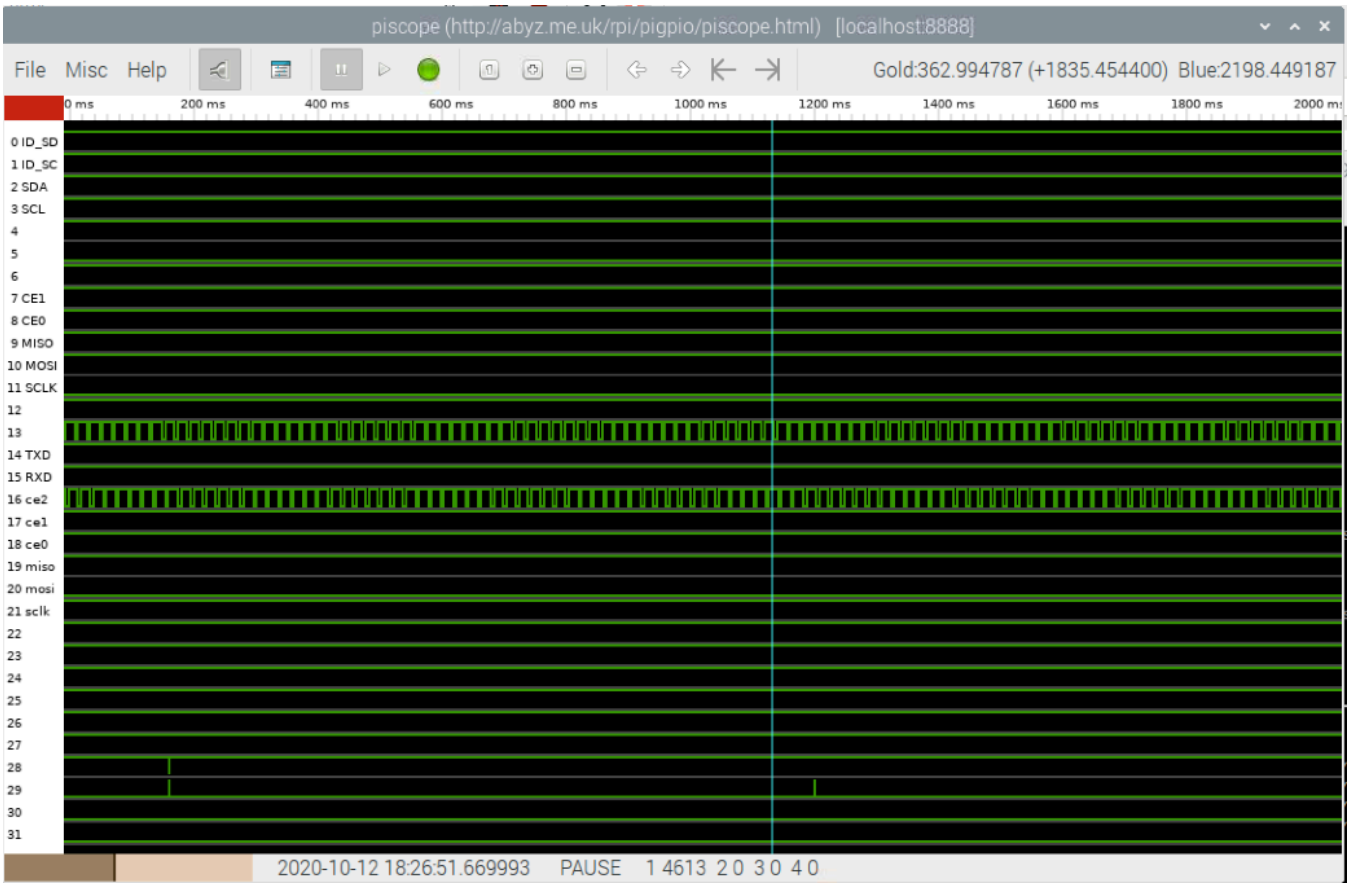
From the /home/pi directory on your RPi, use the following steps to download, compile and install the piscope code:

```
wget abyz.me.uk/rpi/pigpio/piscope.tar
tar xvf piscope.tar
cd PISCOPE
make hf
make install
```

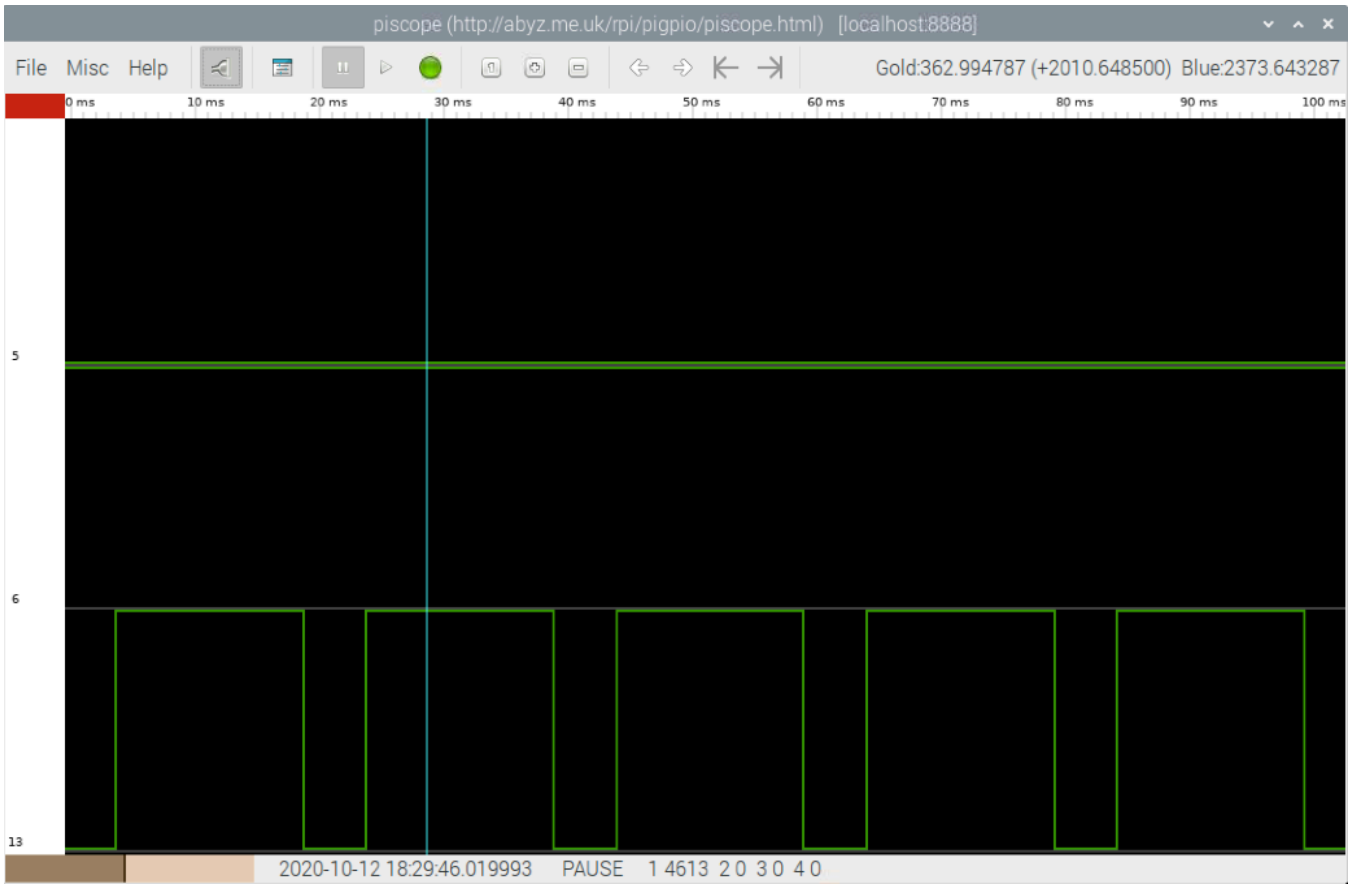
Once complete, run:

```
cd PISCOPE
./piscope
```

and piscope should start with a window:

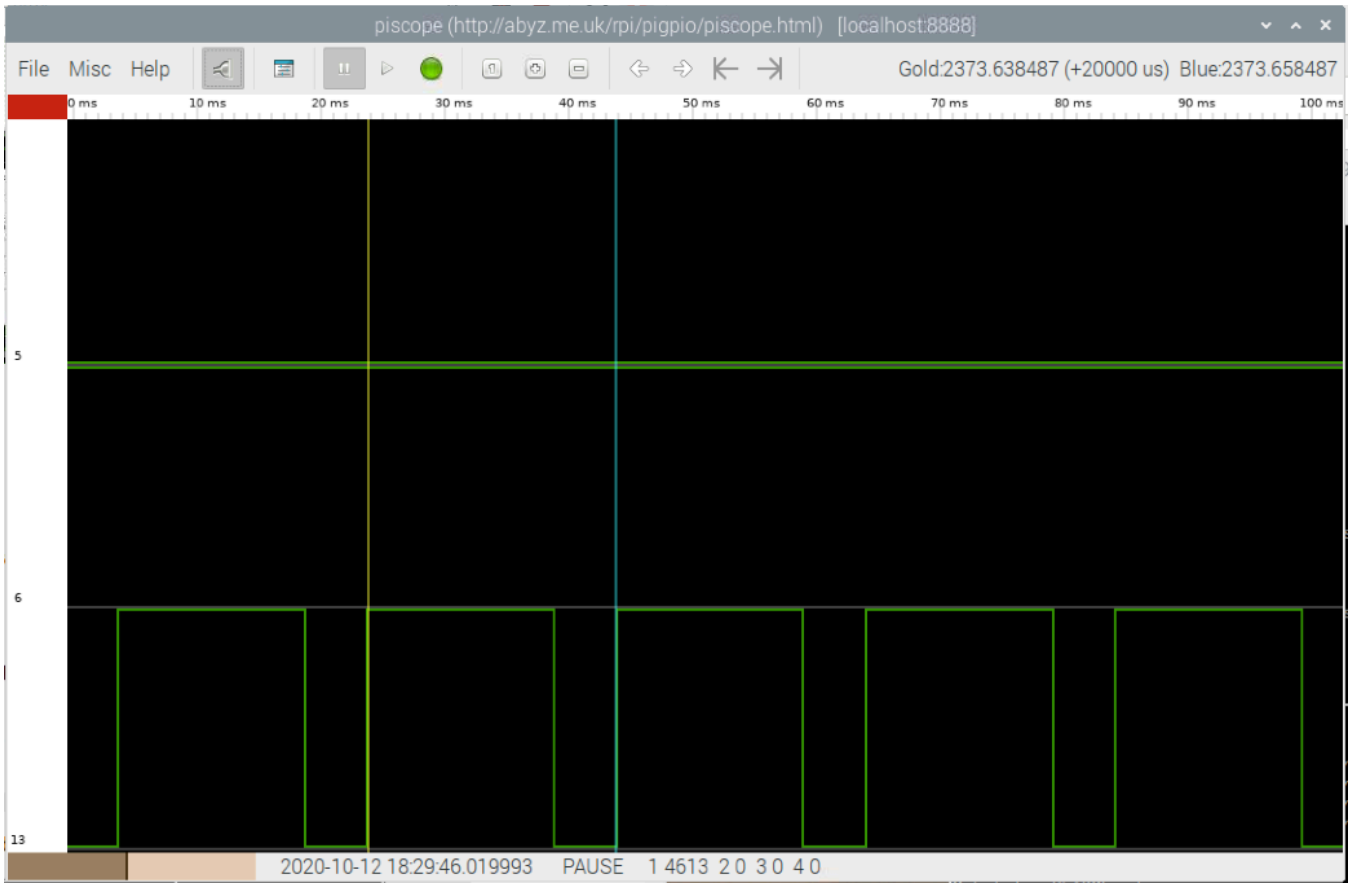


In this example, I am running PWM signals on GPIO 13 and GPIO 16. Use the ‘MISC’ tab to select which GPIOs you would like to monitor. The following example shows GPIO 5, 6, 13, used to control one of the motors:



There are controls sat the top of the screen. The green dot will set piscope to run mode and causes the system to sample the GPIO outputs over time. '+' and '-' are zoom controls. Pause stops recording (which is what I did for these screen shots).

In this next clip, I zoomed, paused, then clicked once in the screen which brings up a second cursor. I have the 2 cursors ranging over the PWM period and, in the upper right corner, you'll see the measurement is 20 millisec, or 50 Hz, which is the signal I am sending. This PWM signal also has a 75% duty cycle which may also be measured using these cursors.



Please check videos on Cornell Box form additional demos of piscope.