Intro to Robotics II

February 2, 2024

Part I
Mobile systems

0 Build PiCar-X

Charge your batteries and download the updated image for your Raspberry Pi before starting. Both of these processes may take a while.

Follow the instructions in the PiCar-X pamphlet to build the robot car. A few notes during your build:

- Make sure to use the smaller flat-ended servo horn screws, and only tighten until the screw is flush with the servo horn. The use of the longer sharp ones can jam the servo mechanisms, and overtightening can break your servo.
- Be very careful of the camera ribbon cable during assembly. They are fragile and will need to be bent at fairly sharp angles.
- Be careful when popping the plastic rivets into place. They break easily. It would be kind to save your spares to give to classmates in case someone breaks too many.

During assembly, you will need to center the PiCar's servos; to do so, you will install SunFounder's version of the Raspberry Pi operating system EzBlock OS and use their centering script. Later on we will install a standard version of the Raspberry Pi OS and use that for the rest of the course, but for now using EzBlock simplifies the build process. The EzBlock installation process is documented at

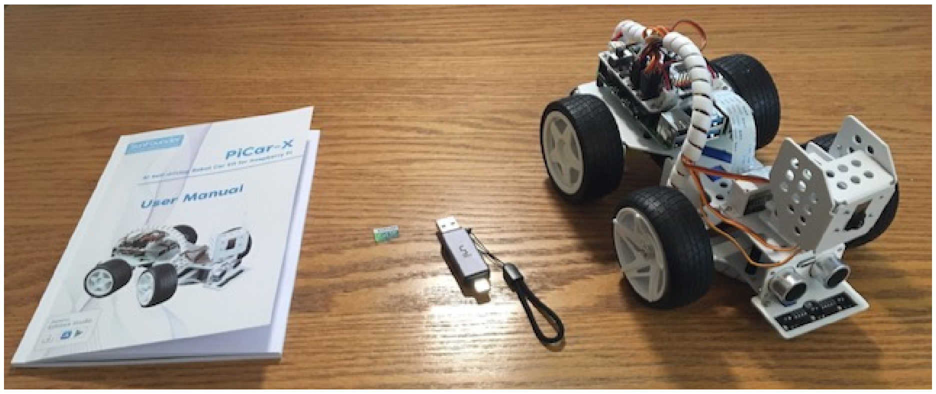
`https://docs.sunfounder.com/projects/ezblock3/en/latest/quick_guide_3.2/install_ezblock_os.html#install-ezblock-os-latest`

Instructions for centering the servos can be found at

`https://docs.sunfounder.com/projects/picar-x/en/latest/ezblock/quick_guide_on_ezblock.html#ezb-servo-adjust`

The batteries we are using are about 5mm longer than a standard 18650 cell, because they incorporate a safety circuit. To accomodate the longer batteries, you will need to modify the battery holder. One of the negative-terminal coils does not compress flat. Clip it off using a pair of heavy-duty clippers. Both batteries should fit inside. If the fit is still too tight the plastic of the battery holder can be clipped near where the terminal coil was to allow for extra flexion.

When turning off the Raspberry Pi, make sure that it is not actively processing anything before flipping the switch. At this point, look to make sure that light patterns are solid or at most steady blinking; in later stages we will set up the system for smooth shutdown commands.



1 Set up and secure your Raspberry Pi

Read through the instructions here all the way through before carrying out the steps in the linked documents – not all steps in the external documents will be followed, and some will be modified.

1. Extract the SD card from the Raspberry Pi and put it back in your SD card reader
2. Follow the Python-specific PiCar setup instructions at

   `https://docs.sunfounder.com/projects/picar-x/en/latest/python/python_start/installing_the_os.html`

   This will **overwrite** the Ezblock image you installed during car setup. Although we will later create a more secure way to sign in to your Pi, please create your own username and password as a temporary measure for better security. It's also strongly suggested that you give your Pi a custom name to avoid accidentally connecting to the wrong Pi. You don't need to enter WiFi information, as we'll be setting up the network configuration in the next steps.
3. Before booting the Raspberry Pi, make sure your PC can still access the contents of the SD card, ejecting and reconnecting if necessary, and follow the instructions at

   `https://www.raspberrypi-spy.co.uk/2017/04/manually-setting-up-pi-wifi-using-wpa_supplicant-conf/`

   to prepare the Pi for a headless configuration. For enabling both home network usage and campus usage your `wpa_supplicant.conf` file should look like

   ```
   country=us
   update_config=1
   ctrl_interface=/var/run/wpa_supplicant

   network={
    ssid="<YOUR_HOME_WIFI_SSID>"
    psk="<YOUR_HOME_WIFI_PASSWORD>"
    priority=1
   }

   network={
    scan_ssid=1
    ssid="robotics"
    key_mgmt=NONE
    priority=2
   }
   ```

   This will allow you to take the picar home if desired.

4. Eject the SD card, insert into the car, boot, and wait 5 minutes. The `wpa_supplicant.conf` file will get moved to the appropriate location, but WiFi will remain disabled and you will not be able to connect to your robot. This is a quirk of using a legacy Raspberry Pi OS. Remove your SD card from the picar and plug it back into your computer. Open `/boot/cmdline.txt` and append

   ```
   systemd.restore_state=0 rfkill.default_state=1
   ```

   to the list of instructions. This will allow WiFi to be auto-enabled on startup and to connect to the networks specified in the wpa file.

5. After the SD card is back in the Pi and the car is booted, if you are on a personal network, use the command prompt to do

   ```
   ping raspberrypi.local
   ```

   to verify that your Pi is on the same network as your computer, replacing `raspberrypi` with your Pi's name if you changed that during setup. If communication is established, use

   ```
   ssh user@raspberrypi.local
   ```

   to connect to the Raspberry Pi, replacing `user` with the username you created in step 2, and using the accompanying password you also created to sign in. For those using the Raspberry Pi on campus, the hostname is tied to the ID number of your Raspberry Pi. Use the following command to ssh into your Raspberry Pi, substituting your Raspberry Pi ID number for `#`:

   ```
   ping PiCar#.engr.oregonstate.edu
   ssh user@PiCar#.engr.oregonstate.edu
   ```

   Remember if swapping between home and school networks to always use the appropriate hostname for the network. If you are not able to make the connection, some trouble-shooting things to try are:

   (a) Make sure that the Raspberry Pi has successfully connected to the network. Log into your network router and look to see if there is a computer named RASPBERRYPI on the network. If there isn't one, repeat the `wpa_supplicant.conf` creation process, double-checking your network name and password.

   (b) If fixing the `wpa_supplicant.conf` file doesn't get the Raspberry Pi onto the WiFi network and you have an ethernet cable available, you can plug the Raspberry Pi directly into the network, and then use the `raspi-config` tool (see below) to configure the wireless network.

   (c) If you can connect, but the password is rejected, someone else on your network may have a Raspberry Pi set up for which they changed the password but left the name at its default value. In this case, log into the router and find the IP addresses of all RASPBERRYPI computers on the network, and then try the IP addresses directly. (In the setup process, we will change the name of the Raspberry Pi).

   (d) If you have connected to a computer named RASPBERRYPI before, you may get a security warning and your computer may refuse to connect to your new computer. In this case, you can use

```
ssh-keyscan $target_host >> ~/.ssh/known_hosts
```

   If you need to do this on a windows machine, there are additional instructions described in various places online.

   (e) Change the name of your Raspberry Pi by using

   ```
   sudo raspi-config
   ```

   and then `System Options > Hostname`.

   (f) Set the localization options on your Raspberry Pi to US. Run

   ```
   sudo raspi-config
   ```

   then scroll down to `en_US.UTF-8`. Enable it by tapping the space bar, then use "return" to move to the next screen.

6. Carry out some basic setup operations for securing your Raspberry Pi. More information on many of these steps can be found in the official Raspberri Pi OS documentation at

   `https://www.raspberrypi.com/documentation/computers/os.html`

   (a) For your robots, I recommend having the `sudo` command require a password and then setting a long timeout on it, by using

   ```
               sudo visudo
   ```

   and then adding a line like

   ```
         Defaults:USER timestamp_timeout=60
   ```

   (where USER is your username, and 60 is the number of minutes I've specified for the timeout)

   (b) Make sure your Pi software is up-to-date by using

   ```
   sudo apt update
   sudo apt full-upgrade
   ```

   (c) Make sure to set up your SSH installation to automatically update itself by setting up a cron job. First open the cron table for editing

   ```
   crontab -e
   ```

   Once the cron table is open in an editor, add the line

   ```
   @reboot sudo apt install openssh-server
   ```

   This will attempt to update your SSH installation whenever your Pi is rebooted.

   (d) If you are on a Windows machine, use

   `https://www.ssh.com/ssh/putty/windows/puttygen`

   to generate your SSH keys, and then follow the instructions at

   `https://pimylifeup.com/raspberry-pi-ssh-keys/`

to save a private key to your device and create a public key on your Raspberry Pi. With this security key setup, you can safely disable the initial password authentication using steps in the previous link. From now on, you will SSH into your Pi by entering your host name and private key location into PuTTY (or your chosen SSH client). If you are on a Unix machine you will set up your SSH keypair using the "Passwordless SSH Access" steps listed at

```
https://www.raspberrypi.com/documentation/computers/remote-
access.html
```

(e) Use `apt` to install `ufw` with

```
sudo apt install ufw
```

and then follow steps 2-4 of

```
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-
firewall-with-ufw-on-ubuntu-20-04
```

to turn on the `ufw` firewall. Then, set the options `sudo ufw allow ssh` and `sudo ufw limit ssh/tcp`.

(f) Install `fail2ban` as per the instructions at

```
https://www.digitalocean.com/community/tutorials/how-to-protect-
ssh-with-fail2ban-on-ubuntu-20-04
```

to block brute force attempts at your passwords. You won't need to make a `jail.local` file, it just overrides `jail.conf` if you are unsatisfied with any of the settings therein. For this class, just make sure `fail2ban` is installed and the service is active.

Deliverables

1. Reflection questions:

   (a) What did you learn while completing the tasks in this lesson?

   (b) If you see any shortfalls with the approach we have taken above, please suggest improvements.

To receive full credit for the weekly reflection assignments, your answers should meaningfully engage with the question – bullet-point lists naming the topics covered in a given week are not sufficient responses. Tell me something about what your starting point for knowledge on the week's topics was, or how it relates to work you've done in the past. Then tell me something specific that you now know about that you didn't before you worked on this assignment (or if this is all stuff you have extensive knowledge of, turn the "starting point" into a story about using it.

2 Motor commands

Once your robot is assembled and you have its computer set up, the next step is to get a handle on the interfaces between the coding you can do in Python and the physical hardware. In this lesson, we will

1. Install SunFounder's python code (and dependencies) for robot control
2. Create a copy of `picarx.py` with modifications to enable offline testing and improve the system performance
3. Create a "shadow" version of the `PiCar` package to enable offline testing
4. Write some basic maneuver functions to demonstrate control over the motors

2.1 Gather code pieces

1. On your Raspberry Pi, follow the instructions located at

   `https://docs.sunfounder.com/projects/picar-x/en/latest/python/`
   `python_start/install_all_modules.html`

   to set up code and dependencies (Do this in your own account, not the 'pi' account). *Note: A couple packages (pygame and one or two others) might not install correctly. You don't need to worry about those, just continue on if you get those errors.*

2. Note that the `install.py` script takes a while. It is recommended that you run this script on wired power, rather than using batteries.

3. Note that depending on how the installation process goes, you might have to install espeak separately through the following command

   ```
   sudo apt install espeak
   ```

4. If you installed the lite version of Raspberry Pi OS, you will also need to run

   ```
   sudo apt install python3-smbus
   ```

   to install one more missing dependency.

5. After successfully installing the modules, make sure to enable the I2C and camera interfaces by following the instructions located at

   `https://docs.sunfounder.com/projects/picrawler-cn/en/latest/python/`
   `python_start/enable_i2c_camera.html`

2.2 Make the robot move

Before really working with the robot, we should make some improvements to the stock code. I know you're here to play with robots, so let's see the car move before we go further. The PiCar-X manual describes the provided demo code; not all of it works as well as advertised. Try running the programs and see what happens. In order to properly calibrate the servo angles and motor directions, you can use the calibration helper by following the instructions at

`https://docs.sunfounder.com/projects/picar-x/en/latest/python/python_calibrate.html`

2.3 Configure your Raspberry Pi to connect to a GitHub account (or other Git repository)

1. Create an account at GitHub.com. If you have a GitHub account already, you can use it. If you have a different Git setup that you prefer to use, go ahead and use it, and modify the other setup instructions as needed.

2. Create a new Git repository in your account named `RobotSystems`. Don't make a `README` file for it, as that file can cause headaches during setup. We'll make one later.

3. Set up SSH keys on your GitHub account.

   (a) Go to the Settings page for your account (Use the dropdown menu in the top right corner of the page; Settings is near the bottom of the menu.)

   (b) Go to "SSH and GPG keys" in the list of settings panes at the left of the screen

   (c) Open the "guide to generating SSH keys" link **in a new tab**.

   (d) SSH into your Raspberry Pi, and then follow the instructions to create an SSH key **on your Raspberry Pi** and add that key to your GitHub account. This will be similar to the process you used to create the SSH key that lets you log into your Raspberry Pi from your computer. On the "Generating a new SSH key and adding it to the ssh-agent" page, make sure you are reading the "Linux" instructions (because you are performing operations on the Raspberry Pi).

   (e) Set a non-empty passphrase for your SSH key. This means that your key is encrypted on your Raspberry Pi, and cannot trivially be used by others if your robot is lost or stolen. (This precaution is more important on something like a Raspberry Pi than on a computer with disk encryption enabled.)

   If you follow the link for "Working with SSH key passphrases", you can follow the instructions under "Auto-launching ssh-agent on Git for Windows" to modify the `.profile` file on your Raspberry Pi so that you will be asked for this passphrase each time you boot the Raspberry Pi, and won't be asked for it again when you perform Git operations. If you are running `zsh`, there are various solutions online, e.g.,

   `https://www.vinc17.net/unix/index.en.html#zsh-ssh-utils`

   that offer more elegant handling of SSH keys, such as only prompting you for a passphrase when you actually use the keys instead of on startup.

(f) On the “Adding a new SSH key to your GitHub account” page, the fact that you are SSH’d into the computer on which you are generating keys makes Step 1 (copying the ssh key into your clipboard) a bit trickier than they otherwise would be. The easiest way to make this work will probably be to run

```
nano .ssh/id_ed25519.pub
```

and then copy the contents of the file out of the terminal.

(g) If you left the original Settings page open in another tab, you can then follow the remaining steps on the “Adding a new SSH key to your GitHub account” page.

2.4 Set up coding environment

Make a copy of the existing `picar-x` directory and link it with your repo

1. Make a local `RobotSystems` repository onto your Pi

```
cd ~
mkdir RobotSystems
```

2. Copy all of the `picar-x` code into the `RobotSystems` repository

```
cp -a picar-x/. RobotSystems/
```

3. Copying all of the folder contents means that the new folder will inherit git settings for sunfounder’s picar repo. Configure them for your personal repo and push to GitHub

```
git branch main
git checkout main
git remote set-url origin git@github.com:<YourGithubUsername>/RobotSystems
git add .
git commit -m "Adding PiCar Files"
git push --set-upstream origin main
```

4. Verify on GitHub that your repo now contains all the `picar-x` files and folders. Edit the `README` to reflect the personal nature of the repo

5. Use `sudo shutdown now` to turn off your car and save your batteries.

6. Clone your Git repository somewhere onto your desktop computer.

2.5 Create basic infrastructure

You will be setting up a remote testing environment, designed to let you test robot code on your desktop before pushing to the PiCar. These steps should be carried out on your desktop computer.

1. Copy `picarx.py` into a new file `picarx_improved.py`

2. Create an empty Python script named `sim_robot_hat`

3. Replace the opening lines of `from robot_hat import ...` in `picarx_improved.py` with

   ```
   import time
   import os
   import math
   try:
       from robot_hat import Pin, ADC, PWM, Servo, fileDB
       from robot_hat import Grayscale_Module, Ultrasonic
       from robot_hat.utils import reset_mcu, run_command
   except ImportError:
       from sim_robot_hat import Pin, ADC, PWM, Servo, fileDB
       from sim_robot_hat import Grayscale_Module, Ultrasonic
       from sim_robot_hat import reset_mcu, run_command
   import logging
   import atexit

   reset_mcu()
   time.sleep(0.2)
   ```

   This lets you define `sim` functions, which shadow PiCar hardware calls when doing testing on your desktop.

4. Open a Python shell, and enter `import picarx_improved.py`. This should return an error along the lines of

   ```
   name 'Pin' is not defined
   ```

5. Find the code in your copy of the `robot_hat` package that implements the missing class, then copy that class to your `sim_robot_hat` file. Methods that don't return an output (e.g. methods that send messages over I2C to the Pi pins) can be implemented as a simple `pass`, other methods should return an output of the appropriate type. It might be a good idea to clone the robot-hat code somewhere onto your working PC so that you have convenient access to the robot-hat classes outside of the Picar:

   `https://github.com/sunfounder/robot-hat/tree/main`

6. Repeat the above steps until you have provisioned `sim_robot_hat` with all of the classes that `picarx_improved.py` is expecting to see.

2.6 Logging

It is often helpful to have your code write out its progress to the command line, especially when debugging. The most basic way to do this is to insert `print` commands at various points in your code. This approach, however, tends to make your code hard to manage – the operational logic of what you are doing becomes interspersed with many `print` commands, and turning those commands off and on again requires manual commenting and uncommenting.

A better way of having your code display outputs is to use the Python `logging` package. For basic usage:

1. Place

   ```
   import logging
   ```

   at the top of your file.

2. After your package imports, put

   ```
   logging_format = "%(asctime)s: %(message)s"
   logging.basicConfig(format=logging_format, level=logging.INFO,
                       datefmt="%H:%M:%S")
   ```

   to set up your basic logging format. You can then set the logging level to DEBUG with

   ```
   logging.getLogger().setLevel(logging.DEBUG)
   ```

   to have all logging commands at the "DEBUG" level print out to the command line. If you comment out this line, then all DEBUG-level logging messages will be suppressed.

3. At points in your code where you want a message printed to the command line, insert a line

   ```
   logging.debug(message)
   ```

   (where "`message`" is the text you want displayed).

The documentation for `logging` describes more advanced usages, such as setting different levels of logging that you can toggle individually, or setting up the `message` to include current values of system variables.

The logging package fixes one problem from naive `print`-logging, but still leaves the logging messages interspersed with your operational code. An even better approach is to use the `logdecorator` package:

1. Place

   ```
   from logdecorator import log_on_start, log_on_end, log_on_error
   ```

   at the top of your file after `import logging`.

2. Immediately before the `def` line for each function or method that you want to display logging information, insert lines like

   ```
   @log_on_start(logging.DEBUG, "Message when function starts")
   @log_on_error(logging.DEBUG, "Message when function encounters an error
       before completing")
   @log_on_end(logging.DEBUG, "Message when function ends successfully")
   ```

   These messages can be set to include any inputs provided to the function, using Python formatting notation. For example, if one of the inputs to the function is a string `name`, this string can be included in the logging message as in

   ```
   @log_on_start(logging.DEBUG, "{name:s}: Message when function starts")
   ```

   The `@log_on_end` decorator can also display the result returned by the function, e.g.,

```
@log_on_end(logging.DEBUG, "Message when function ends successfully: {result
    !r}")
```

Note that this logging only displays information as you go into and out of functions, and does not say anything about progress through a function. If you follow good code-abstraction processes and write small functions that carry out well-defined tasks, this will still give you a fine-grained understanding of how your program proceeds through its code.

2.7 Improve the PiCar-X code

1. As you may have noticed when running the demo code, the stock controller can leave the motors running if you terminate a program while they are on. Use the `atexit` Python module to make sure that the motors are set to zero speed when any program incorporating the `picarx_improved.py` code is terminated. A good way to do this is to register the `stop` function at the end of the class initialization.
2. The stock code takes the commanded motor speeds and scales them (probably to prevent the programmer from commanding too slow a speed, which would not provide enough motor power to overcome friction in the system). Find the code that implements this scaling and remove it so that you get actual speed control.
3. The `forward` and `backward` functions run the motors at different speeds using a linear scaling on the steering angle. We can get turning motion that skids less by improving this speed scaling. Use an ackerman steering approximation to write a function that defines the wheel speeds as a sinusoid function of steering angle, and modify the `forward` and `backward` functions to implement this relationship.
4. Calibrate your steering: Write a function that imports `picarx_improved.py`, then commands the system to drive forward at zero steering angle for a short time. Based on how far left or right the car pulls over this motion, modify the steering calibration angle in `picarx_improved.py`.
5. To make future troubleshooting simpler, add a `logging.debug(<message>)` line in functions where a motor setting is changed with a useful message letting you know what exactly is being done on the robot.

2.8 Maneuvering

1. Write a set of functions (either in `picarx_improved.py` or in a separate file that imports `picarx_improved.py`) that move the car via discrete actions:
   (a) Forward and backward in straight lines or with different steering angles
   (b) Parallel-parking left and right
   (c) Three-point turning (K-turning) with initial turn to the left or right

   (If we had encoders on the wheels, we could explicitly set the amount by which to turn the wheels. These robots don't, so you'll need to approximate the desired distance to travel via speed and length of time for which you turn on the motors. You might consider measuring your car's speed at 100% commanded power for use as a reference).

2. Write a script that runs a while loop in which each iteration

   (a) Asks the user for keyboard input via the `input` function
   (b) Maps the keyboard input to one of the maneuver functions
   (c) Executes the selected maneuver

   (For good style, set up the input parsing so that there is an input that cleanly breaks the while loop)

Deliverables

1. Reflection questions:

   (a) What did you learn while completing the tasks in this lesson?
   (b) What other opportunities for improving the `picarx.py` code did you observe?
   (c) If you see any shortfalls with the approach we have taken above, please suggest improvements.

3 Sensors and Control

Now that we've got basic motor control handled, we can start using sensors to provide control inputs

The code in `adc.py` provides code for reading the ADC (analog-to-digital converter) pins on the breakout board, and `grayscale_module.py` provides an example of collecting data from the ground-scanning photosensors attached to the ADC pins. The `examples/minecart_plus.py` script uses these values to control the steering angle of the car, but it is not particularly robust to material and lighting conditions.

In this lesson, we create Python classes to handle three aspects of the sensor-to-control process

1. Sensing the darkness of the ground below the robot
2. Interpreting data from the photosensors into a description of the current state of the robot
3. Controlling the steering angle based on the interpretation of the photosensors

3.1 Sensing

The sensor class should incorporate the following features:

1. The `__init__` method should set up the ADC structures as attributes using the `self.` syntax
2. Your sensor-reading method should poll the three ADC structures and put their outputs into a list which it returns

3.2 Interpretation

The interpreter class should incorporate the following features:

1. The `__init__` method should take in arguments (with default values) for both the sensitivity (how different "dark" and "light" readings are expected to be) and polarity (is the line the system is following darker or lighter than the surrounding floor?)
2. The main processing method take an input argument of the same format as the output of the sensor method. It should then identify if there is a sharp change between two adjacent sensor values (indicative of an edge), and then using the edge location and sign to determine both whether the system is to the left or right of being centered, and whether it is very off-center or only slightly off-center. Make this function robust to different lighting conditions, and with an option to have the "target" darker or lighter than the surrounding floor.
3. The output method should return the position of the robot relative to the line as a value on the interval $[-1, 1]$, with positive values being to the left of the robot.

3.3 Controller

The controller class should incorporate the following features:

1. The `__init__` method should take in an argument (with a default value) for the scaling factor between the interpreted offset from the line and the angle by which to steer.

2. The main control method should call the steering-servo method from your car class so that it turns the car toward the line. It should also return the commanded steering angle.

3.4 Sensor-control integration

Write a function that combines the sensor, interpreter, and controller functions in a loop so that the wheels automatically steer left or right as you move the car right and left over a dark line in the floor. You may need to adjust the sensitivity and polarity values in your interpreter function. Once automatic steering is working, add a "move forward" command to your script to make the car drive along the line with automatic steering. You may need to adjust the magnitude of the steering angle and the delay in your loop to make this motion smooth and robust.

3.5 Camera-based driving

Sensing the line via the three photocells is essentially using a three-pixel camera to look for the line to follow. The robots have a camera with a much higher resolution. Write sensor-interpreter-controller functions that use the camera to identify the line location and drive along it.

1. Set up VNC:
   (a) Follow sunfounder's instructions to set up VNC at

      `https://docs.sunfounder.com/projects/picar-x/en/latest/python/python_start/remote_windows.html#remote-desktop`

   (b) Secure VNC by requiring user login on connect:

      ```
      sudo raspi-config > "System Options" > "S5 Boot / Auto Login" > "B4
         Desktop Autologin"
      ```

      Because all users are permitted to log in, ensure that the `pi` user has a strong password. You should still be logging into the Raspberry Pi with your own user account.

   (c) Allow VNC connections through the UFW firewall:

      ```
      sudo ufw allow VNC
      ```

   (d) Follow step 11 from

      `https://desertbot.io/blog/headless-raspberry-pi-4-remote-desktop-vnc-setup`

      to be able to see the desktop. Some users report that it's necessary to set the resolution to the maximum allowed.

   (e) If you are still getting the "cannot currently show the desktop" message on RealVNC after changing the resolution and changing the default boot method to desktop autologin, you might need to add some lines to the boot configuration file as in

      `https://www.youtube.com/watch?v=hA9r13ZUS08`

   At this point, you should be able to connect to the Raspberry Pi with VNC, and run camera code.

2. The picamera documentation

`https://picamera.readthedocs.io/en/release-1.13/recipes1.html`

has the logic needed for speaking with the camera.

3. A good example on lane-following with OpenCV is at

   `https://const-toporov.medium.com/line-following-robot-with-opencv-and-contour-based-approach-417b90f2c298`

Deliverables

1. Code review from "Car week 2" partner
2. Code review of "Car week 2" partner's work
3. Reflection questions:
   (a) What did you learn while completing the tasks in this lesson?
   (b) If we spent more time "hardening" the control loop in this lesson, what feature or capability would you add?
   (c) Why does the use of VNC add a security vulnerability? How would you fix or improve the issue?

4 Simultaneity

It is often useful to decouple the various operations that your robot performs, so that sensing, interpretation, and control run independently and in parallel. This notion of *concurrency* can be implemented via

- Multitasking, in which a single processor rapidly switches between the different functions it is executing, and
- Multiprocessing, in which the functions are assigned to dedicated processors.

Multitasking can be further characterized as being *cooperative* or *pre-emptive*, respectively based on whether the time-sharing of the processor is determined by the functions themselves, or is allocated externally by the computer's operating system. Pre-emptive multitasking is also called *threading*. More information on these concepts can be found at

`https://realpython.com/python-concurrency/`

For pre-emptive multitasking and multiprocessing, there is a risk that multiple threads and processes will attempt to operate on the same data structures at the same time. At a low level, consequences can include a task attempting to read a data from a location in the middle of another task writing to the location, and getting a mix of the bits in old values and new values, or two tasks attempting to write data to the same location at the same time, and ending up storing a mix of bits corresponding to the two values.

At a higher level, interleaving the execution of tasks can cause other problems even if bit-mixing is avoided. For example, the operation `x=x+1` in many languages does not mean "increment the value in `x` by `1`" but instead means "read the value in `x`, add `1` to it, then write the resulting value into `x`". If two threads or processes attempt to execute this operation at the same time, the sub-operations may become interleaved such that the second thread reads the value in `x` before the first has incremented it. In this case both threads will write out a value to `x` that is `1` more than its *original* value, and an increment will be lost.

These difficulties with concurrency can be addressed by

1. Designing the system to minimize the opportunity for conflicts (e.g., by avoiding situations in which two operations write to the same location) and to be robust in the case that one such error does occur. This tends to be easier when the information being passed through the system is an approximation of an analog signal rather than an encoded text. (See also "Gray code" for an example of handling concurrency problems at a hardware level).

2. Ensuring that operations are *atomic* (not interruptible). Different programming languages handle atomicity at different levels. Some languages guarantee that reading and writing simple data types such as integers and floats is atomic, so that bit-mixing is not a worry, but do not guarantee that writing values to an array will not be interrupted partway through by a read operation. Python goes further, making read and write operations to many array types (including lists) atomic.

   Atomicity for higher-level operations generally needs to be specified by the programmer (to ensure that it is applied to operations for which interruption would be problematic are made atomic, while allowing other operations to be interruptible to keep the program flowing).

Setting up the *locks* that ensure atomicity can quickly become complicated. If this were a CS course, we'd stop here and spend a good chunk of time going over various approaches to locking. Many of these approaches, however, are related to passing along discrete messages. By focusing our attention on tasks that broadcast their most-recent estimates of system states on dedicated channels, we can sidestep most of this extra complexity and leave it on a need-to-know basis.

A second effect to be careful of in pre-emptive multi-tasking is when the task scheduler interrupts a set of nominally simultaneous interactions with an external system. For example, if a program polls several sensors in succession to get a snapshot of the external world at a given time, the sensor measurements will be slightly offset in time from each other, which can cause "rolling shutter" distortion. This distortion will be amplified if the task scheduler switches to a different task in the middle of cycling through the sensors, increasing the delay between sensor readings. *Cooperative multitasking* can help to mitigate these effects, by having the programmer specify places where it is safe for threads to switch out (at the cost of requiring the programmer to think about and specify these points. Cooperative multitasking is not directly covered in this lesson (but example code of how it could have been used here will be made available in the following week).

For this lesson, you will assign the sensor, interpretation, and steering-control functions to independent threads of the program, so that they can run at independent rates. In doing so, we will implement *consumer-producer* functions passing *messages* via *busses.*

4.1 Busses

When we assign functions to individual threads, we generally need to provide a means for them to pass information between threads, e.g., so that the interpreter can see the most recently reported reading from the sensors, and the controller can see the most recent interpretation of the system state.

A basic means of passing messages between threads or processes is to create "message busses" that processes can read from or write to. The messages on these busses can be "state updates" that act as "one-way broadcasts", in that reading the message does not remove it from the bus, or "queued", in that messages are "cleared" as they are read.

For this class, we will use "broadcast" messages. Conceptually, this is similar to how global variables can be used to pass information between functions, but with a bit more structure

---

Define a simple Python class to serve as your bus structure. It should have:

1. A "message" attribute to store values
2. A "write" method that sets the message
3. A "read" method that returns the message

4.2 Consumer-producers

The core elements of a robot control program are operations that carry out tasks such as polling sensors, interpreting data, and controlling motors. When these processes exchange data via busses, we can think of them as

- Producers, writing newly-created data to busses;
- Consumer-producers, reading information from some busses, processing that data, and writing the output to other busses; and
- Consumers, reading information from busses and acting on it without publishing data to busses.

These categories closely correspond to our previous notions of sensor, interpretation, and control functions.

Define producer, consumer-producer, or consumer functions for the sensor, interpretation and control processes in the previous lesson.

1. Each consumer/consumer-producer/producer should be defined as a function that takes instances of your bus class and a delay time as arguments
2. Each consumer/consumer-producer/producer function should contain a while loop
3. The sensor, interpretation, or control function should be executed inside the loop function, with data read from or written to bus classes as appropriate
4. The loop should `sleep` by the delay amount once each cycle

4.3 Concurrent execution

Once the busses and consumer-producers have been created, we can have the consumer-producer functions execute concurrently. Some basic steps to get concurrent execution up and running are:

1. Import the `concurrent.futures` module with:

   ```
   import concurrent.futures
   ```

   and tell the system to run your system components concurrently. For example, to run your `sensor_function` with inputs of the `sensor_values_bus` to write data to and the `sensor_delay` to set the polling rate together with your `interpreter_function`, you can run

   ```
   with concurrent.futures.ThreadPoolExecutor(max_workers=2) as executor:
           eSensor = executor.submit(sensor_function, sensor_values_bus,
               sensor_delay)
           eInterpreter = executor.submit(interpreter_function,
               sensor_values_bus, interpreter_bus,interpreter_delay)
   ```

   You will place a similar block of code setting up the three sensor, interpreter, and control threads where you originally ran the full line-following functionality. Make sure you allocate enough workers. Note that to cancel execution of a program running under `concurrent.futures` you will need to hit `ctrl-C` multiple times. Depending on your simulation environment setup, Ctrl-C might not kill the control process when testing code off the picar. Ctrl-C presses should work when the code is run on the picar. Because of this threading setup, any errors thrown by the functions will not be displayed in the terminal; this problem can be worked around by adding a line

```
eSensor.result()
```

outside of the `with` block.

Set up concurrent futures for each of your consumer-producer functions.

2. We want to ensure that writing and reading bus messages is properly locked. The suggested locking code here is an "writer-priority read-write lock", which means that:

   (a) Only one thread is allowed to write the bus message at any given time, and may not start writing while any other thread is reading the message

   (b) Any number of threads may read the bus message at any given time, but may not start reading while a thread is writing the message

   (c) Write operations are given priority over read operations

   A physical analogy of this situation is a whiteboard at the front of a room, with instructors taking turns writing information on the board, and students able to read the information on the board whenever an instructor is not blocking their view. Writer-priority means that the students may have to wait for information, but it will always be the most up-to-date information (see `https://en.wikipedia.org/wiki/Readers-writer_lock` for more details).

Set up locking for your busses.

   (a) Import and implement concurrent futures

   (b) Install the `readerwriterlock` Python module by entering

   ```
   python3 -m pip install -U readerwriterlock
   ```

   at the command line.

   (c) Put

   ```
   from readerwriterlock import rwlock
   ```

   in your header

   (d) Put

   ```
   self.lock = rwlock.RWLockWriteD()
   ```

   inside your `__init__` method for the `bus` class

   (e) Use

   ```
           with self.lock.gen_wlock():
               self.message = message
   ```

   inside your "write" method, and

   ```
           with self.lock.gen_rlock():
               message = self.message
   ```

   inside your "read" method.

Deliverables

1. Code review from “Car week 3” partner
2. Code review of “Car week 3” partner’s work
3. Reflection questions:
   (a) What did you learn while completing the tasks in this lesson?
   (b) With the architecture that you’ve created, what is the next feature that you would add to the robot control stack?

5 Multimodal control

The threaded architecture in the previous lesson lets us execute multiple processes concurrently. The assignment, however, only really had a single task flow, and as such didn't really need to be threaded. In this lesson, we will set up a system that uses threads to separately poll two different sensors and combine their information for the control system.

5.1 RossROS

On github at `https://github.com/rlhatton/RossROS`, you can find a file `RossROS.py` that contains bus and consumer-producer classes (reference implementations of the structures discussed in the last lesson), alongside an example of a concurrent execution architecture `rr_demo.py`. Further documentation can be found at `rossros.org`.

1. Download `RossRos.py` and reimplement your concurrent line-follower script using the classes therein.

2. As you do so, use the `Timer` class from `RossROS.py` to control the run-time of the script. `Timer` instances are Producers that change their output bus value to "true" after a set amount of time; if you set the output bus to the termination bus that it and the other Consumer-Producers are reading, all of the threads will shut down when the `Timer` reaches the set time.

5.2 Concurrent control

The consumer-producer/busses framework allows us to run multiple control loops at the same time. For example, we can have the car use the photosensors to follow a line, while having the ultrasonic sensors stop the robot if it comes too close to an obstacle on the line.

1. Create sensor and interpreter classes for the ultrasonic sensors, along with a controller that moves the car forward if the way forward is clear, and stops it if there is an obstacle immediately in front of it

2. Wrap the ultrasonic sensor, interpreter, and controller into `RossROS` consumer-producers

3. Add the ultrasonic-based driving control to the `ThreadPoolExecutor` execution list

Deliverables

1. Code review from "Car week 4" partner

2. Code review of "Car week 4" partner's work

3. Reflection questions:

   (a) What did you learn while completing the tasks in this lesson?

   (b) With the architecture that you've created, what is the next feature that you would add to the robot control stack?

Part II

Fixed-base manipulators

0 Set up and secure your Raspberry Pi

Read through the instructions here all the way through before carrying out the steps in the linked documents – not all steps in the external documents will be followed, and some will be modified. (Also note that some of these steps, such as creating your GitHub account, will have already been completed in the car unit; they are repeated here for separability.)

Disk images for the arms can be found at

```
https://drive.google.com/drive/folders/1VFbx3n0GjP46kT-yj92-
lW7LcfXnrcEs?usp=sharing
```

0.1 If you have a camera mounted on a separate support

1. Extract the SD card from the robot and flash it using the provided image.

2. Arrange the arm, localization mat, and camera support.

3. On your computer, download and install VNC Viewer from

   ```
   https://www.realvnc.com/en/connect/download/viewer/
   ```

4. Connect your computer to the HiWonder wifi network provided by the arm. (During this step, you will be temporarily not able to connect to the internet).

5. Use VNC viewer to log into the arm, with hostname `raspberrypi.local`, username `pi` and password `raspberry`.

6. Remove the Chinese localization from the system:

   (a) Edit the locale file by entering

   ```
   sudo nano /etc/default/locale
   ```

   at a terminal command line and changing it to read

   ```
   LANG=en_US.UTF-8
   LC_ALL=en_US.UTF-8
   LANGUAGE=en_US.UTF-8
   ```

   (b) Fix the keyboard layout by modifying `/etc/default/keyboard` so that `XBLAYOUT="US"`

   (c) Right-click on the keyboard symbol in the top-right of the system menu bar, select “Configure”, and then replace the Chinese input methods with an appropriate English keyboard.

   (d) Modify `/etc/wpa_supplicant/wpa_supplicant.conf` so that the country is “US”

7. Change the name of your Raspberry Pi by using

   ```
   sudo raspi-config
   ```

   and then `System Options > Hostname` (In some cases, this may be listed under “network options”).

8. Move your Raspberry Pi onto a wireless network accessible from the internet:

    (a) Open the file

    ```
    /boot/hiwonder_wifi_conf.py
    ```

    (You will need super-user privileges for this, e.g., using `sudo nano` or `sudo vim`)

    (b) Uncomment the line that starts with `HW_WIFI_MODE` and set it to mode 2, so that it reads

    ```
    HW_WIFI_MODE = 2
    ```

    (c) Uncomment the line starting with `HW_WIFI_STA_SSID` and fill in your network name, so that it reads

    ```
    HW_WIFI_STA_SSID = 'MyNetworkName'
    ```

    (d) Uncomment the line starting with `HW_WIFI_STA_PASSWORD` and fill in your network password, so that it reads

    ```
    HW_WIFI_STA_PASSWORD = 'MyNetworkPassword'
    ```

    (e) Reboot your system.

9. Carry out some basic setup operations for securing your Raspberry Pi. When copying passwords from your password manager into a terminal running inside VNC, you should be able to right-click in the terminal window to paste from your computer's buffer.

    (a) Start with the steps outlined at

    ```
    https://www.raspberrypi.com/documentation/computers/
    configuration.html#securing-your-raspberry-pi
    ```

    but **do not delete the user 'pi'**. For your robots, I recommend having the `sudo` command require a password and then setting a long timeout on it, by using

    ```
    sudo visudo
    ```

    and then adding a line like

    ```
    Defaults:USER timestamp_timeout=60
    ```

    (where USER is your username, and 60 is the number of minutes I've specified for the timeout)

    (b) After creating your username and using `sudo su - username` to switch to your user, use `sudo raspi-config` and then use

    ```
    Boot Options > Desktop / CLI > Desktop Autologin Desktop GUI, require
        user to log in
    ```

    I recommend having your password generator produce a "memorable" password for your account rather than a "fully random" password, because it is not possible to paste a password into the login dialog. You could also choose to have the system automatically log you in under your username, but this reduces the security of your installation.

(c) Make sure to follow the instructions at

    https://www.raspberrypi.com/documentation/computers/os.html#
    updating-and-upgrading-raspberry-pi-os

(which are linked from the security-setup instructions above) for updating the system code. You are already starting with a "Buster" install, so you can ignore the section about upgrading from "Stretch". Don't worry about the "Third-party solutions" material for this class (though it is worth knowing about it if you start operating robots that you cannot physically access).

(d) Turn on SSH for your system in `raspi-config > Interface Options`.

(e) Make sure to set up your SSH installation to automatically update itself by following the instructions at

    https://www.raspberrypi.com/documentation/computers/using_linux.
    html#scheduling-tasks-with-cron

(also linked from the main security-setup instructions). You'll be rebooting this robot often, so it's probably easiest to put

```
reboot sudo apt install openssh-server
```

as your line in the `crontab` file.

(f) For the allow/deny options, put in an "Allow" for the new username you created, and a "Deny" for the 'pi' user.

(g) If you are on a Windows machine, use

    https://www.ssh.com/ssh/putty/windows/puttygen

to generate your SSH keys.

(h) When disabling password-based authentication (after setting up SSH keys), note that `PasswordAuthentication` needs to be uncommented as well as changed from `yes` to `no`.

(i) Turn on the `ufw` firewall, and set the options `sudo ufw allow ssh`, `sudo ufw limit ssh/tcp`, and `sudo ufw allow 5900` (which enables VNC access).

(j) Install `fail2ban` as per the instructions. Skip the part about modifying the `jail.local` file, which appears to be slightly out of date – the jail for sshd is already configured and active. If you expose your robot to more public parts of the internet, you may want to learn more about `fail2ban` settings.

(k) Remove the stored plaintext of your WiFi password by following the instructions at

    https://carmalou.com/how-to/2017/08/16/how-to-generate-passcode-
    for-raspberry-pi.html

and then also put the passcode into the `hiwonder_wifi_conf.py` file. This step isn't critical. Some discussion on why it's worth doing is at

    https://superuser.com/questions/1411604/what-security-does-the-
    wpa-passphrase-tool-actually-add-to-a-wpa-supplicant-conf

10. Configure your Raspberry Pi to connect to a GitHub account (or other Git repository)

(a) Create an account at GitHub.com. If you have a GitHub account already, you can use it. If you have a different Git setup that you prefer to use, go ahead and use it, and modify the other setup instructions as needed.

(b) Create a new Git repository in your account named `RobotSystems`

(c) Set up SSH keys on your GitHub account.

   i. Go to the Settings page for your account (Use the dropdown menu in the top right corner of the page; Settings is near the bottom of the menu.)

   ii. Go to "SSH and GPG keys" in the list of settings panes at the left of the screen

   iii. Open the "guide to generating SSH keys" link **in a new tab**.

   iv. SSH into your Raspberry Pi, and then follow the instructions to create an SSH key **on your Raspberry Pi** and add that key to your GitHub account. This will be similar to the process you used to create the SSH key that lets you log into your Raspberry Pi from your computer. On the "Generating a new SSH key and adding it to the ssh-agent" page, make sure you are reading the "Linux" instructions (because you are performing operations on the Raspberry Pi).

   v. Set a non-empty passphrase for your SSH key. This means that your key is encrypted on your Raspberry Pi, and cannot trivially be used by others if your robot is lost or stolen. (This precaution is more important on something like a Raspberry Pi than on a computer with disk encryption enabled.
   If you follow the link for "Working with SSH key passphrases", you can follow the instructions under "Auto-launching ssh-agent on Git for Windows" to modify the `.profile` file on your Raspberry Pi so that you will be asked for this passphrase each time you boot the Raspberry Pi, and won't be asked for it again when you perform Git operations. If you are running `zsh`, there are various solutions online, e.g.,

   `https://www.vinc17.net/unix/index.en.html#zsh-ssh-utils`

   that offer more elegant handling of SSH keys, such as only prompting you for a passphrase when you actually use the keys instead of on startup.

   vi. On the "Adding a new SSH key to your GitHub account" page, the fact that you are SSH'd into the computer on which you are generating keys makes Step 1 (copying the ssh key into your clipboard) a bit trickier than they otherwise would be. The easiest way to make this work will probably be to run

   ```
   nano .ssh/id_ed25519.pub
   ```

   and then copy the contents of the file out of the terminal.

   vii. If you left the original Settings page open in another tab, you can then follow the remaining steps on the "Adding a new SSH key to your GitHub account" page.

0.2 If you have an arm with a camera mounted above the gripper

1. Extract the SD card from the robot and flash it using the provided image.

2. Arrange the arm and localization mat.

3. Connect your computer to the HiWonder wifi network provided by the arm. (During this step, you will be temporarily not able to connect to the internet).

4. On your computer, download and install the NoMachine remote desktop software from

   ```
   https://www.nomachine.com
   ```

   When you run the NoMachine client, change “Desktop shared” in the lower right corner of the main screen to “Desktop not shared” (unless you also want to be able to use NoMachine to remotely control your computer).

5. Log into the arm, with username `ubuntu` and password `hiwonder`.

6. Change the name of your Raspberry Pi by using

   ```
   sudo nano /etc/hostname
   ```

   to open the file that the Raspberry Pi takes its name from, and then replacing `ubuntu` with the name you want the arm to have. Then use

   ```
   sudo nano /etc/hosts
   ```

   to open the file that the Raspberry Pi uses to map computer names to IP addresses. Add a line

   ```
   127.0.0.1 my_new_hostname
   ```

   to this file (but **do not delete** the lines with `localhost` or `ubuntu`).

7. Move your Raspberry Pi onto a wireless network accessible from the internet:

   (a) Open the file

   ```
   /boot/firmware/Hiwonder/wifi.yaml
   ```

   (You will need super-user privileges for this, e.g., using `sudo nano` or `sudo vim`)

   (b) Uncomment the line

   ```
   mode: "client"
   ```

   (c) Uncomment the lines

   ```
   client_mode:
     ssid: "my_network_name"
     password: "my_network_password"
     timeout: 30
   ```

   (filling in the appropriate network name and password)

   (d) Reboot your system.

8. Carry out some basic setup operations for securing your Ubuntu Raspberry Pi. When copying passwords from your password manager into a terminal running inside NoMachine, you should be able to right-click in the terminal window to paste from your computer’s buffer. For this system, you will be working as the `ubuntu` user rather than setting up your own username, because the operating software on this arm depends on this user being active.

(a) Change the password for the `ubuntu` user using the `passwd` command and a secure password generated by your password manager.

(b) Make the `sudo` command require a password and then set a long timeout on it, by using

```
sudo visudo
```

and then adding a line like

```
Defaults: ubuntu timestamp_timeout=60
```

(where ubuntu is the username, and 60 is the number of minutes I've specified for the timeout)

(c) Update any system packages by running:

```
sudo apt update
sudo apt upgrade
sudo apt full-upgrade
sudo apt autoremove
```

(d) Install SSH on your system with:

```
sudo apt install openssh-server
```

Confirm the SSH service is running with:

```
sudo systemctl status ssh
```

If it is not, enable it with

```
sudo sytemctl enable ssh
```

(e) Set up your SSH installation to automatically update itself. The process is the same for Ubuntu as it is on Raspberry pi OS; follow the instructions at

`https://www.raspberrypi.com/documentation/computers/using_linux.html#scheduling-tasks-with-cron`

(also linked from the main security-setup instructions). You'll be rebooting this robot often, so it's probably easiest to put

```
@reboot sudo apt install openssh-server
```

as your line in the `crontab` file.

(f) Edit your SSH configuration, located at `/etc/ssh/sshd_config`:

i. Only allow access by the ubuntu user, by adding an "AllowUsers ubuntu" line.

ii. If you are on a Windows machine, use

`https://www.ssh.com/ssh/putty/windows/puttygen`

to generate your SSH keys.

iii. When disabling password-based authentication (after setting up SSH keys), note that `PasswordAuthentication` needs to be uncommented as well as changed from `yes` to `no`. Make sure your key-based authentication works *before* you do this!

   iv. Turn on the `ufw` firewall, and set the options `sudo ufw allow ssh` and `sudo ufw limit ssh/tcp`.
   v. Install `fail2ban` as per the instructions. Skip the part about modifying the `jail.local` file, which appears to be slightly out of date – the jail for sshd is already configured and active. If you expose your robot to more public parts of the internet, you may want to learn more about `fail2ban` settings.

9. Configure your Raspberry Pi to connect to a GitHub account (or other Git repository)

   (a) Create an account at GitHub.com. If you have a GitHub account already, you can use it. If you have a different Git setup that you prefer to use, go ahead and use it, and modify the other setup instructions as needed.
   (b) Create a new Git repository in your account named `RobotSystems`
   (c) Set up SSH keys on your GitHub account.
      i. Go to the Settings page for your account (Use the dropdown menu in the top right corner of the page; Settings is near the bottom of the menu.)
      ii. Go to "SSH and GPG keys" in the list of settings panes at the left of the screen
      iii. Open the "guide to generating SSH keys" link **in a new tab**.
      iv. Use NoMachine to connect to into your Raspberry Pi, and then follow the instructions to create an SSH key **on your Raspberry Pi** and add that key to your GitHub account. This will be similar to the process you used to create the SSH key that lets you log into your Raspberry Pi from your computer. On the "Generating a new SSH key and adding it to the ssh-agent" page, make sure you are reading the "Linux" instructions (because you are performing operations on the Raspberry Pi).
      v. Set a non-empty passphrase for your SSH key. This means that your key is encrypted on your Raspberry Pi, and cannot trivially be used by others if your robot is lost or stolen. (This precaution is more important on something like a Raspberry Pi than on a computer with disk encryption enabled.
      If you follow the link for "Working with SSH key passphrases", you can follow the instructions under "Auto-launching ssh-agent on Git for Windows" to modify the `.profile` file on your Raspberry Pi so that you will be asked for this passphrase each time you boot the Raspberry Pi, and won't be asked for it again when you perform Git operations. If you are running `zsh`, there are various solutions online, e.g.,

      `https://www.vinc17.net/unix/index.en.html#zsh-ssh-utils`

      that offer more elegant handling of SSH keys, such as only prompting you for a passphrase when you actually use the keys instead of on startup.
      vi. On the "Adding a new SSH key to your GitHub account" page, the fact that you are SSH'd into the computer on which you are generating keys makes Step 1 (copying the ssh key into your clipboard) a bit trickier than they otherwise would be. The easiest way to make this work will probably be to run

```
nano .ssh/id_ed25519.pub
```

      and then copy the contents of the file out of the terminal.

vii. If you left the original Settings page open in another tab, you can then follow the remaining steps on the “Adding a new SSH key to your GitHub account” page.

1 Basic arm operations

The arm comes with some basic programs for identifying colored blocks, picking them up and placing them in designated locations, and stacking them on top of each other. Run these programs to get familiar with their general operation. You may need to edit the position value at which the gripper servo is considered “closed”.

The sample code with the arms is written as a large monolithic loop with no abstraction. Assignments for the remainder of the term will be about refactoring the pick-and-place code into a better format.

Deliverables

1. Code review from “Car week 5” partner
2. Code review of “Car week 5” partner’s work
3. Reflection questions:
    (a) What are your initial thoughts on the provided code?

2 Perception

The pick-and-place code has two key components: A perception component that identifies and locates blocks in the target area, and a motion component that sends commands to the servos.

This week, your mission is to refactor the perception code to make it readable with useful abstractions. As a first step:

1. Identify where in `ColorTracking.py` the perception code is located
2. Make a copy of the code and add comments to it. (Google Translate may be helpful here)
3. Make a flow chart of high-level tasks that the perception code accomplishes
4. Write a Python class with methods corresponding to the high-level tasks you identified.
5. Set up a simple program that uses this class to identify the location of a block in the pickup area and labels it on the video display from the camera.

Once you have finished the above tasks:

1. Identify any additional functionality that appears in the perception code from `ColorSorting.py` and `ColorPalletizing.py`. You may find a code comparison tool (such as the difference viewer in PyCharm) to be useful here.
2. Add any methods necessary to implement this functionality into your perception class.
3. Set up a simple program that demonstrates this added functionality.

Deliverables

1. Flow chart showing steps in perception process (including the extra steps in the `ColorSorting.py` code)
2. Reflection questions:
   (a) What did you learn while completing the tasks in this lesson?

3 Motion

The pick-and-place code has two key components: A perception component that identifies and locates blocks in the target area, and a motion component that sends commands to the servos.

This week, your mission is to refactor the motion code to make it readable with useful abstractions. As a first step:

1. Identify where in `ColorTracking.py` the motion code is located
2. Make a copy of the code and add comments to it. (Google Translate may be helpful here)
3. Make a flow chart of high-level tasks that the motion code accomplishes
4. Write a Python class with methods corresponding to the high-level tasks you identified.
5. Combine this motion class with your perception class to perform a basic pick-and-place operation.

Once you have finished the above tasks:

1. Identify any additional functionality that appears in the motion code from `ColorSorting.py` and `ColorPalletizing.py`. As in the perception assignment, you may find a code comparison tool (such as the difference viewer in PyCharm) to be useful here.
2. Add any methods necessary to implement this functionality into your motion class.
3. Use your motion class with your perception class to implement block sorting and block stacking.

Deliverables

1. Code review from "Arm week 2" partner
2. Code review of "Arm week 2" partner's work
3. Flow chart showing steps in motion process (including the extra steps in the `ColorSorting.py` code)
4. Reflection questions:
   (a) What did you learn while completing the tasks in this lesson?
   (b) The final two weeks of the course will be a project with the arm. Propose an idea for something "interesting" to do with the arm.

4 Final Project

For the two weeks, do something "interesting" with the arm that goes beyond the previous lessons. "Interesting" here is open to wide interpretation, and could range from getting interesting behavior out of the arm (e.g., writing with a pen or chalk, or attaching the camera to the arm and implementing visual servoing) to taking a deeper dive into the lower-level arm code (e.g., reworking the arm kinematics implementations). You may form groups of two or three students for the projects.

Deliverables, Week 9

1. Code review from "Arm week 3" partner
2. Code review of "Arm week 3" partner's work
3. Progress update for your project:
   (a) Names of group members
   (b) One paragraph description of project concept
   (c) Flow chart or other high-level description of how your project code is expected to work
   (d) Description of where you are in the design/implementation process
4. Reflection questions:
   (a) What did you learn this week?

Deliverables, Week 10

1. Project report:
   (a) Description of your project
   (b) Flow chart or other high-level description of how your project code works
   (c) Video of your robot operating with your code. (If you do a lower-level code dive, you may want to include something else here to show off what you did)
2. Reflection questions:
   (a) What did you learn this week?

Deliverables, Finals Week

1. Short (< 5 minute) video presentation about your project. I'll make the videos viewable in a shared folder, and we'll have a watch session with time for questions during the "Final exam" time for the class.
2. Code review of your project from "Arm week 5" partner (if your partner is in your group, find a new partner – there should be another group in a similar situation)
3. Code review of "Arm week 5" partner's project