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 **modular code blocks** that can be reused 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 modules, used to achieve some degree of modularity, 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 continuous rotation Parallax servo motor using Pulse Width Modulation (PWM) from a GPIO output.

From the Parallax data sheet (included on Canvas), some information on the servo includes:

- Bidirectional continuous rotation
- 0 to 50 RPM, with a linear response to PWM for easy ramping
- 38 oz-in torque @ 6 V

- Power requirements: 4 to 6 VDC
- Maximum current draw: 140 +/- 50 mA at 6 VDC when operating in no load conditions
- 15 mA when in static state
- Communication: pulse-width modulation
- Dimensions: approx 2.2 x 0.8 x 1.6 in (5.58x 1.9 x 4.06 cm) excluding servo horn
- Operating temperature range: 14 to 122 °F (-10 to +50 °C)

Do not use this servo with an unregulated wall-mount supply. Such power supplies may deliver variable voltage far above the stated voltage.

For timing diagrams and additional information on the servo please refer to the data sheet.

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 servo. 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 servo (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:

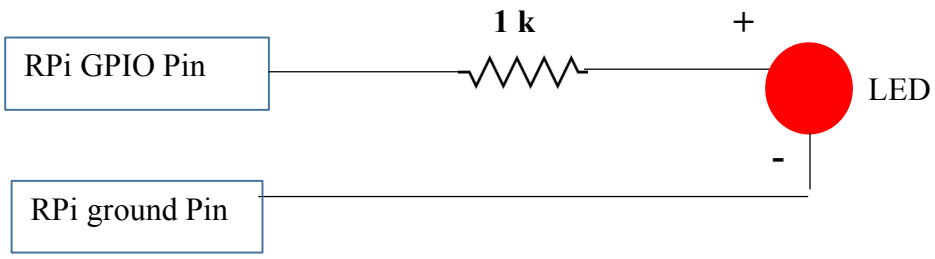
```
p.stop()
```

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. Use the devices available to you, which are present in front of you, especially the regulated power supply, digital voltmeter and the oscilloscope.
3. Please read the data sheet for the servo: There may be an initialization step to perform which could include a small python script

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 should also use the oscilloscope to verify the PWM signal from the output pin. A simple schematic of an LED on a GPIO pin:



Use blink.py to blink the LED on and off over a second. The blink program should allow an integer argument which is used to adjust the blink frequency.

Once you are satisfied that you understand the PWM parameters, connect the Servo motor as indicated in the data sheet for the continuous rotation servo motor.

Stop: Please have the TA check your servo motor connections before proceeding. Note that you can leave the LED in place as a double check on output operation.

Once the circuit is checked is, develop a python code 'pwm_calibrate.py' that can be used during future parts of the lab to calibrate the servo and to provide clean resets of GPIO pins that remain set after failed test code runs. The blink code is a good model for pwm_calibrate.py

Next, develop a python application named 'servo_control.py' to perform the following functions:

- Start the servo as close to stopped as possible
- Range the speed of the servo through ten speed steps 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 servo through ten speed steps 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:

- Begin development by checking PWM pulse dimensions on the oscilloscope. Plan to include several scope screen-shots including idle, and max speed in both directions

- 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 Servos

Select two unused GPIO pins for the attachment for two continuous rotation servos. Attach the servos 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 servos 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: consider the difference between a calibration signal and a stopped state for the servo. What input signal will cause the servo to completely stop?

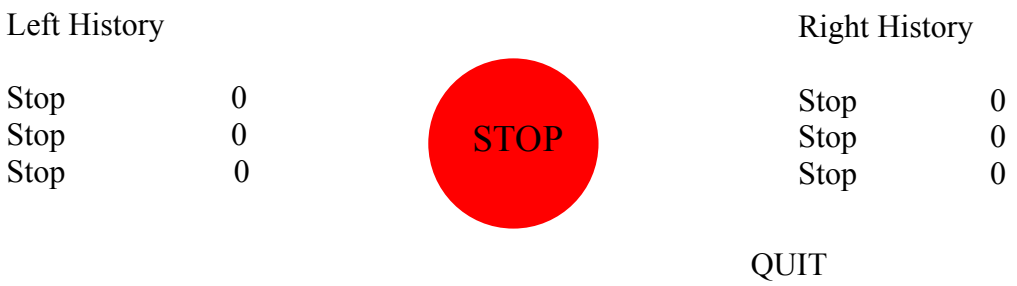
Step3: rolling control

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

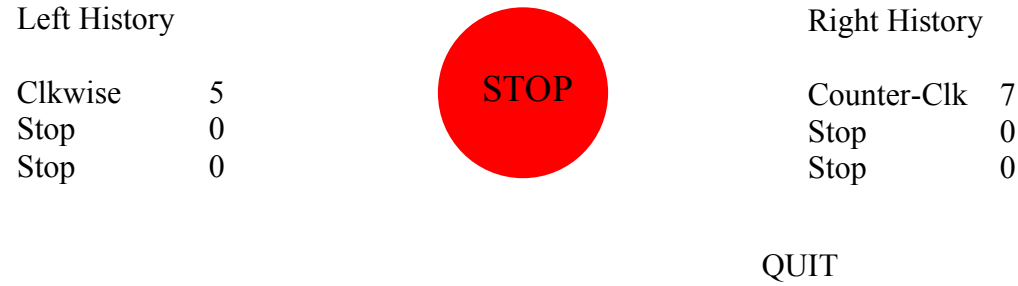
- On the piTFT screen, display direction (clockwise, counter-clockwise, stopped) for each motor
- Implement a single, red 'panic stop' button on the piTFT. If pressed, motors immediately stop and 'panic stop' changes to a green 'resume' button
- Implement a 'quit' button on the piTFT. When hit, quit 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:

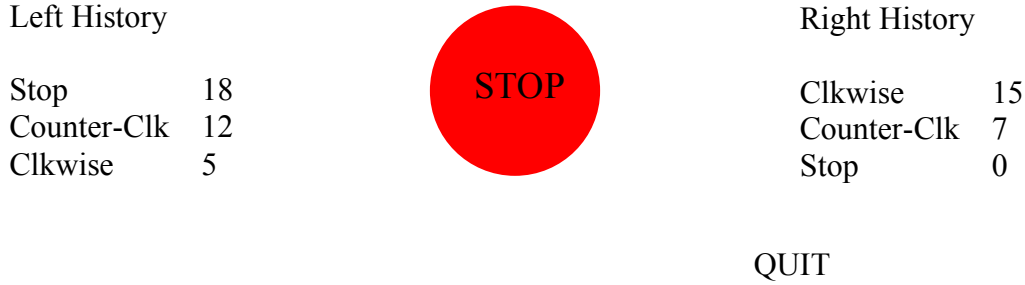
Initial state, motors not running:



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.

Demonstrate the following python applications to the TA before proceeding:

- blink.py
- pwm_calibrate.py
- servo_control.py
- two_wheel.py
- rolling_control.py

Take a backup of your SD card at the end of your demo.

Week 2

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:

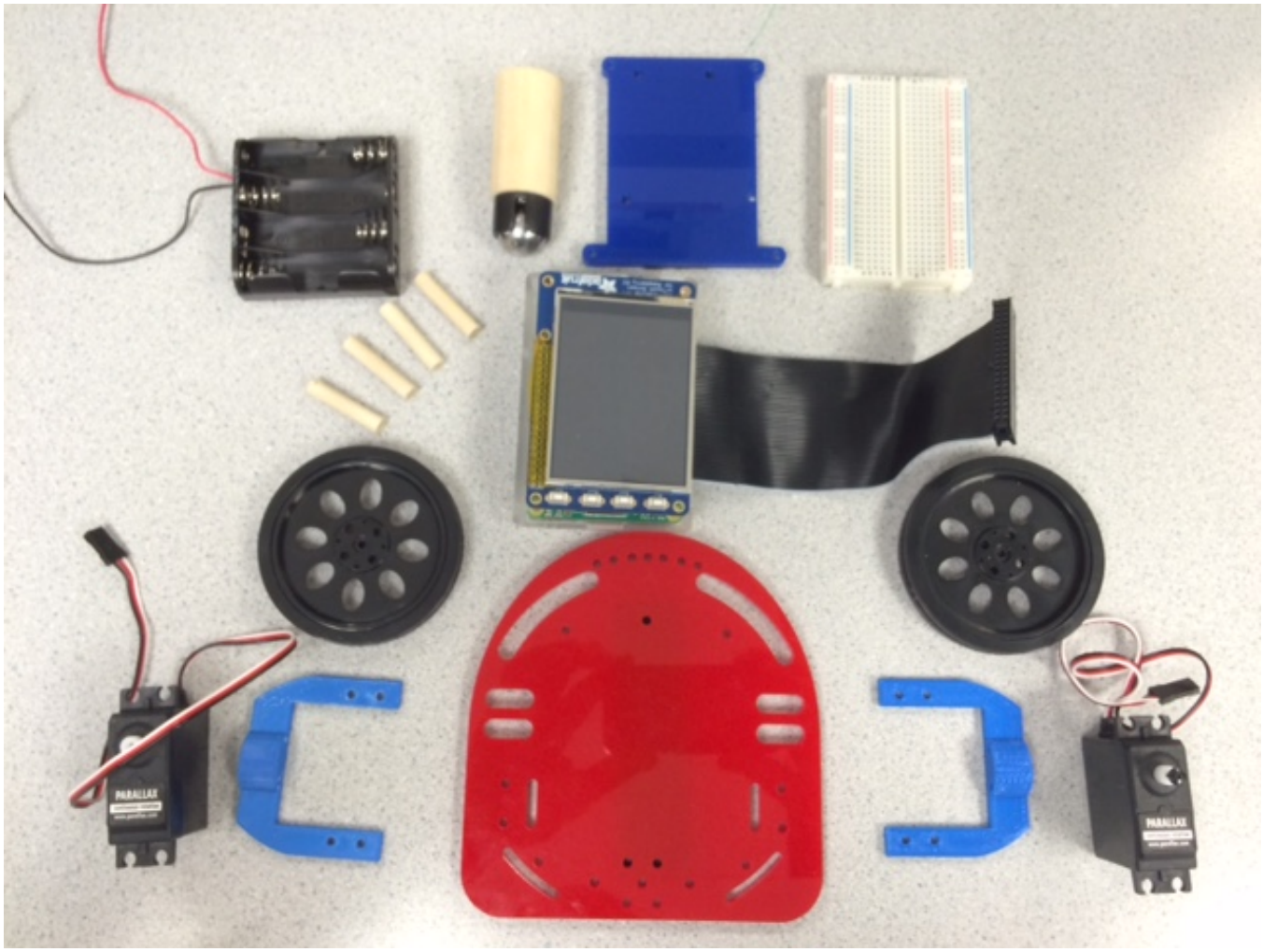
- We will be using sheet metal screws, which are self-tapping screws for metal, wood, or plastic.
- A self-tapping screw forms threads in the material as the screw is installed.
- Do not over tighten the self-tapping screws. Once they are fully installed, tighten the screw just enough for it to grab the material and hold the two pieces together.

¼, 5/8, and ½ inch sheet metal screws

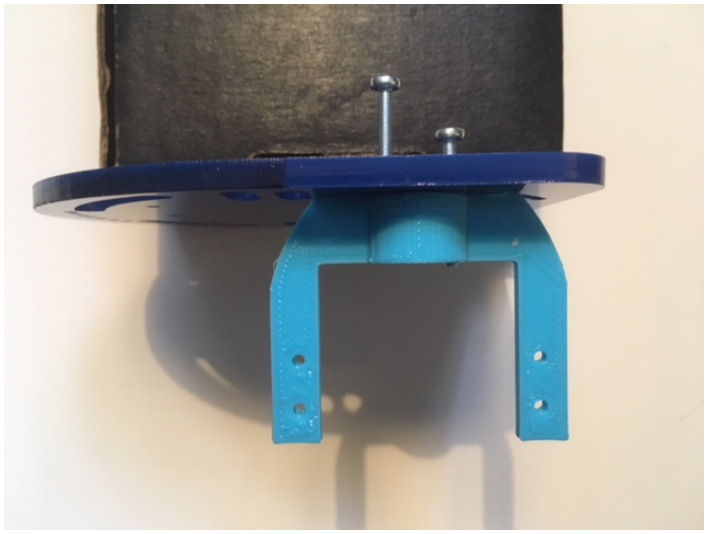
Machine screw, washer and nut

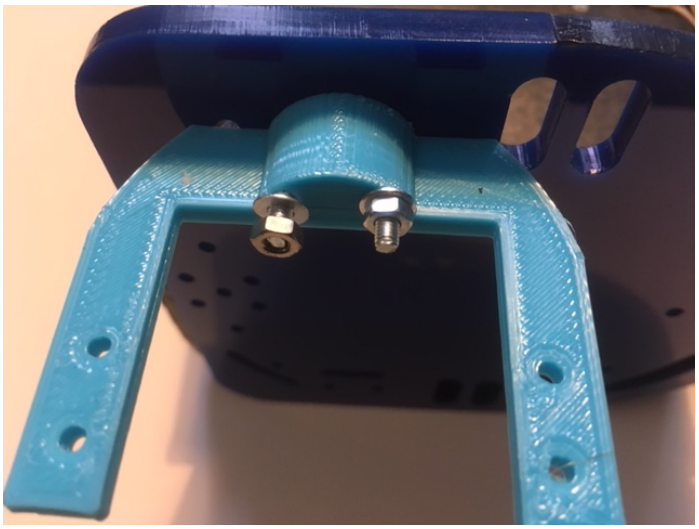
- In some cases, we may require the use of a machine screw. A machine screw is threaded for the attachment of a washer and nut.
- Please talk to a staff member if you would like a demonstration of correct fastening procedures.

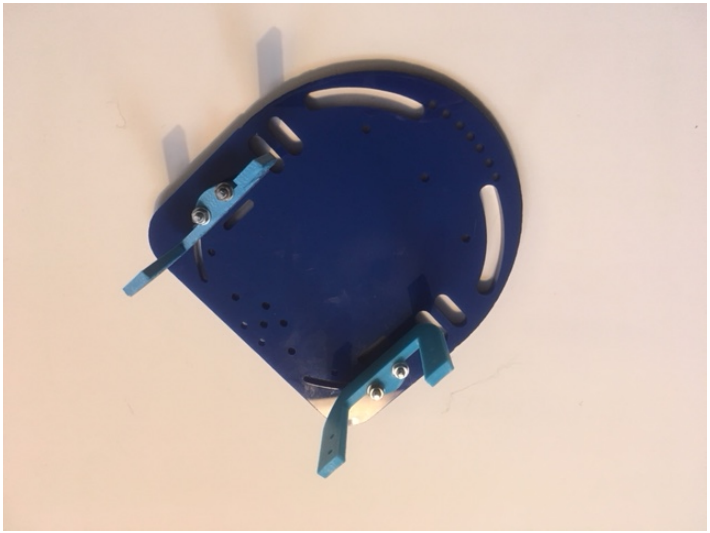
Build a robot frame as follows:



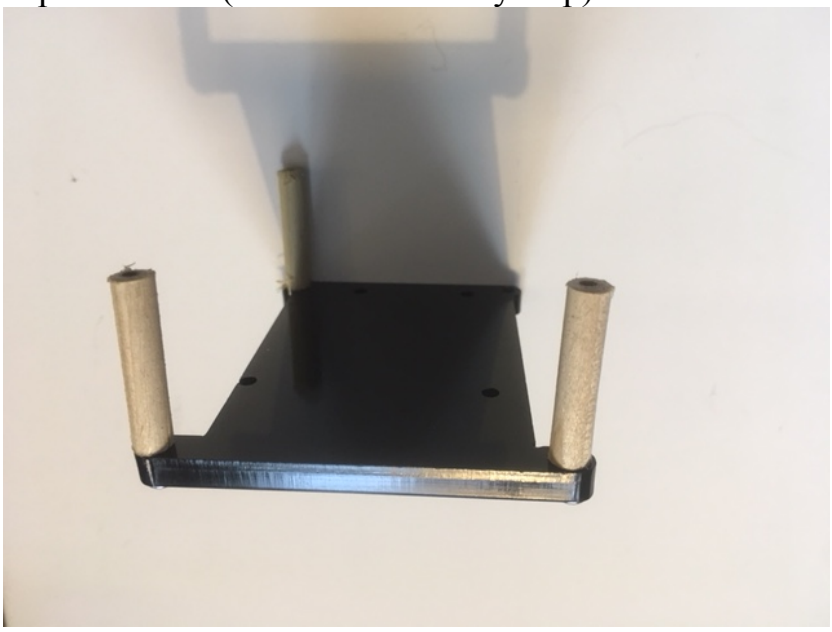
- Attach servo brackets: use 2, #3, 25mm machine screws, washers and nuts for each bracket. Note the position of the washer and nut in the photos.





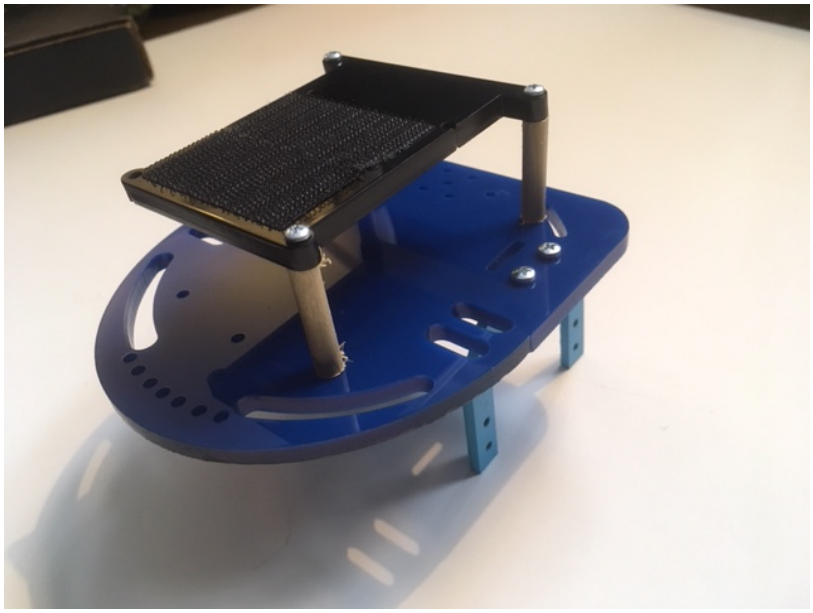


- Attach 3 wooden stand-offs to the upper frame piece using 3, 4 X ½ sheet metal screws. Note: it is convenient to eliminate the forth wooden standoff for ease of fitting the small proto board (in a later assembly step)

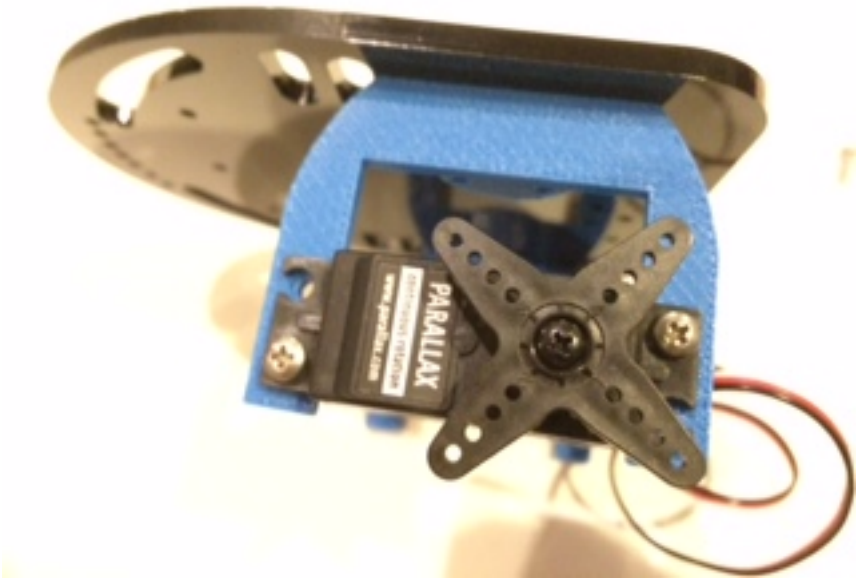


- Attach second level to the base plate (blue part in photo) using 2, 4 X ½ sheet metal screws

  Note: only add screws to two of the standoffs on the bas plate

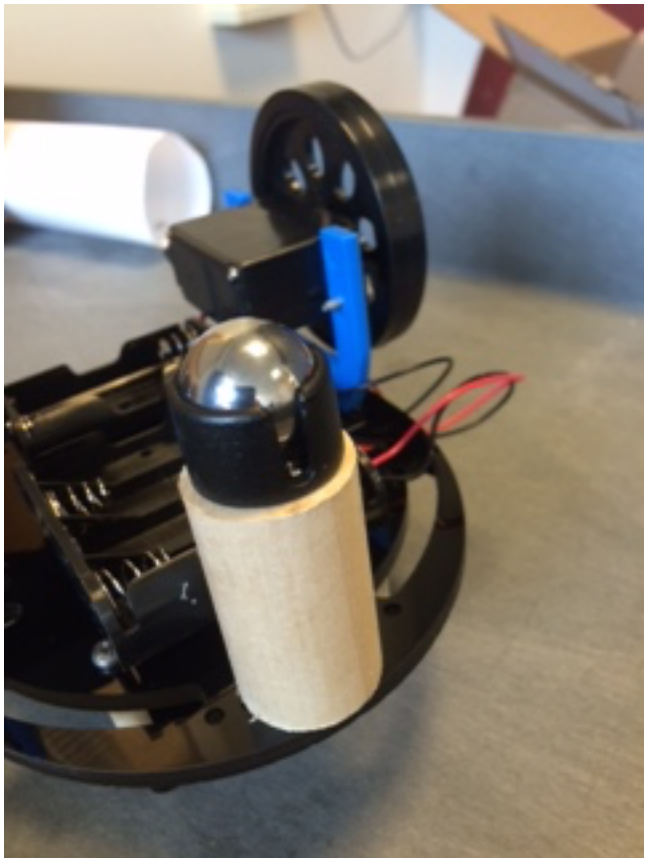


- Connect motors to the bracket: use 2, #3, 10 mm screws, washers and nuts

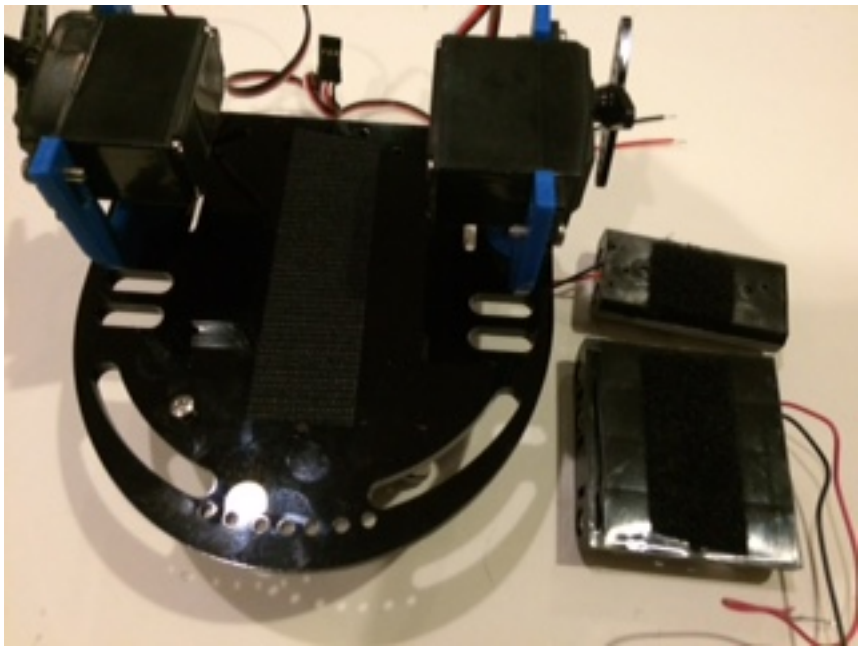


- Install bearing to rear dowel: Pop the ball bearing out of the frame and use the tiny bracket screws in the 2, pre-drilled holes in the dowel. Note: you may have a rear dowel with a pre-installed ball bearing with no assembly necessary.

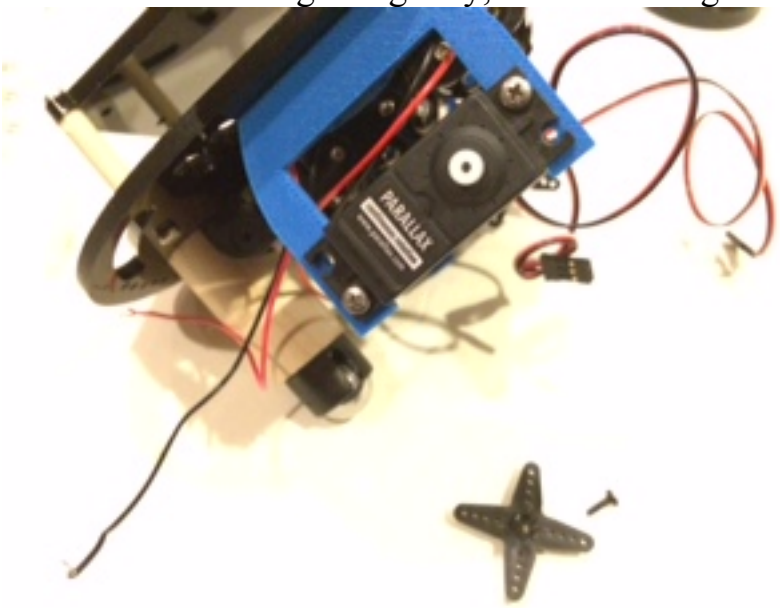
- Attach dowel to the center of the frame using the middle hole along the outer rim: use 1, 4 X ¾ sheet metal screw

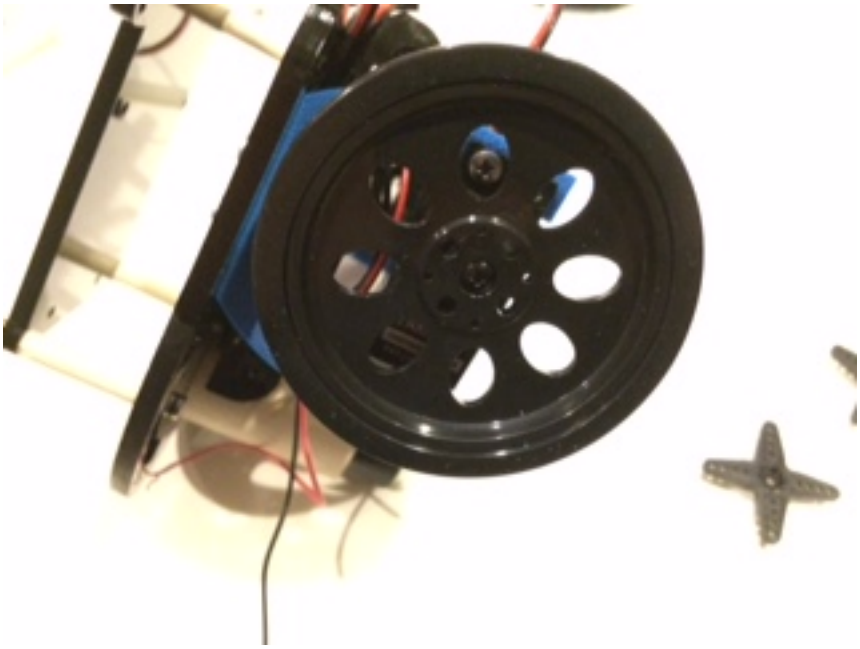


- Attach adhesive Velcro to the underside of the main body and to the back of the battery holders:
    - Attach the ‘loop’ side (fuzzy) of the Velcro to the back of the battery holder (see the photo, below)
    - Attach the ‘hook’ side (stiff) of the Velcro to the underside of the main body (between the 2 motors): see the photo below
    - Note: 4-battery pack fits right against wooden dowel
    - Note: ignore smaller battery pack in the picture below



- Install wheels on the motors
  - Remove center screw from motor (save the screw)
  - Remove the ‘horn’ if installed on the servo.
  - Place the wheel on the spline fitting – note that the wheel hub will fit tightly to the motor shaft even without the center screw IF you have the wheel oriented correctly.
  - Reattach the center screw. Tighten gently, do not over tighten.

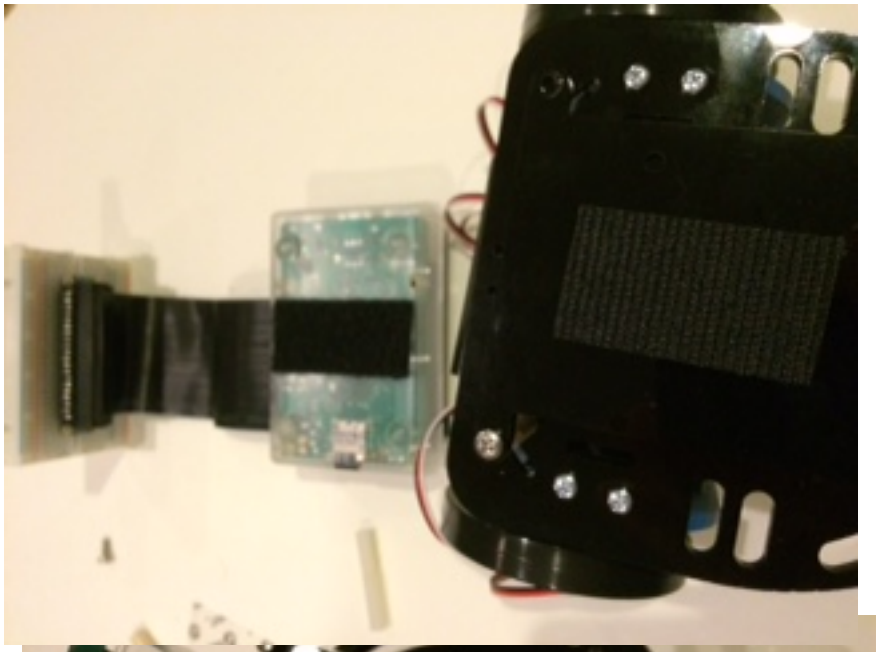




- Apply ‘loop’ (fuzzy)Velcro to the underside of the RPi case
- Apply ‘hook’ (stiff) Velcro (same length) to the top of the upper platform
- Attach the RPi and place the small proto board as shown. Proto board is ‘trapped’ by the wooden standoffs

  Note: remove one standoff to slide the protoboard into position and to wire components.

  Note: Please do NOT remove the ‘sticker’ from the bottom of the proto board. This would make later disassembly difficult. See base plate example in 239 Lab.



Step 5: Test

Wire RPi into motor controls using the small proto-board. Power the motors from the bench power supply. Test correct operation of rolling_control.py and two_wheel.py once the initial assembly is complete.

Review of connections for this step:

- Motor power: external 5V power supply
- Motor control: GPIO pin from RPi
- RPi Power: 5V RPi power supply
- RPi Input/Output: Keyboard, mouse
- RPi display: piTFT and, optionally, monitor

Demonstrate functions to TA before proceeding to the next section

Step 6: More build

- NOTE: When working on connecting the battery holder, do not install batteries until all connections are complete and verified. This will prevent any mishaps due to incorrect connections while the 6 Volt supply is 'live'
- Connect a switch to the 4-AA battery pack to allow the motors to be disconnected from the battery pack. Why do we need to do this?
- Install 4 AA batteries in the battery holder on the underside of the main platform
- Wire this 6 Volt power supply into one side of the + and – rails of the small protoboard
- Rewire motor plus and minus leads to use the 6V battery power.
- Caution: do NOT use this 6V power to power anything on the RPi. Only use this supply to provide power to the servo motors.

Show the connection of the battery pack to the TA before installing batteries

Step 7: Initial battery test

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 all of the functions defined in rolling_control.py from week one of the lab (for on-screen display and emergency control). This includes the left and right history displays, panic stop/resume and quit button.

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.

Step 8: enable WIFI

Using the Linux desktop, hover over the network icon in the upper right of the screen. You should see an option for enabling WIFI; go ahead and enable WIFI. The RPi WIFI module should locate networks; select RedRover.

In order for WIFI to operate correctly on-campus, you will have to register the WIFI adapter to the list of computers associated with your netid. On your laptop, go to this site: https://dnsdb.cit.cornell.edu/register.html and select the option for registering a device WITHOUT a browser. Using the appropriate Linux commands, find the MAC address for the WIFI adapter on the RPi and enter it on the Cornell IT registration page. With this registration, you will eventually be able to ssh into the RPi when it is connected on the WIFI network. Note that registration may take a few minutes to process in the system.

Once WIFI is configured on the RPi, you can unplug the Ethernet cable

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

Step 9: 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:

- 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 has 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.

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