MODULE 7: More camera, Motors, Plotting & Python

The process of taking pictures with the camera can be sped up considerably if the step of actually writing the image to the disk is eliminated. The use of the motor will be explored as well as how to analyze the data that is gathered to determine the speed of the wheel.

GOALS:

- Illustrate how to obtain the image directly in RAM to speed up processing
- Investigate the use of a small DC motor
  - Discuss the need for power isolation
- Develop a method to determine the motor speed
- Examine how to plot and analyze data in python

More Camera

Taking a picture and saving it to non-volatile storage (the SD card), then reading that image from the SD card takes time. Also, it is not necessary to save a copy of the image for later use, which just wastes the limited space available on the SD card. So, when using the image data to make real time decisions, it is advantageous to reduce the amount of time it takes for the Pi to process the image. As long as saving the image for later use is not a priority, it should be much faster to just import the image directly to a matrix format that python can use and skip the entire process of writing and then reading the image.

The following code illustrates the methods used to save pictures, as well as how long each of these tasks takes. When the code was run, the times printed for each task were printed. Lines 7-9 set up the camera object and the stream, these would be done just once for a program and take nearly a quarter of a second. By far the longest time taken, nearly 2 seconds, was in taking and saving a large image, but when the resolution was lowered which results in a smaller picture, the time was cut by over 60%. Saving the image directly to ram further reduced the time by another 25% [1]. What follows is the data printed when the program was executed.

> *** *HINT* ***
> Checking how long a task takes can be important when the task will be implemented in a real-time application. Discovering time consuming tasks can help to identify places where the code will later cause bottlenecks for the overall system.

```
# time to setup object PiCamera: 0.234
# time to take and save first picture: 1.852
# time to take and save smaller picture: 0.658
# time to take streamed image: 0.488
# time to save streamed image: 0.027
```

It turns out that the time to save directly to RAM is fastest, but not significantly. Although at times, even a small savings can be advantageous and it should be remembered that the image would not need to be saved to the SD card in this process, that step was added simply to determine how long it might take if it was needed.

IF ANYONE FIGURES OUT HOW TO SPEED THIS UP FURTHER, LET ME KNOW.

```
1>  from picamera import PiCamera
2>  import picamera.array
3>  import time
4>  import cv2
5>
6>  t1 = time.time()
7>  camera = PiCamera()
8>  camera.framerate = 30
9>  stream = picamera.array.PiRGBArray(camera)  # Create the stream
10>
11> t2 = time.time()
12> camera.capture('testing1.png')          # Take & save large image
13>
14> t3 = time.time()
15> camera.resolution = (320, 208)
16> camera.capture('testing2.png')          # Take & save smaller image
17>
18> t4 = time.time()
19> camera.capture(stream, format='bgr')    # Save directly to memory
20> image = stream.array
21> stream.truncate(0)                      # needed to take another pic
22>
23> t5 = time.time()
24> cv2.imwrite('testing3.png', image)     # Save the image from memory
25> t6 = time.time()
26> print ("time to setup object PiCamera: %1.3f" % (t2-t1))
27> print ("time to take and save first picture: %1.3f" % (t3-t2))
28> print ("time to take and save smaller picture: %1.3f" % (t4-t3))
29> print ("time to take streamed image: %1.3f" % (t5-t4))
30> print ("time to save streamed image: %1.3f" % (t6-t5))
```

DC Motors

Direct Current (DC) motors such as the low powered motor on the right can be used for moving objects. Unlike servos, these motors have just two input lines for power and are not controlled with a GPIO pin, but rather with the amount of voltage that they are provided. The higher the voltage supplied to the motor, the faster the motor will go. Also, by changing the polarity of the voltage, the direction that the motor is spinning can be reversed.

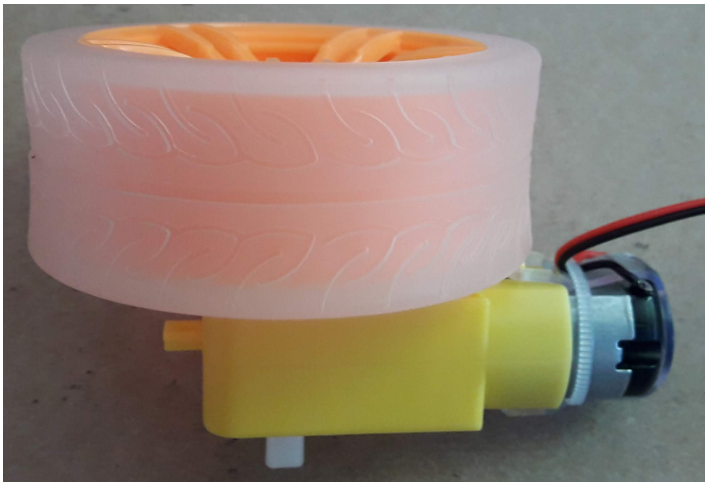


Illustration 1: DC Motor with Wheel

As a low power device, the Pi is not suited to drive significant amounts of current to power other devices. In addition, motors can produce significant

amounts of electrical “noise” which can cause issues with the cpu on the Pi. For this reason it is often advantageous to isolate the power for external devices that require significant power.

> *** *NOTE* ***
> Not only are the IO pins unable to provide enough power for a motor, but in most cases the voltage supplies for the Pi are also incapable of providing enough power to run external motors. For this reason it is often required to run the motors on a power source that is different than the source for the Pi. As mentioned in a previous module, when adding a separate power source, if communication between the devices using the different sources is required, the ground lines will need to be connected together.

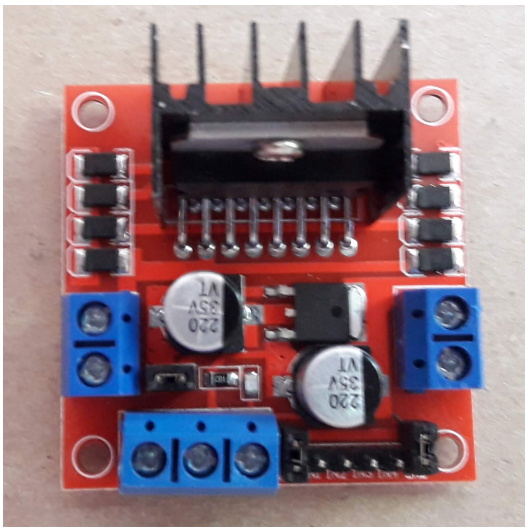


Illustration 2: L298N H-Bridge Module

A common circuit used to drive a motor is an H-Bridge, and in the lab the L298N H-Bridge will be used [2]. The H-Bridge module accepts up to a 12V voltage supply, accepts input from the Pi for controlling the voltage to the motor. This module can be used to control two different motors. Three pins from the Pi are used to control one motor and are labeled on the L298N module as: ENA or ENB – PWM input to control the output voltage to the motor, IN1 or IN3 – if 1, the motor moves in a forward direction, IN2 or IN4 – if 1, the motor moves in a backward direction.

The tutorial listed at the link above has a program that was modified and placed in the repository for this module for use in the lab.

Like the servo, PWM is used to help control the motor. But in this case, PWM is used to regulate the voltage to the motor. The higher the duty cycle, the faster the motor will spin. Due to static friction, small duty cycles may not move the motor at all. And it should not be expected that the output speed will relate linearly to the PWM output driving the motor.

> *** *Warning* ***
> Another reminder. Be careful when connecting multiple power supplies to insure that they are connected properly to insure that no shorts occur that may damage the Pi or other external devices.

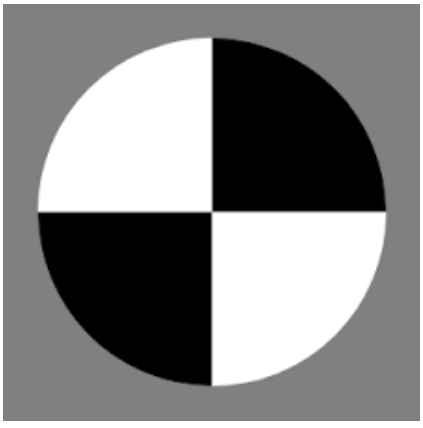


Illustration 3: Disk [3]

Method to Determine Motor Speed

Encoders can be used to determine motor speed. In this lab, the photoresistor will be used along with a colored disk, show in Illustration 3, to determine the motor speed. Since the dark regions will reflect less light than the light regions, this difference may be used to then calculate the speed of the motor. Under different loads, a specific voltage used to drive the motor will result in different speeds, so this device can be used to help regulate the input voltage so that the motor could be driven at the desired speed regardless of the load on the motor.

MODULE 7: More camera, Motors, Plotting & Python

The output of the 10 bit A/D converter is shown in the figure to the right. Clear shifts exist between the dark and light circles. Each transition from dark to light results in a shift down with transitions from light to dark resulting a shift up. 256 samples were taken in this example every 10msec. So, with approximately 11.5 full cycles of transition from black to white occurring over 2.56 seconds the speed of the wheel can be determined.

11.5/2.56 cycles/sec * ½ rev/sec = **2.3 rev/sec**

While this approach works, it needs to be automated in code, which is made more difficult by the fact that the ambient lighting conditions will affect the actual A/D readings.

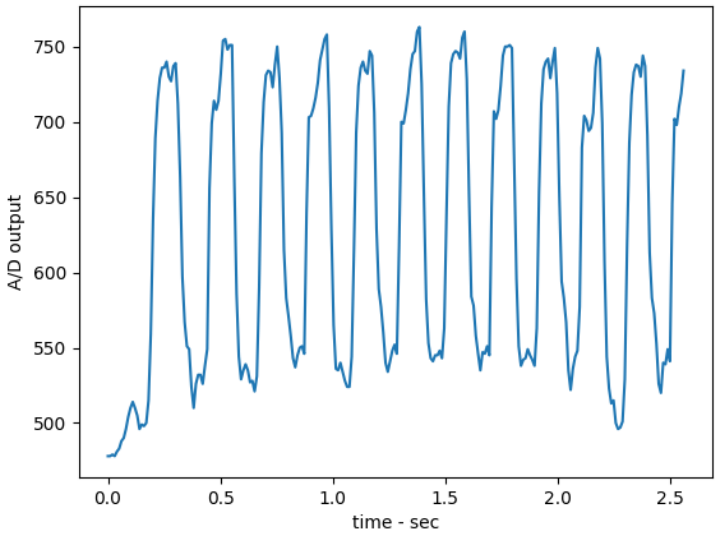


Illustration 4: A/D Output of Photoresistor

This is shown in Illustration 5 which compares the output of the samples taken in two different lighting conditions. Note that the changes are still evident, but it is not just a simple task of comparing the output to a specific value to detect the light/dark and dark/light transitions.

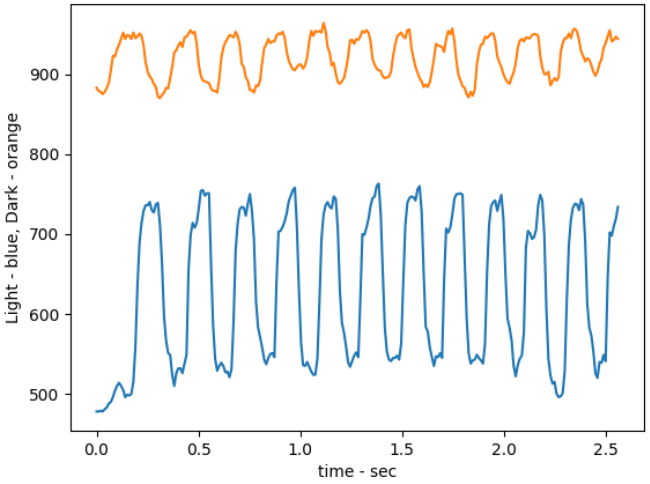


Illustration 5: Comparison of Ambient Light Conditions

One thing that can be done to highlight the transitions is to look at the *change* in the output. However as the relative changes also change with respect to the ambient light, it is best to examine an overall average of the swing of the output and use a threshold based on the anticipated swing. In this manner the algorithm can *adapt* to new lighting conditions. An algorithm was developed that used the following techniques:

- Only examine the change in the output
- Examine the 3 point moving average of the change, this helps to reduce the affect of noise on the readings
- The average of the maximum positive swing above and below the average was found over a subset of the data. This would need to be recalculated if the motor were to be run continously as this average would change with ambient light changes.
- 20% of this average value was used as the threshold to detect a transition
- Once a transition from dark to light was detected, only look for a transition from light to dark next
- Similarly, only look for light/dark after dark/light was detected

The light/dark transitions were marked with a 1 and the dark/light detected were marked with a -1 and plotted with the output for the first data set in Illustration 6.

This algorithm could then just keep track of the detected changes, as well as the time in between them to determine motor speed in real time.

*** *Hint* *** Tips for python plotting [4].

Alternate Method to Detect Speed

A Fourier transform will convert the time based signals over to the frequency domain components for that time based signal. The python scipy library contains code for the FFT or fast Fourier transform, an algorithm that will quickly compute the transform [5].

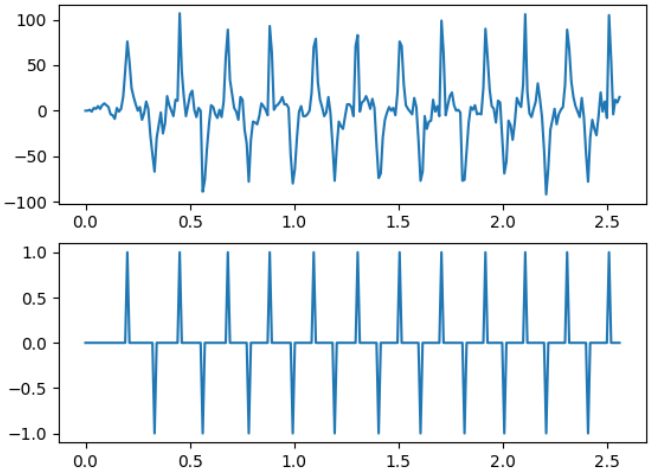


Illustration 6:
Top - Change in output, Bottom - Detected Changes

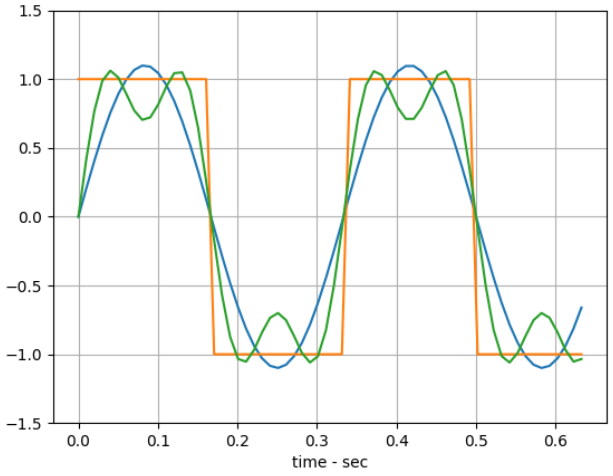


Illustration 7: Time Signals
Square @3Hz, sin @3Hz, sin@3Hz + sin @9Hz

Illustrations 7 & 8 demonstrate the concept. The first shows a square wave at 3Hz, a sinusoid at 3Hz (1.1*sin(2pi*3*t) in blue) and the sum of a sinusoid at 3Hz and a sinusoid at 9Hz (1.1*sin(2*pi*t) + 0.4*sin(2*pi*t) in green). The magnitude of each of the sinusoidal components was obtained from the magnitude of the components at those frequencies for the FFT computed for the square wave shown on Illustration 8. The values of 1.1 and 0.4 were taken from the graph, but can be computed directly from the formulas for the Fourier transform [6].

The more sinusoidal components that are included in the summation, the more the resulting wave will comet to reproducing the square wave. Other components clearly exist at 15Hz, 21Hz, etc.

Now, how can this method be used to determine the speed of the motor? Clearly the time based signal has a component that repeats with a consistent frequency.

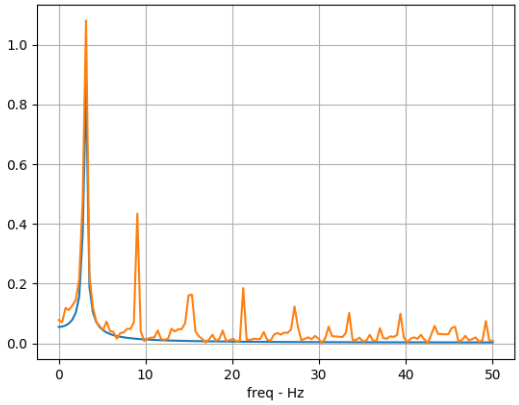


Illustration 8: FFT
*square (orange) & Sin(2*pi*3*t) (blue)*

So, the FFT of this signal can be used to determine this dominant component of the signal.

Illustration 9 is output of the FFT of the data compared with the FFT of the change in the data. The data itself has a large offset, recall from Illustration 4 that the average output was nearly 500. The average value constitutes a Fourier component at a frequency of 0 Hz (since cos(0) equals 1). By taking the FFT of the change in the outout, this offset of DC component is removed. The lower graph clearly shows a peak at 5 Hz, which translates to approximately 2.5 rev/sec, which is comparable to the previous method.

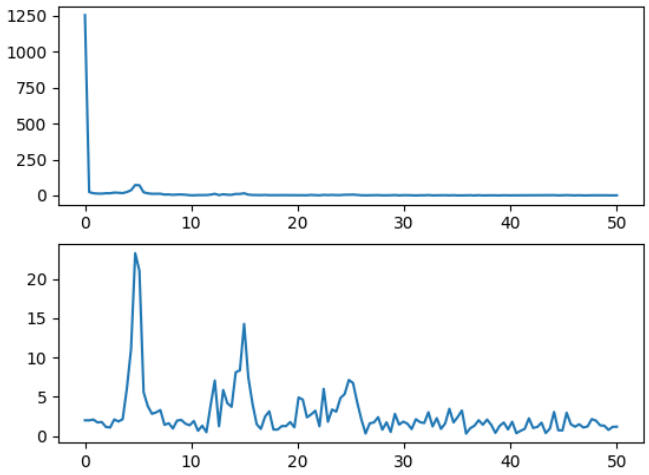


Illustration 9:
Top - FFT of Output, Bottom - FFT of Change in Output

The code used to produce the last two plots is provided below.

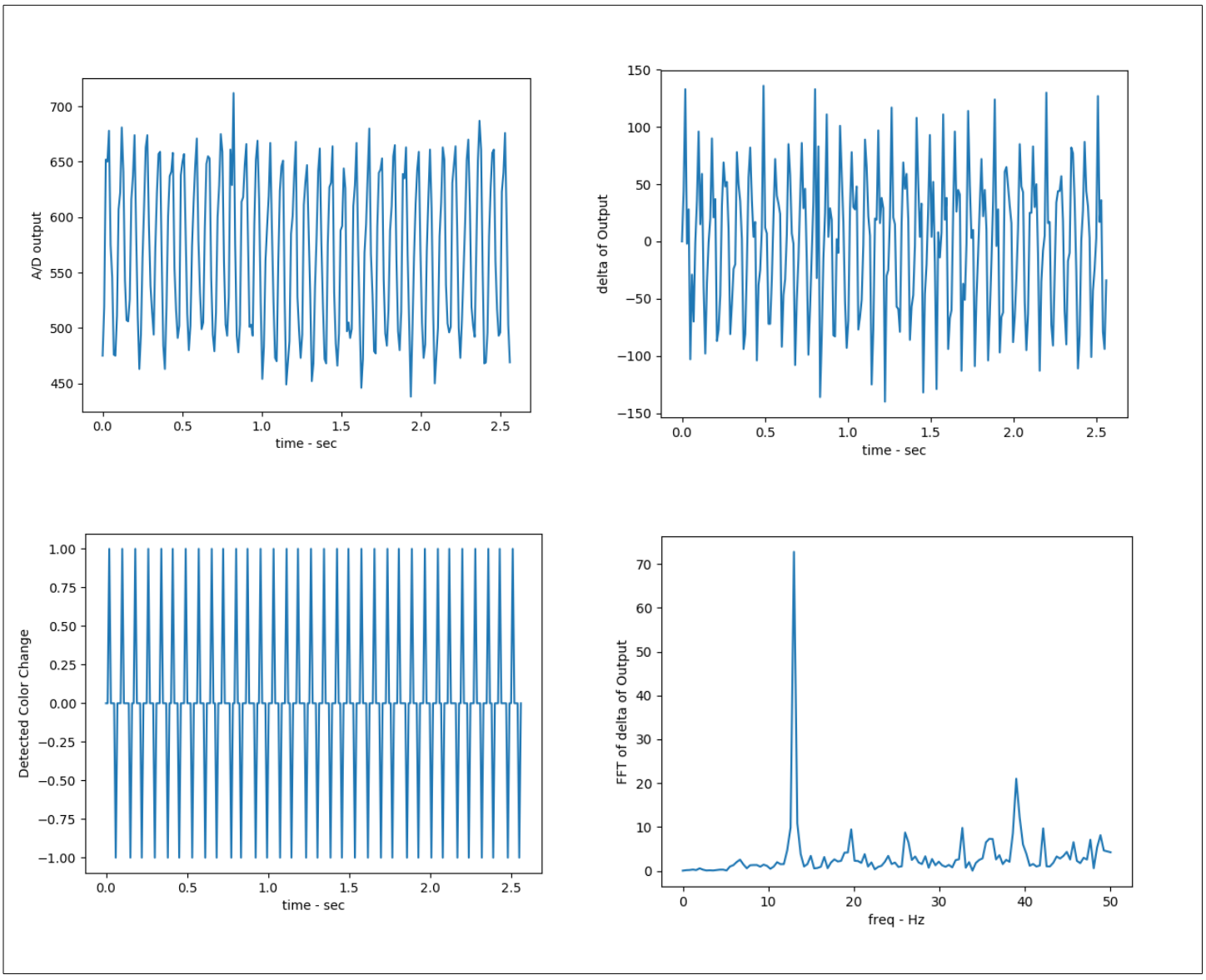
```
1>  from scipy import signal
2>  from scipy.fftpack import fft
3>  import matplotlib.pyplot as plt
4>  import numpy as np
5>
6>  N = 256                      # number of samples
7>  T = 0.01                     # time between samples
8>  t = np.linspace(0, N*T, N)  # time array
9>  squ3Hz  = signal.square(2 * np.pi * 3 * t)
10> sin3Hz  = np.sin(2 * np.pi * 3 *t)
11> sin9Hz  = np.sin(2 * np.pi * 9 *t)
12>
13> sinf = fft(sin3Hz)                    # now calculate the fft
14> squf = fft(squ3Hz)
15> freq = np.linspace(0, 1/(2*T), N//2)   # //2 performs int div by 2
16> plt.plot(freq, 2/N*np.abs(sinf[0:N//2]), freq, 2/N*np.abs(squf[0:N//2]))
17> plt.grid(True)
18> plt.xlabel("freq - Hz")
19> plt.savefig("fft.png")
20>
21> plt.clf()
22> plt.plot(t[:N//4], 1.1*sin3Hz[:N//4],
23>          t[:N//4], squ3Hz[:N//4],
24>          t[:N//4], 1.1*sin3Hz[:N//4] + 0.4*sin9Hz[:N//4])
25> plt.grid(True)
26> plt.ylim(-1.5, 1.5)
27> plt.xlabel("time - sec")
28> plt.savefig("timeSignals.png")
```

MODULE 7: More camera, Motors, Plotting & Python

Quiz/Homework

Prelab – Have this completed prior to showing up in the lab

1. Examine the code submitted for module 5 with the camera. Examine the code from the first section of this module and determine the line numbers relevant to change the program from module 5 so that it does not capture the original image and save it directly to the hard drive.
2. Review the Hbridge connections from the prelab of module 6 and include the circuit diagram required.
3. Assuming that a disk (see Illustration 3) on a wheel.
   a. If 40 transitions from light to dark and dark to light were detected in 3 seconds, what is the rotational speed of the wheel in rotations per second?
   b. If a desired speed of 300 rpm was desired, how many transitions would need to occur every second?



4. Above are images of samples taken at 10msec for a period of 2.56sec, the difference between each sample, the detected transition changes given the algorithm developed and the FFT of the difference between each sample.

   a. What is the span of values expected from the data?
   b. What is the average value of the data collected, the average of the difference?
   c. Count the transitions in the data and compare the the transitions detected.
   d. Use the data from the previous part to calculate a wheel speed in rps (rotations per second).
   e. Use the FFT to compare what the average rotational speed is as well.
5. In order to write the program identified in section 5 of the lab, write code segments that will:
   a. Open a file, write to a file and close the file.
   b. Use the A/D converter to obtain a sample from the photoresitor
   c. Loop for 4 seconds, sampling every 10msec without the use of the sleep method.

LAB

1. Run the timing code from the first part of the lab to check how long it takes for various image capture and saving commands to work on your Pi.
2. Modify the program that find the center of mass of the ball to capture an image and import it directly into ram without saving it to a file. After placing the center of mass on the image, then save it with the center of mass marked with a small dot. Save this as module7a.py.
3. Connect the Hbridge module to the motor and run the code provided to investigate how the motor works. Get approval from the TA/instructor before connecting power to the circuit.
4. `mod7_func.py` is a collection of functions that are useful for the programs in module 7. Create a program that will use the functions to run and test the motor. The function `motor_control` function has a new option for variable duty cycle adjustments.
5. Modify the program from the last part to collect data using the photoresistor regarding the speed of the wheel.
   a. Sample at 10msec for 4 seconds.
   b. Save the data to a file.
   c. Either ask the user for a filename in which to save the data and save that four second set of data with that name (forwardLow.txt for example), or just save it with one name and rename it every time your run the program. You will need several sets of data for the lab (slow, medium and high or you may substitute one of your own duty cycles for one of these values).
   d. Attempt to adjust the lower duty cycles and see what the required duty cycle is required to overcome the initial static friction.
   e. Save this program as module7b.py
6. Process the data obtained from the previous part.
   a. Plot the data and plot the change from one sample to the next, see link [4] under the section "Working with multiple figures" for examples. Make sure the plot has the proper time scale for the horizontal axis.
   b. Manually determine the speed for wheel for one of the data sets.
   c. Implement an algorithm that determines the transitions for the disk and calculates the average rpm for the wheel for a given dataset.
   d. Save the program that plots and calculates the average rpm as module7c.py
7. Write a program that takes one of the datasets, uses a multiple of 2 number of data elements and calculates and displays the fft for this data set and plots the fft as a function of frequency. Use the results from the plot to obtain an estimate for the rpm for that dataset and compare that result to the previous results. Submit this program as module7d.py.

Bibliography

[1] 10. API – picamera.camera Module – PiCamera 1.10 documentation, https://picamera.readthedocs.io/en/release-1.10/api_camera.html, Accessed 10/14/19

[2] Raspberry Pi L298N Interface Tutorial, https://www.electronicshub.org/raspberry-pi-l298n-interface-tutorial-control-dc-motor-l298n-raspberry-pi/, Accessed 10/3/19

[3] Secchi Disk Body of Water Circle Limnology Turbidity PNG, https://imgbin.com/png/J7BJsivT/secchi-disk-body-of-water-circle-limnology-turbidity-png, Accessed 10/14/19

[4] Pyplot tutorial – Matplotlib 3.1.1 documentation, https://matplotlib.org/tutorials/introductory/pyplot.html, Accessed 10/14/19

[5] Fourier Transforms (scipy.fftpack – SciPy v1.3.1 Reference Guide, https://docs.scipy.org/doc/scipy/reference/tutorial/fftpack.html, Accessed 10/14/19

[6] Fourier Transform, https://en.wikipedia.org/wiki/Fourier_transform, Accessed 10/14/19