

Talking with Kyle the Robot

Log into the robot's Linux operating system using SSH and the hostname

If you are logged into the same wifi network as the robot, most systems will be able to login at the Terminal / Git Bash using the simple SSH command. Just say yes to any security warnings you get.

```
$ ssh pi@dex.local
```

Password: kiavorobot

If this doesn't work, log in using SSH and the robot's IP address:

```
$ ssh pi@192.168.254.23
```

If you are successful, the command prompt will be colorful and look like this:



Tell Kyle to play a song

Kyle's brain is a Raspberry Pi computer that uses a **flavor** of the Linux operating system called Raspbian, and a specific custom flavor designed for the GoPiGo robot called Raspbian for Robots. The operating system is almost entirely located on the SD card.

Raspbian comes preinstalled with a command line media player called **omxplayer**.

I have loaded a couple of music files onto Kyle in the ~/Music directory. Go into that directory and list its contents:

```
$ cd ~/Music  
$ ls
```

There are a couple of **wav** and **mp3** audio files. Play one of those files with the command:

```
$ omxplayer filename.mp3
```

Obviously you replace the filename with the name of the file that you find inside of Music.

Once the music is playing, the 'p' key will pause or play the music, 'q' will quit the omxplayer program, and '+' and '-' will change the volume.

Load a song onto Kyle

Go onto YouTube and find a song that you would like Kyle to play. Copy the shareable link to that video. Search for a "Youtube to mp3 converter" online, paste the Youtube link to the appropriate place on the website, and download that mp3 file into your Downloads directory.

In a separate Terminal / Git Bash window, navigate on your own computer to the Downloads directory. From here, we can use the SCP command line utility to "secure copy" the file to Kyle's Music folder.

```
~/Downloads $ scp ./filename.mp3 pi@dex.local:~/Music
```

If "dex.local" doesn't work, you can use the IP address listed above.

Once your file makes it onto Kyle, you can play it using

```
$ omxplayer filename.mp3
```

Create your own personal directory on the robot

To control the robot, you will need a place to store your Python files. I have made a directory called ~/Students, so take a look at the contents of Students. It may be empty or already have some student names.

```
$ ls ~/Students
```

Make a directory with your name or username. If I were to create a directory for myself, I would probably call it 'marko': (**marko is just an example name!!**)

```
$ mkdir ~/Students/marko
```

Go into your custom directory:

```
$ cd ~/Students/marko
```

List the contents (there shouldn't be anything yet)

```
$ ls
```

Important! Create a reset button program

When you run a program, you can kill it at the command line with control-c. Sometimes, however even after a program has been stopped, the robot doesn't quite get the message and just keeps doing what it's doing. This can be a bad thing! If the robot's wheels are spinning, the robot could hit a wall. If it's on a table, it could drive itself off the table and fall off the floor.

So we need a reset button! We need something that will really shut down all of the robot's external processes.

In your personal directory, we will create a file called reset.py.

Linux has a command called **touch** for creating files. In your own personal directory, create a new file called reset.py:

```
$ touch reset.py
```

List the contents of the directory with the **ls** command.

```
$ ls
```

Kyle's operating system, Raspbian for Robots, has a text editor called nano that you can use even over a remote connection to write code or text.

Open reset.py up for editing with nano:

```
$ nano reset.py
```

Put the following code into this file and save it.

```
import gopigo3
GPG = gopigo3.GoPiGo3()
GPG.reset_all()
```

You can save the file with control-O (the letter O, not 0).

A message will appear in nano asking if you want to write to the filename reset.py. Press enter to confirm that you would like to write/save to this filename. Now exit nano with control-X.

Test this file at the command line with Python:

```
$ python reset.py
```

Nothing should happen, but keep a terminal window open with this command ready so that in case you need to stop Kyle from doing his crazy stuff, you just run the reset.py program and he will stop.

The GoPiGo libraries

To get Kyle to take action, we will be using the Python language and a few custom Python libraries designed specifically for the GoPiGo robot. For the reset.py program, we already used the gopigo3 library. Dexter Industries, the developer of the GoPiGo robot, has developed a few useful libraries for telling the robot what to do.

The available libraries are easygopigo3, gopigo3, and easysensors.

To use one of these libraries, in the top of your Python program you just add a line that says

```
import easygopigo3
```

or

```
import gopigo3
```

or

```
import easysensors
```

Tell Kyle to flash his blinkers

There are a couple ways to give Python instructions to the robot. One of them is to open the Python shell and type commands one line at a time. This is useful for getting immediate results that show you cause and effect when you write Python code.

```
$ python
>>> import easygopigo3 as easy
>>> gpg = easy.EasyGoPiGo3()
```

This loads a set of useful robot functions into the Python's memory.

Kyle has two red led's on the front side that are kind of in the same place as a car's blinkers.

To turn on the left "blinker"

```
>>> gpg.led_on("left")
```

To turn off the left "blinker"

```
>>> gpg.led_off("left")
```

How do you think you turn on the right "blinker"?
How do you think you turn off the right "blinker"?

Once you are done playing with the blinkers, quit the Python shell and go back to the BASH shell with:

```
>>> quit()
$
```

If the blinkers are still on even though you quit the program, you can turn them off by running your reset.py program:

```
$ python reset.py
```

Tell Kyle to flash the colorful LED lights on his back

Start the Python shell:

```
$ python
>>> import easygopigo3 as easy
>>> gpg = easy.EasyGoPiGo3()
```

Define some colors. For now, we'll just use red, green, and blue.

```
>>> RED = (125, 1, 1)
>>> GREEN = (1, 125, 1)
>>> BLUE = (1, 1, 125)
```

Using these constants, set the left eye's color:

```
>>> gpg.set_left_eye_color(RED)
```

"Open" the left eye with the color you have set:

```
>>> gpg.open_left_eye()
```

Close the left eye with:

```
>>> gpg.close_left_eye()
```

Try setting the left eye to a different color, then open and close it.

Then try setting the right eye to a different color, and open and close it. We will come back to this later.

Once you are done playing with the eyes, quit the Python shell and go back to the BASH shell with:

```
>>> quit()
$
```

If the eyes are still on even though you quit the program, you can turn them off by running your reset.py program:

```
$ python reset.py
```

Tell Kyle to drive forward

Please do this part with a partner! One person should be at the command line and the other should be spotting/watching the robot to make sure that it doesn't bump into anything! And take turns!

Start the Python shell:

```
$ python
>>> import easygopigo3 as easy
>>> gpg = easy.EasyGoPiGo3()
```

Below there is a command will make Kyle drive forward. Be careful, he will drive forward forever until you tell him to stop!

With that in mind, copy the command for stop into your clipboard so that you can paste it onto the Python shell prompt to stop Kyle in an emergency.

```
>>> gpg.stop()
```

Now with gpg.stop() ready to go in your clipboard, you can type the command to make Kyle drive forward:

```
>>> gpg.forward()
```

Make sure to stop him before he hits a wall or other obstacle!

This command will cause Kyle to drive for 50 cm and then stop:

```
>>> gpg.drive_cm(50, True)
```

Now that you have caused Kyle to drive forward a couple of times, exit out of the Python shell with

```
>>> quit()
```

and if necessary run the reset script:

```
$ python reset.py
```

Other drive commands

While you wait your turn to run the robot, please study the following commands:

```
# drives backward forever
gpg.backward()
```

```
# turns left forever
gpg.left()
```

```
# turns right forever
gpg.right()
```

Write a multi-step driving program

Create a new file in your personal directory and call it driving.py

```
$ touch driving.py
```

Open the file for editing with the nano text editor

```
$ nano driving.py
```

First I will show you an example multi-step program that causes Kyle to move forward for one second, turn left for one second, and then stop.

```
import time
import easygopigo3 as easy
gpg = easy.EasyGoPiGo3()

gpg.forward()
time.sleep(1)

gpg.left()
time.sleep(1)

gpg.stop()
```

Save this file with ctrl-O, hit enter to confirm the filename, then exit nano with ctrl-x.

Making sure to have a partner standing next to the robot to make sure it doesn't drive off a cliff, run the program with

```
$ python driving.py
```

Now make your own custom driving program using the tools you have learned so far!

Turn Kyle's servo (the motor that rotates his "head"):

It might be best to do this at the Python shell rather than as a multiline program:

```
$ python
```

```
>>> import easygopigo3 as easy
>>> import gopigo3
>>> import easysensors
>>> GPG = gopigo3.GoPiGo3()
>>> gpg = easy.EasyGoPiGo3()
>>> servo = easysensors.Servo(port='SERVO1', gpg = GPG)
```

Now that all the libraries have been imported and the special robot objects have been created, we can use them to turn Kyle's head.

To bring the servo to straight forward:

```
>>> servo.rotate_servo(90)
```

To turn the servo 10 degrees to the right:

```
>>> servo.rotate_servo(80)
```

What number do you think would turn the servo 10 degrees to the left?

If the motor sounds like it is working but the servo is not moving, this could be bad and could burn out the motor. Stop the motor with

```
>>> GPG.reset_all()
```

Exit the shell with

```
>>> quit()
$
```

Detect the distance in front of Kyle

Using the tools in the previous section, make sure that the servo is pointed straight forward (to 90 degrees).

This is probably best as a multiline program. Call it distance.py.

```
import time
import easygopigo3 as easy

gpg = easy.EasyGoPiGo3()

my_distance_sensor = gpg.init_distance_sensor()
```

```
while True:
    print("Distance Sensor Reading: {} mm ".format(my_distance_sensor.read_mm()))
```

Run the program with Python:

```
$ python distance.py
```

Get Kyle to tell the temperature and weather

This is probably best as a multiline program. Call it weather.py.

```
from di_sensors import easy_temp_hum_press as ea$
THP_sensor = easyTHP.EasyTHPSensor()
while True:
    print('Temperature (°F):', THP_sensor.safe_fahrenheit())
```

Run the program with Python and see what happens.

Control the robot with the keyboard

Inside of ~/robot/basicmoves there is a file called keyboard_control.py. If you run this script, you can drive the robot with the keyboard.

```
$ cd ~/robot/basicmoves
```

```
$ python keyboard_control.py
```

Make sure to read the instructions displayed by the program before you touch any keys, but once you understand what to do, feel free to drive the robot around, making sure it doesn't crash into anything!

View the webcam

Inside the ~/RPi_Cam_Web_Interface directory there is a file called start.sh. A package has already been installed that will take the Pi's webcam and serve it (broadcast it) so that you can view it in your web browser. To start this program, navigate to ~/RPi_Cam_Web_Interface and then start the shell script:

```
$ cd ~/RPi_Cam_Web_Interface
$ ls
$ ./start.sh
```

In a web browser, navigate to dex.local/cam. You might have to replace dex.local with the robot's IP address.

Now you should see the camera's output!

To stop the webcam, use the stop.sh script:

```
$ ./stop.sh
```


