Components

2019

Components

This chapter explains in depth all the components used to produce the final product. Find below components used to achieve a working product.

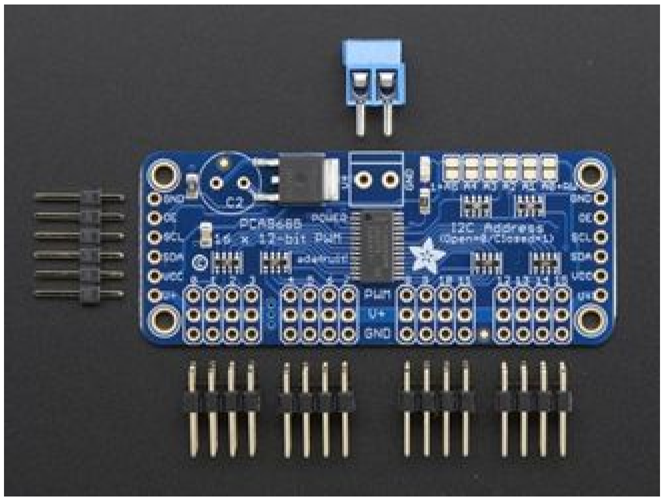
Adafruit PCA9685

This Raspberry Pi add-on is perfect for any motion project as it can drive up to 4 DC or 2 Stepper motors with full PWM speed control.

Servo motors are often driven using the PWM outputs available on most embedded MCUs. But while the Pi does have native HW support for PWM, there is only one PWM channel available to users at GPIO18.

That kind of limits your options if you need to drive more than one servo or if you also want to dim an LED or do some sort of other PWM goodness as well. Thankfully ... the PI does have HW I2C available, which we can use to communicate with a PWM driver like the PCA9685, used on Adafruit's 16-channel 12-bit PWM/Servo Driver!

Using this breakout, you can easily drive up to 16 servo motors on your Raspberry Pi using Python library



Power Pins

- GND - This is the power and signal ground pin, must be connected.
- VCC - This is the logic power pin, connect this to the logic level you want to use for the PCA9685 output, should be 3 - 5V max! It's also used for the 10K pullups on SCL/SDA so unless you have your own pullups, have it match the microcontroller's logic level too!
- V+ - This is an optional power pin that will supply distributed power to the servos. If you are not using for servos you can leave disconnected. It is not used at all by the chip. You can also inject power from the 2-pin terminal block at the top of the board. You should provide 5-6VDC if you are using servos. If you must, you can go higher to 12VDC, but if you mess up and connect VCC to V+ you could damage your board

Control Pins

- SCL - I2C clock pin, connect to your microcontrollers I2C clock line. Can use 3V or 5V logic, and has a weak pullup to VCC
- SDA - I2C data pin, connect to your microcontrollers I2C data line. Can use 3V or 5V logic, and has a weak pullup to VCC
- OE - Output enable. Can be used to quickly disable all outputs. When this pin is low all pins are enabled. When the pin is high, the outputs are disabled. Pulled low by default so it's an optional pin!

Output Ports

- There are 16 output ports. Each port has 3 pins: V+, GND and the PWM output.
- Each PWM runs completely independently but they must all have the same PWM frequency.
- That is, for LEDs you probably want 1.0 KHz but servos need 60 Hz - so you cannot use half for LEDs @ 1.0 KHz and half @ 60 Hz.
- They're set up for servos, but you can use them for LEDs! Max current per pin is 25mA.

- There are 220-ohm resistors in series with all PWM Pins and the output logic is the same as VCC so keep that in mind if using LEDs.

Using the Adafruit Library

It's easy to control servos with the Adafruit 16-channel servo driver. There are multiple Circuit Python libraries available to work with the different features of this board including Adafruit Circuit Python PCA9685

(https://adafru.it/tZF), and Adafruit Circuit Python Servo Kit (https://adafru.it/Dpu). These libraries make it easy to write Python code to control servo motors.

Python Installation of Servo Kit Library

You'll need to install the Adafruit_Blinka library that provides the Circuit Python support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the Circuit Python on Linux guide to get your computer ready (https://adafru.it/BSN)

Once that's done, from your command line run the following command:

sudo pip3 install adafruit-circuit python-servo kit

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use Circuit Python on Python 2.x, it isn't supported!

Controlling Servos

We've written a handy Circuit Python library for various PWM/Servo boards called Adafruit Circuit Python Servo Kit (https://adafru.it/Dpu) that handles all the complicated setup for you. All you need to do is import the appropriate class from the library, and then all the features of that class are available for use. We're going to show you how to import the Servo Kit class and use it to control servo motors with the Adafruit 16-channel servo driver breakout.

First, you'll need to import and initialize the Servo Kit class. You must specify the number of channels available on your board. The breakout has 16 channels, so when you create the class object, you will specify 16.

From adafruit_servokit import Servo Kit

kit = Servo Kit(channels=16)

Now you're ready to control both standard and continuous rotation servos.

Standard Servos

To control a standard servo, you need to specify the channel the servo is connected to. You can then control movement by setting angle to the number of degrees.

For example, to move the servo connected to channel 0 to 180 degrees:

kit.servo[0].angle = 180

To return the servo to 0 degrees:

kit.servo[0].angle = 0

With a standard servo, you specify the position as an angle. The angle will always be between 0 and the actuation range. The default is 180 degrees, but your servo may have a smaller sweep. You can change the total angle by setting actuation range.

For example, to set the actuation range to 160 degrees:

kit.servo[0].actuation_range = 160

Often the range an individual servo recognizes varies a bit from other servos. If the servo didn't sweep the full expected range, then try adjusting the minimum and maximum pulse widths using set_pulse_width_range(min_pulse, max_pulse).

To set the pulse width range to a minimum of 1000 and a maximum of 2000:

kit.servo[0].set_pulse_width_range(1000, 2000)

That's all there is to controlling standard servos with the PWM/Servo HAT or Bonnet, Python and Servo Kit.

Continuous Rotation Servos

To control a continuous rotation servo, you must specify the channel the servo is on. Then you can control movement using throttle. For example, to start the continuous rotation servo connected to channel 1 to full throttle forwards:

kit.continuous_servo[1].throttle = 1

To start the continuous rotation servo connected to channel 1 to full reverse throttle:

kit.continuous_servo[1].throttle = -1

To set half throttle, use a decimal:

kit.continuous_servo[1].throttle = 0.5

And, to stop continuous rotation servo movement set throttle to 0 :

kit.continuous_servo[1].throttle = 0

That's all there is to controlling continuous rotation servos with the PWM/Servo breakout, Python and Servo Kit.

(https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all,2014)

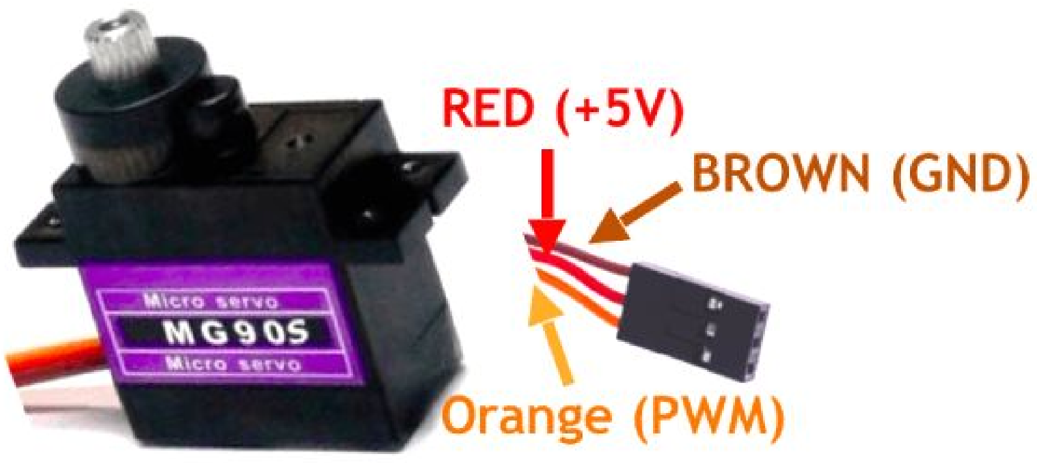
Servos

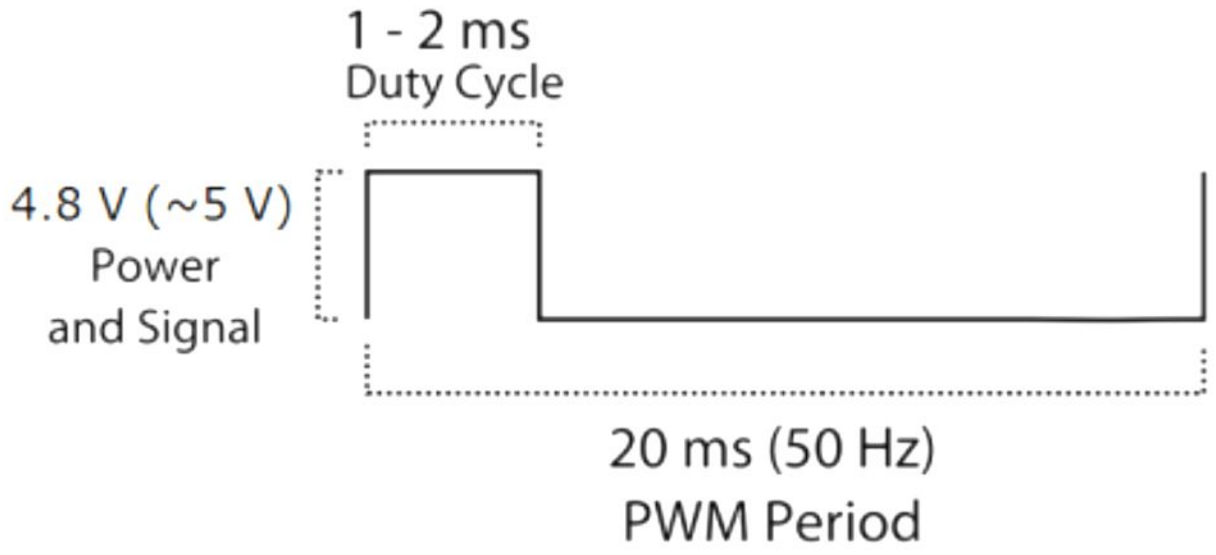
TowerPro MG-90S Features

- Operating Voltage: 4.8V to 6V (Typically 5V)
- Stall Torque: 1.8 kg/cm (4.8V)
- Max Stall Torque: 2.2 kg/cm (6V)
- Operating speed is 0.1s/60° (4.8V)
- Gear Type: Metal
- Rotation: 0°-180°
- Weight of motor: 13.4gm
- Package includes gear horns and screws

How to use a Servo Motor

To make this motor rotate, we must power the motor with +5V using the Red and Brown wire and send PWM signals to the Orange color wire. Hence, we need something that could generate PWM signals to make this motor work, this something could be anything from Microcontroller platforms like Arduino, PIC, ARM to even a microprocessor like Raspberry Pie. Now, how to control the direction of the motor?





From the picture we can understand that the PWM signal produced should have a frequency of 50Hz that is the PWM period should be 20ms. Out of which the On-Time can vary from 1ms to 2ms. So, when the on-time is 1ms the motor will be in the 0° and when 1.5ms the motor will be 90°, similarly when it is 2ms it will be 180°. So, by varying the on-time from 1ms to 2ms the motor can be controlled from 0° to 180°

MG90S Wiring Description

Wire Number	**Wire Color**	**Description**
1	Brown	Ground wire connected to the ground of system
2	Red	Powers the motor typically +5V is used
3	Orange	PWM signal is given in through this wire to drive the motor

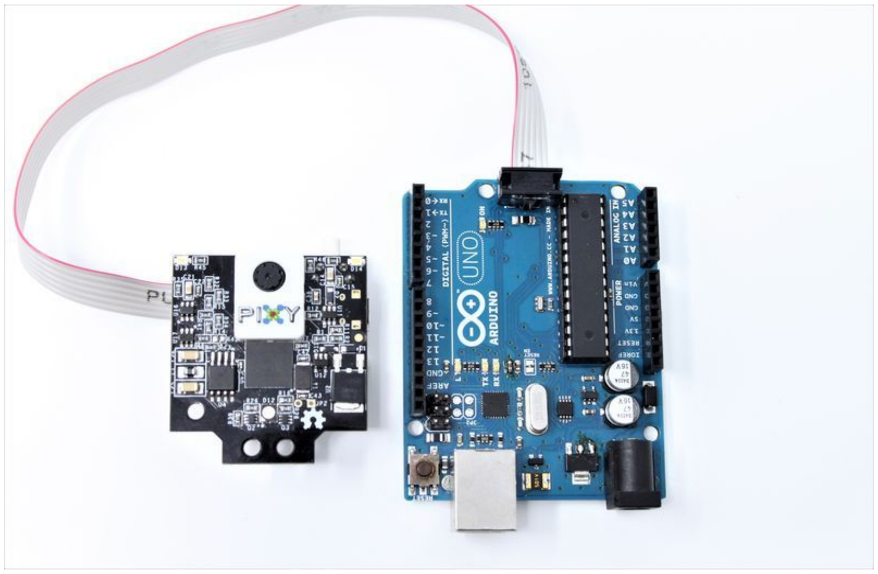
(https://components101.com/motors/mg90s-metal-gear-servo-motor, 2010)

C

Pixy2

Pixy2 is smaller, faster and more capable than the original Pixy. Like its predecessor, Pixy2 can learn to detect objects that you teach it, just by pressing a button. Additionally, Pixy2 has new algorithms that detect and track lines for use with line-following robots. The new algorithms can detect intersections and “road signs” as well. The road signs can tell your robot what to do, such as turn left, turn right, slow down, etc. And Pixy2 does all of this at 60 frames-per-second, so your robot can be fast, too.

Connecting directly to Raspberry Pi



Pixy2 comes with a special cable to plug directly into an Arduino and a USB cable to plug into a Raspberry Pi, so you can get started quickly. Pixy2 has several interfaces (SPI, I2C, UART, and USB) and simple communications, so you get your chosen controller talking to Pixy2 in short order.

Object Recognition

The main feature of the Pixy2 is its ability to recognize and track objects. This can be accomplished by:

The Pixy2 uses a hue or color-based filtering algorithm to detect objects. So, objects to be detected should have a distinct color. It can be fine-tuned to some degree, but it is difficult to distinguish two objects with the same color.

In addition to the object's hue, the Pixy2 also uses a "region growing algorithm" to distinguish an object.

Training the Pixy2 to detect an object is done by assigning the object a "color signature". The device can be trained to memorize up to seven color signatures.

You can also train the Pixy2 to remember up to seven additional "color codes". A color code consists of two colors, a couple of pieces of colored tape can make a good color signature. This can be useful in robotics applications, you can place “color signature signs” at specific locations to allow your robot to take action once it “sees” them.

There are two ways to train your Pixy2 to recognize objects – manually and with PixyMon. All object recognition is performed in Color Connected Components mode.

Capturing Signatures Manually

The Pixy2 is capable of being trained in a stand-alone mode without being connected to any computer or microcontroller. All you need to do is provide a source of power for the camera.

The push-button on the top of the Pixy2 is used along with the RGB LED to train the device.

Power up the Pixy2 and observe the status of the RGB LED, located near the bottom on the front of the camera. This indicator is the key to training the device manually.

When the camera is powered up the RGB LED will go through a series of flashes. This is the initialization sequence and you’ll need to wait while it runs (it only takes a few seconds). After initialization the indicator will turn off. You are now ready to train the Pixy2.

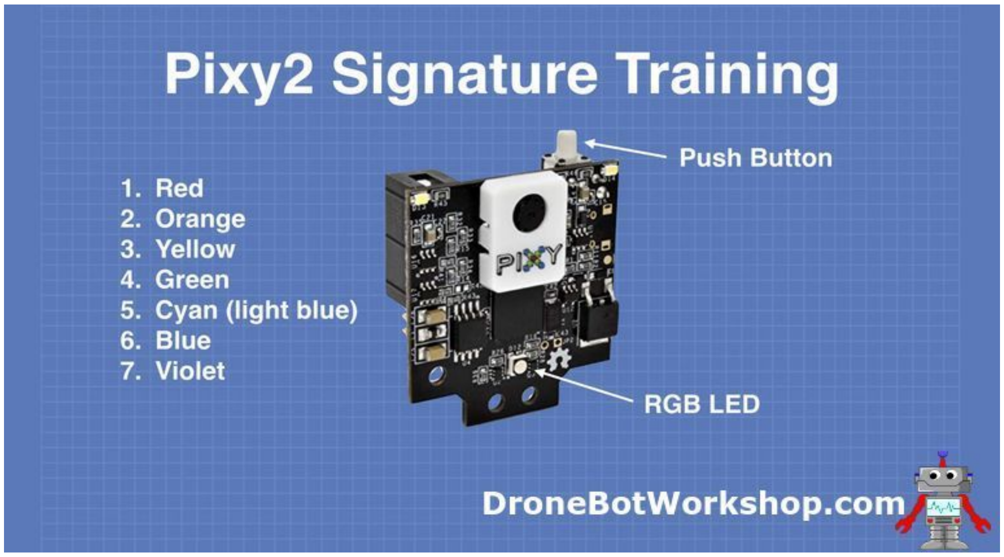
Find a suitable object to train the Pixy2 to detect. A good candidate object will have a distinct color.

While you can train the Pixy2 in stand-alone mode it is a good idea to have PixyMon running when you are new to setting color signatures. The monitor will make it easier to be sure you have your camera locking onto the correct object. Once you become more adept at training the device you can dispense with PixyMon and do your training in stand-alone mode.

So, with your object in place and PixyMon running to monitor your progress, it’s time to train the Pixy2 to detect its first object!

Start by pressing down and holding the push button. After about a second the RGB LED should glow white. Continue to hold down the button until it glows red, then immediately release the push button.

The red indicates that you are setting the color signature for signature number 1. If you were to continue holding down the button it will cycle through seven different colors, each one representing one of the color signatures.



Once you release the push button for signature number 1 (or any of the other six signatures) the Pixy2 will be in “light pipe” mode. The RGB LED will now glow in a color that approximates the color of the object you are trying to train the camera to recognize.

Observe the RGB LED and the video screen on PixyMon.

When the object is picked up by the Pixy2 you will see a grid pattern engulfing it. Move the object around until the grid covers it completely, or as much as possible, without picking up on extraneous objects (like your fingers for example). At that point, the RGB LED should be illuminated in a color like the target object.

Once you are satisfied that you are locked onto the object press and release the push button. The object will now be assigned to the color signature you were training it for.

You have now trained the Pixy2 to recognize its first object!

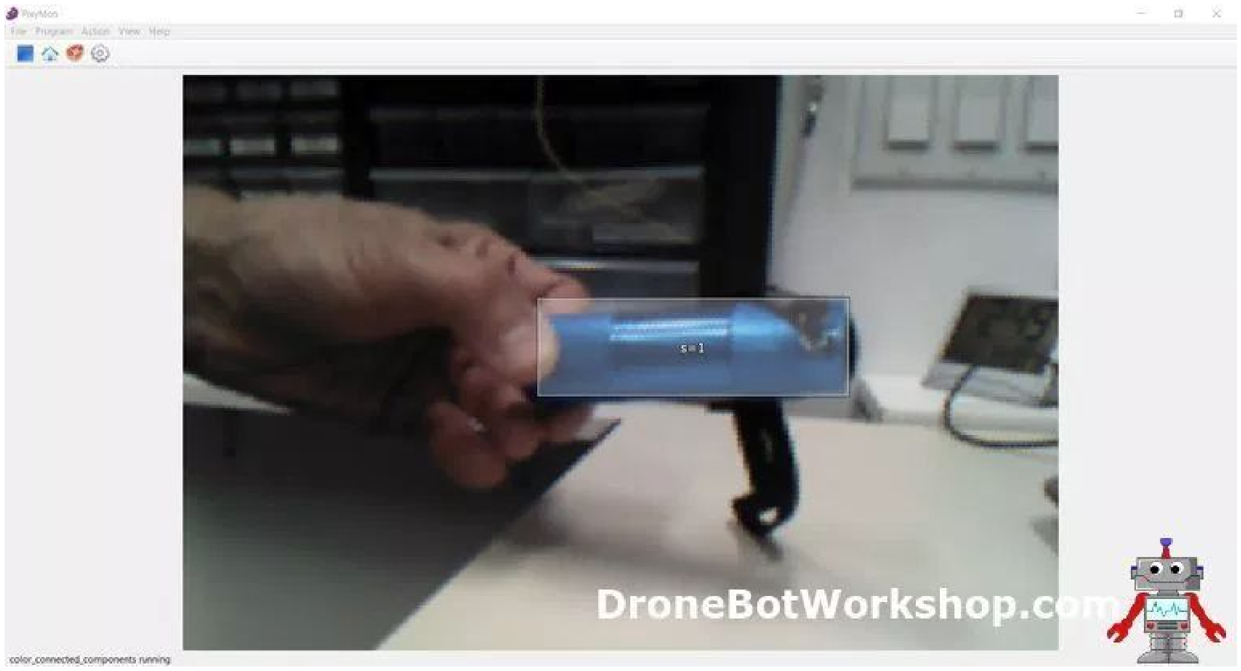
Capturing Signatures with PixyMon

Another way to train the Pixy2 to recognize objects and assign signatures to them is to use PixyMon.

Once again, you'll need to place the object you want to recognize in front of the camera. In PixyMon be sure you are in Color Connected Components mode, which is the default mode when you first start the program.

Once the object is visible in your monitor go to the *Action* menu and choose one of the "Set Signature" selections, i.e. *Set Signature 1*.

Now use your mouse and click and drag to outline the object you are trying to recognize.


PixyMon
File Program Action View Help
s=1
DroneBotWorkshop.com
color_connected_components running

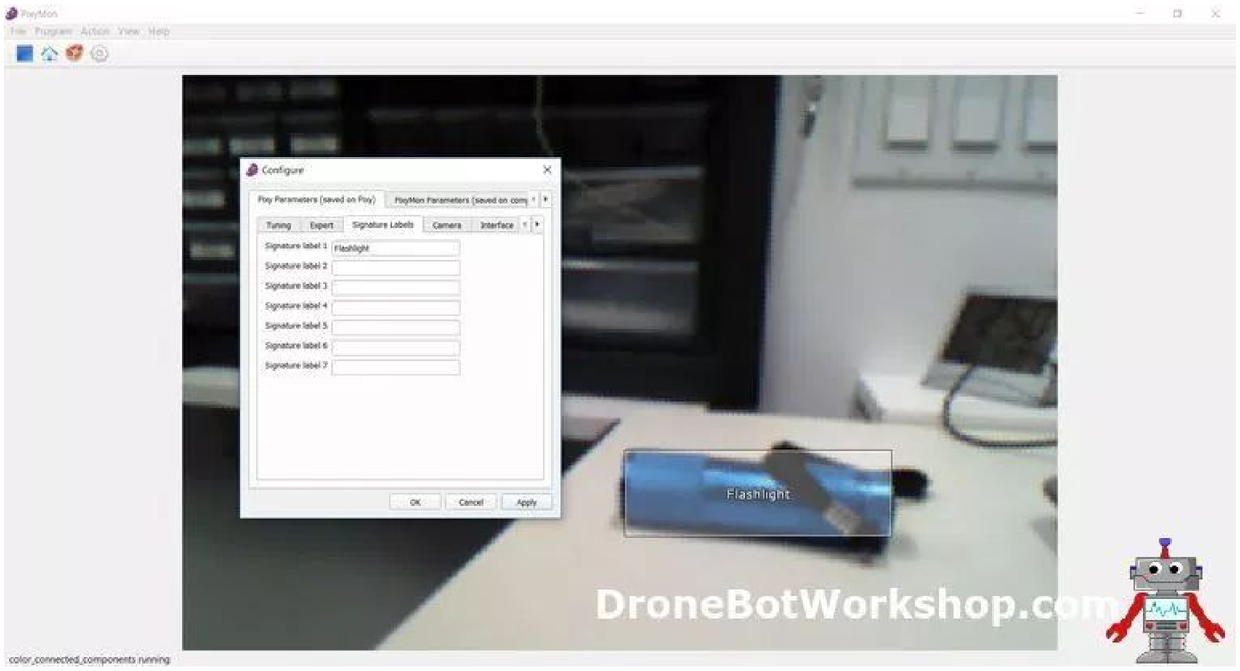
Once you have selected the area, release the mouse button. PixyMon will now have learned the object. It's as simple as that!

Labels and Fine Tuning

When you train the Pixy2 to recognize an object it will be displayed in PixyMon with a label that indicates its signature number. For example, the object assigned to signature number 1 will have “s=1” printed in its display block.

You may also want to “fine tune” the camera to recognize your object better or to prevent similar colored objects from falsely registering.

To fine tune the Pixy2 or change the signature labels open the *File* menu and choose *Configure*. This will open the configuration menu.


PixyMon
File Program Action View Help
Configure
Pixy Parameters (saved on Pixy)
PixyMon Parameters (saved on com
Tuning
Expert
Signature Labels
Camera
Interface
OK
Cancel
Apply
Flashlight
DroneBotWorkshop.com
color_connected_components running

Signature label 1 Flashlight
Signature label 2
Signature label 3
Signature label 4
Signature label 5
Signature label 6
Signature label 7

The configuration menu has two main tabs. One tab (the one that is active by default) allows you to configure settings on the Pixy2 itself, the second tab is for configuring PixyMon.

Within the *Pixy Parameters* tab there are several sub-tabs:

- **Tuning** – This tab has several sliders that allow you to “fine tune” the signatures. It is very useful to prevent false triggering. The sliders are quite sensitive, so a little bit goes a long way!
- **Expert** – As the name implies this tab has several advanced parameters.

- **Signature Labels** – This is the tab we are looking for. Here you can assign proper names to the objects you have trained your Pixy2 to detect.
- **Camera** – Parameters specific to the camera on the Pixy2.
- **Interface** – This allows you to set the default interface (i.e. I2C, SPI) for the Pixy2. Note that the micro USB port can still be used for PixyMon regardless of the interface settings.
- **Servos** – Parameters for the two servo motors used in the optional pan and tilt assembly.(www.PixyCam.com 2019/07/10)

Installation

Raspberry Pi

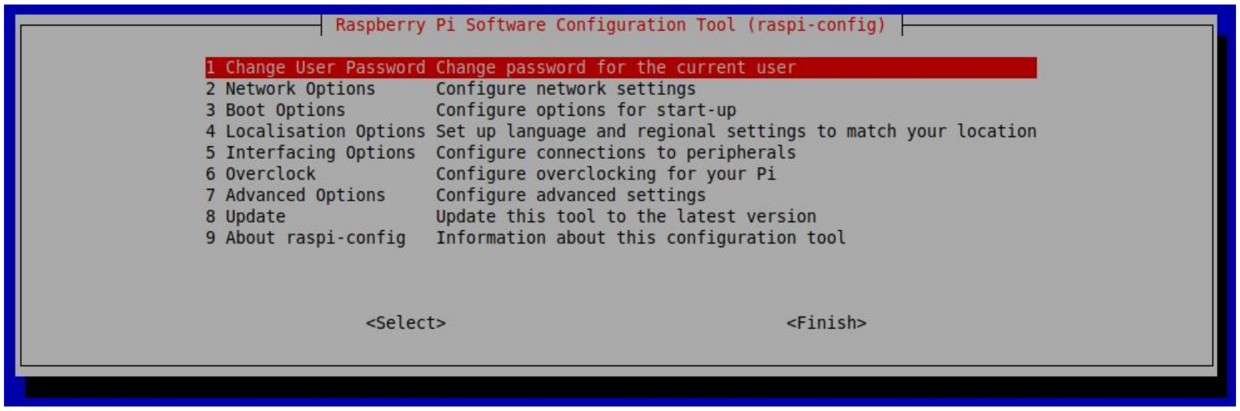
The Raspberry Pi must be fresh and run Raspbian Lite which is a Debian Linux distribution without GUI. That means that everything must be done in the console. The device should have the following services running on startup:

- Wireless network access point
- Web server
- SSH server
- Command parser for the robot arm

Enable I2C step by step

1. Enter a command and write raspi-config

   You will see the following screen



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password Change password for the current user
2 Network Options      Configure network settings
3 Boot Options         Configure options for start-up
4 Localisation Options Set up language and regional settings to match your location
5 Interfacing Options  Configure connections to peripherals
6 Overclock            Configure overclocking for your Pi
7 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

                 <Select>                       <Finish>
```

   (https://www.raspberrypi.org/documentation/configuration/raspi-config.md , 2013)

2. Choose number 5(Interfacing Options)
3. Inside the interface option you should enable I2C option
4. Select "Finish" to save changes

Wireless access point

The wireless access point is needed for stable independent connection. The connection has to be there even if the robot is in a different environment. The access point's SSID is ***"NHLStendenRobot"*** and the password is ***"12345678"***.

Web server: The web server is serving a website on which the users can control the robot arm with sliders. The website is written in HTML and JavaScript and consumed by Python Flask App. The JavaScript part checks for changes in slider values. If there is a change it writes it to a file.

SSH server: The SSH server is necessary for development, since we won't have access to a screen and an external keyboard all the time.

Command parser: The command parser is parsing the output from the web server and processes it to the *Adafruit-PCA9685* motor controller which directly controls the servos on the robot arm. The parser is written in Python 3. The basic structure of the parser looks like this:

1. The web server writes the output into a file.
2. The parser checks if there's anything new in the file.
3. If there's nothing new, it waits some time.
4. If there is, it parses it and sends it to the corresponding servos through the motor controller.

Connecting one Raspberry Pi to two networks simultaneously with two network adapters

Prerequisites:

1. Operating System must be Raspbian

Steps:

1. Connect a second Wi-Fi adapter via USB

2. In ***/etc/network/interfaces/wpa-conf.conf*** refer to two different WPA configuration files:

```
iface wlan0 inet auto
wpa-conf /etc/wpa_supplicant/wpa_supplicant_wlan0.conf
```

```
iface wlan1 inet auto
wpa-conf /etc/wpa_supplicant/wpa_supplicant_wlan1.conf
```

3. The content of each of those files should look like this:

```
country=NL
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
  ssid="NHLStendenRobot"
   psk="12345678"
}
```

4. For this to work the two networks must have different local nets.

SSH Server

Step 1: Enable SSH on Raspberry Pi

SSH is disabled by default in Raspberry Pi, hence you'll have to enable it when you turn on the Pi after a fresh installation of Raspbian.

1. Enter `sudo apt install openssh`
2. Use systemctl to enable and start the service
   - `sudo systemctl enable sshd`
   - `sudo systemctl start sshd`

Step 2. Find the IP Address of Raspberry Pi

You will need to note down the IP address of your Pi in order to connect to it later. Using the ifconfig command will display information about the current network status, including the IP address, or you can use hostname -I to display the IP addresses associated with the device.

Step 3. SSH into your Raspberry Pi

Now that you have enabled SSH and found out your IP address you can go ahead and SSH into your Raspberry Pi from any other computer. You'll also need the username and the password for the Raspberry Pi. In this case it was used OpenSSH as the client.

Default Username and Password is:

- username: pi
- password: raspberry

Making a Raspberry Pi into an Access Point

Follow this tutorial:

https://thepi.io/how-to-use-your-raspberry-pi-as-a-wireless-access-point/

Or paste these commands:

```
sudo apt-get install hostapd
sudo apt-get install dnsmasq
sudo systemctl stop hostapd
sudo systemctl stop dnsmasq
sudo nano /etc/dhcpcd.conf
```

add these lines at end of file:

```
interface wlan0
static ip_address=192.168.0.10/24
denyinterfaces eth0
denyinterfaces wlan0

sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
sudo nano /etc/dnsmasq.conf
```

add these lines in file:

```
interface=wlan0
```

```
 dhcp-range=192.168.0.11,192.168.0.30,255.255.255.0,24h
sudo nano /etc/hostapd/hostapd.conf
```

add these lines to file:

```
interface=wlan0
bridge=br0
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
ssid=NETWORK
wpa_passphrase=PASSWORD
```

```
sudo nano /etc/default/hostapd
```

In this file, track down the line that says #DAEMON_CONF=”” – delete that **#** and put the path to our config file in the quotes, so that it looks like this:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
sudo nano /etc/sysctl.conf
```

uncomment this line:

```
#net.ipv4.ip_forward=1
```

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

edit the file /etc/rc.local

```
iptables-restore < /etc/iptables.ipv4.nat
```

```
sudo apt-get install bridge-utils
sudo brctl addbr br0
sudo brctl addif br0 eth0
sudo nano /etc/network/interfaces
```

add these lines at end of file:

```
auto br0
iface br0 inet manual
bridge_ports eth0 wlan0
```

Reboot.

Using the install.sh

Installing PyTorch on Windows

PyTorch can be installed and used on various Windows distributions. Depending on your system and compute requirements, your experience with PyTorch on Windows may vary in terms of processing time. It is recommended, but not required, that your Windows system has an NVIDIA GPU in order to harness the full power of PyTorch's CUDAsupport.

PREREQUISITES

Supported Windows Distributions

PyTorch is supported on the following Windows distributions:

- Windows 7 and greater; Windows 10 or greater recommended.
- Windows Server 2008 r2 and greater

Python

Currently, PyTorch on Windows only supports Python 3.x; Python 2.x is not supported.

As it is not installed by default on Windows, there are multiple ways to install Python:

- Python website
- Anaconda

Package Manager

To install the PyTorch binaries, you will need to use at least one of two supported package managers: Anaconda and pip. Anaconda is the recommended package manager as it will provide you all the PyTorch dependencies in one, sandboxed install, including Python and pip.

Installation

Pip: *pip3 install torch===1.4.0 torchvision===0.5.0 -f https://download.pytorch.org/whl/torch_stable.html*

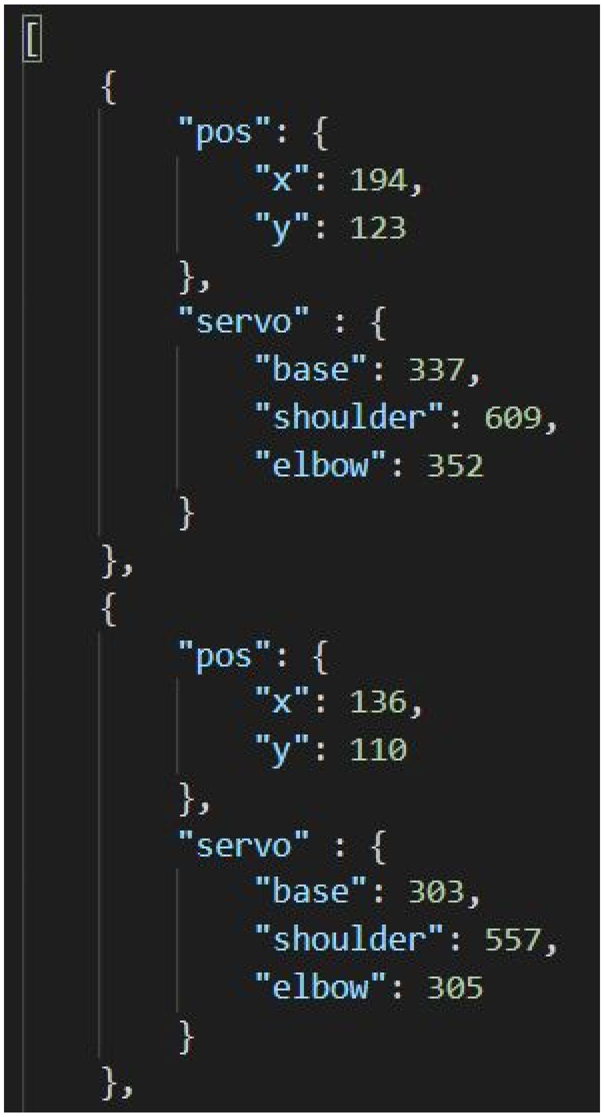
Anaconda: *conda install pytorch torchvision cudatoolkit=10.1 -c pytorch*

Machine learning

Create data set as json file:

You need to move the robot arm to X,Y position and note the coordinates ,

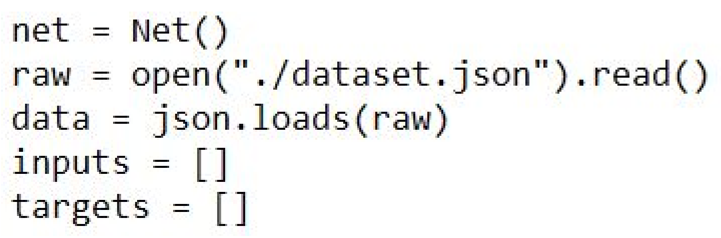
After that note the servos value and write it in the json file like the json example you can find the full document in the appendix 3



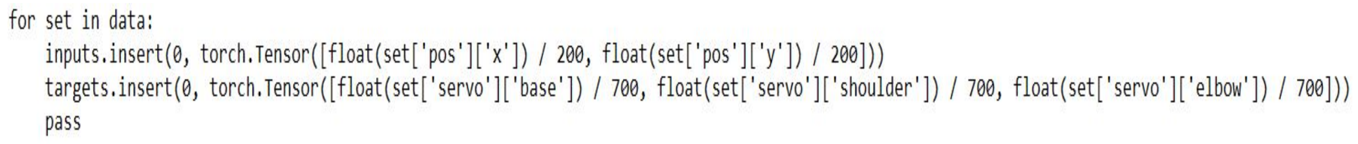
```json
[
    {
        "pos": {
            "x": 194,
            "y": 123
        },
        "servo" : {
            "base": 337,
            "shoulder": 609,
            "elbow": 352
        }
    },
    {
        "pos": {
            "x": 136,
            "y": 110
        },
        "servo" : {
            "base": 303,
            "shoulder": 557,
            "elbow": 305
        }
    },
```

Json example

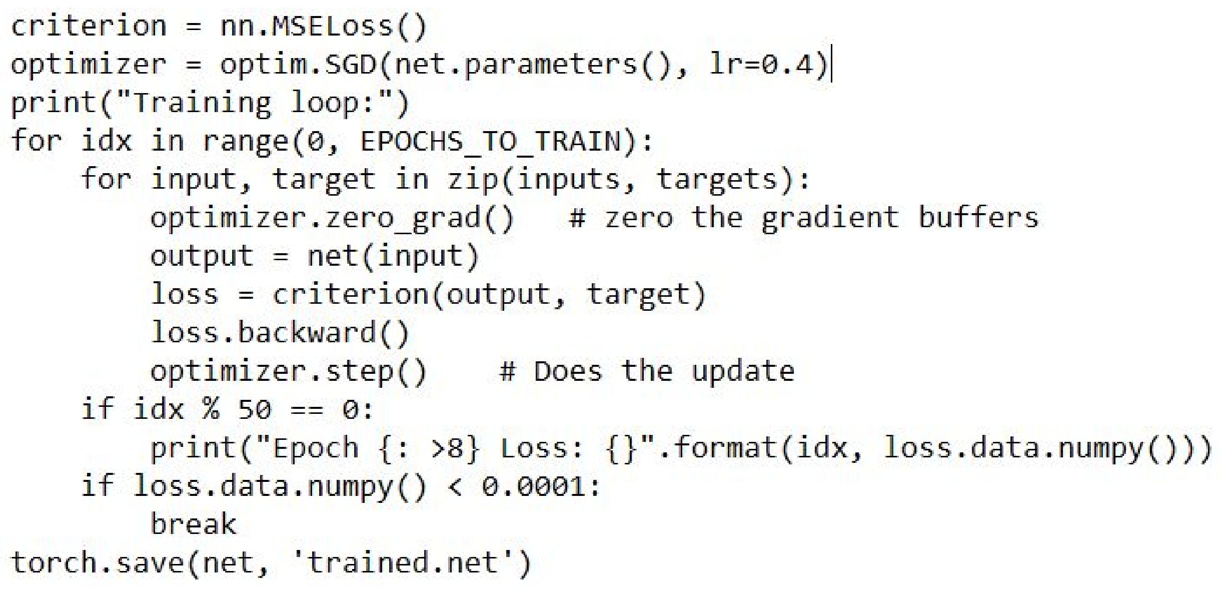
1. Create a Python Script to use the data set and train the robot arm to get the exact value of the servos to move the arm . see appendix 4



```
net = Net()
raw = open("./dataset.json").read()
data = json.loads(raw)
inputs = []
targets = []
```



```
for set in data:
    inputs.insert(0, torch.Tensor([float(set['pos']['x']) / 200, float(set['pos']['y']) / 200]))
    targets.insert(0, torch.Tensor([float(set['servo']['base']) / 700, float(set['servo']['shoulder']) / 700, float(set['servo']['elbow']) / 700]))
    pass
```



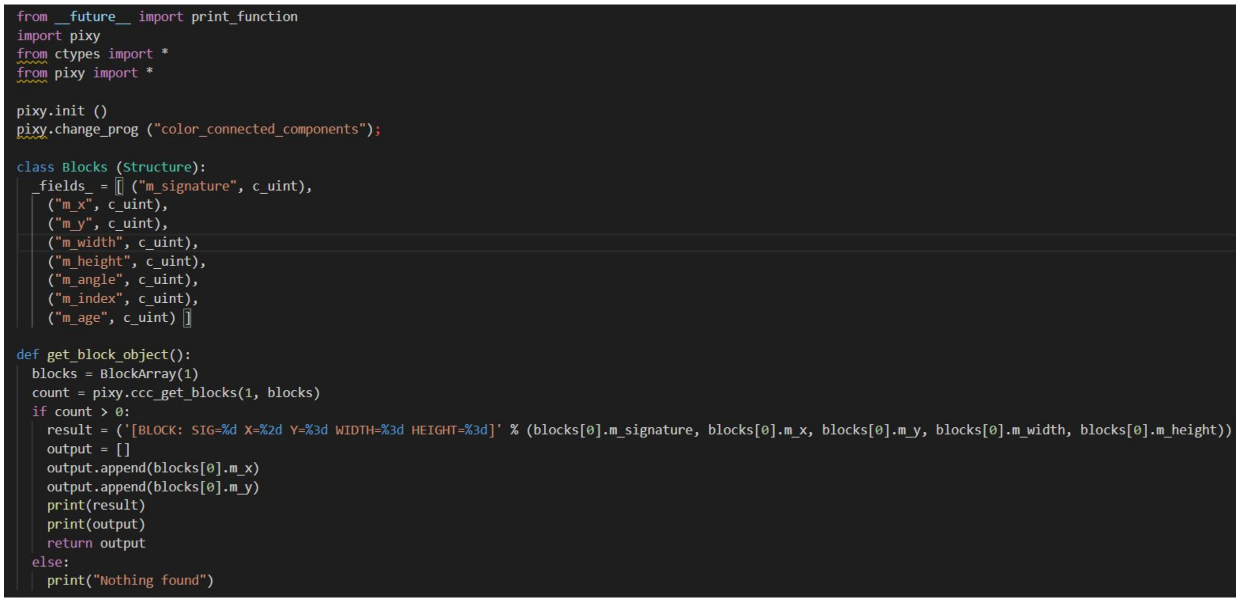
```
criterion = nn.MSELoss()
optimizer = optim.SGD(net.parameters(), lr=0.4)
print("Training loop:")
for idx in range(0, EPOCHS_TO_TRAIN):
    for input, target in zip(inputs, targets):
        optimizer.zero_grad()   # zero the gradient buffers
        output = net(input)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()    # Does the update
    if idx % 50 == 0:
        print("Epoch {: >8} Loss: {}".format(idx, loss.data.numpy()))
    if loss.data.numpy() < 0.0001:
        break
torch.save(net, 'trained.net')
```

2. The first image, loads the json dataset into memory.
3. The second, create two Tensor arrays from the dataset.
4. The last one is the actual learning algorithm with backpropagation
    a. In every loop the optimizers zeroes the buffers, then we run the model and get the output, we compare the output to the target and run the built-in backward-propagation algorithm. After that we print out the information about the training and check if the model is accurate enough for stopping it prematurely.
5. The output will be **trained with 150,000 iteration**.

Integrating

1. Capturing signature of an object like we had explained in the pixyCam chapter.

2. Create a Python script to get the coordinates of an object from the pixyCam . see appendix 5



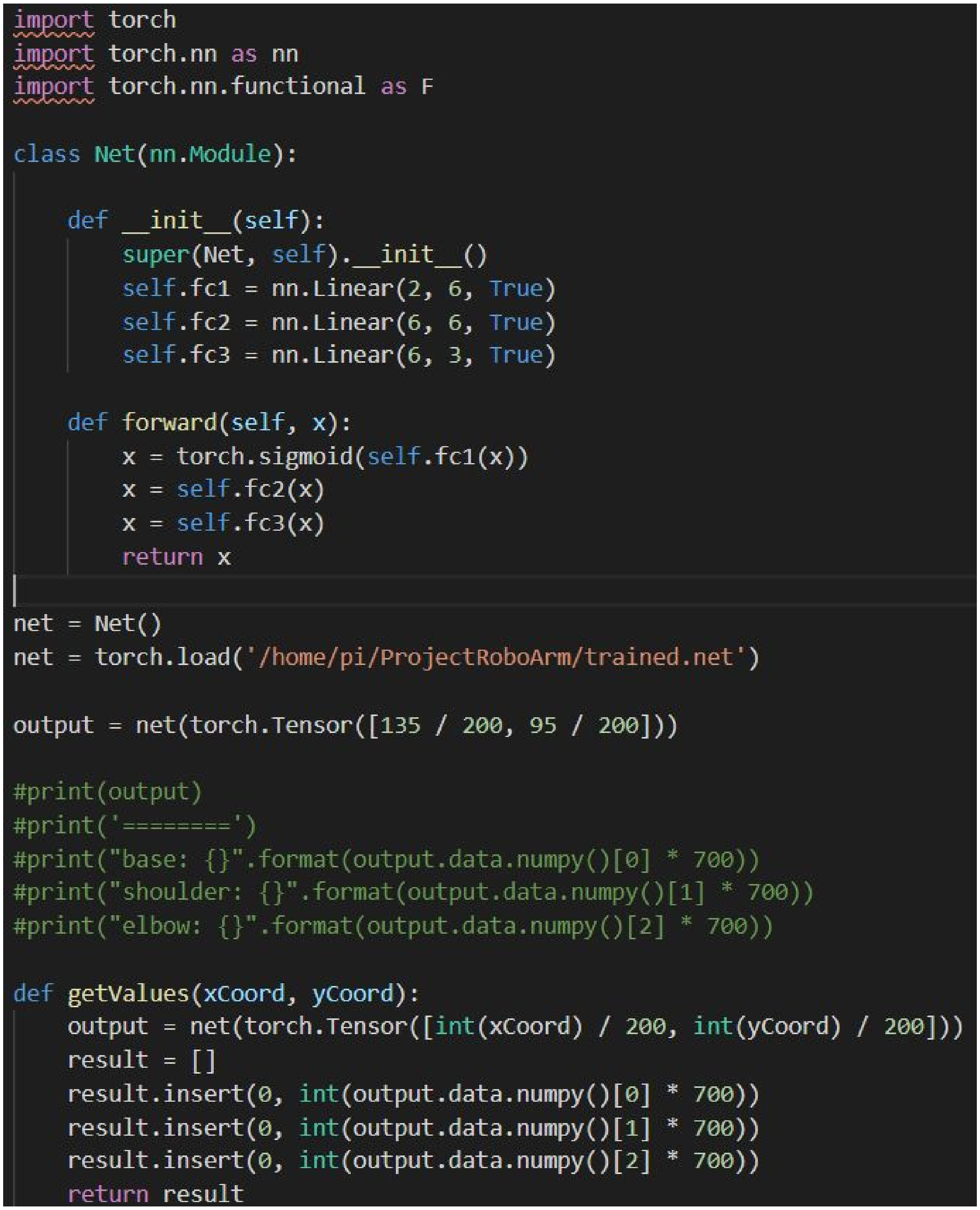
```python
from __future__ import print_function
import pixy
from ctypes import *
from pixy import *

pixy.init ()
pixy.change_prog ("color_connected_components");

class Blocks (Structure):
  _fields_ = [ ("m_signature", c_uint),
    ("m_x", c_uint),
    ("m_y", c_uint),
    ("m_width", c_uint),
    ("m_height", c_uint),
    ("m_angle", c_uint),
    ("m_index", c_uint),
    ("m_age", c_uint) ]

def get_block_object():
  blocks = BlockArray(1)
  count = pixy.ccc_get_blocks(1, blocks)
  if count > 0:
    result = ('[BLOCK: SIG=%d X=%2d Y=%3d WIDTH=%3d HEIGHT=%3d]' % (blocks[0].m_signature, blocks[0].m_x, blocks[0].m_y, blocks[0].m_width, blocks[0].m_height))
    output = []
    output.append(blocks[0].m_x)
    output.append(blocks[0].m_y)
    print(result)
    print(output)
    return output
  else:
    print("Nothing found")
```

3. Create a Python script to get the value of the servos by using the training data we have “trained.net” inserting the coordinates from the pixyCam 2 and get the values. see appendix 6



```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 6, True)
        self.fc2 = nn.Linear(6, 6, True)
        self.fc3 = nn.Linear(6, 3, True)

    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = self.fc2(x)
        x = self.fc3(x)
        return x

net = Net()
net = torch.load('/home/pi/ProjectRoboArm/trained.net')

output = net(torch.Tensor([135 / 200, 95 / 200]))

#print(output)
#print('========')
#print("base: {}".format(output.data.numpy()[0] * 700))
#print("shoulder: {}".format(output.data.numpy()[1] * 700))
#print("elbow: {}".format(output.data.numpy()[2] * 700))

def getValues(xCoord, yCoord):
    output = net(torch.Tensor([int(xCoord) / 200, int(yCoord) / 200]))
    result = []
    result.insert(0, int(output.data.numpy()[0] * 700))
    result.insert(0, int(output.data.numpy()[1] * 700))
    result.insert(0, int(output.data.numpy()[2] * 700))
    return result
```

4. Create a Python script where the results from AI will be handled to implement the servos values. See appendix 7.

5. Create the main Python Flask script that will handle the request from the website and integrate them with the AI and servos in order to move the arm of the robot to the coordinates. see appendix 8

Steps to follow to start the robot arm

1. Start the raspberry PI.
2. Connect via wireless to NHLStendenRobot using the password '12345678'
3. Open the browser and type (192.168.0.10:8181)

Now the Robot arm is ready to be used manually and automatically. Please note that there can only be one connection to the robot at a time.

Software Installation guide

First of all you need to set up the Pi based on the previous chapters:

- I2C, access point, SSH
- Make sure there is internet connection to the device (ping google or something)

Once done, run the following commands:

```
sudo apt update
sudo apt install git
git clone https://github.com/TomiEckert/ProjectRoboArm.git
cd ProjectRoboArm
chmod +x install.sh
sudo ./install.sh
```

The device should reboot afterwards. Once it booted up again, connect to the access point and go to 192.168.0.10:8181 in your browser.

IMPORTANT:

- If the website does not load, open /home/pi/ProjectRoboArm/app.py
- Put a '#' in front of the line: "import PixyBlockDetectorThingy"
- Save the file
- Wait 5-10 seconds
- Remove the '#' character
- Repeat until the website loads

Appendix

Running two WiFi adapters on the one Pi - Raspberry Pi Forums. (n.d.). Retrieved from https://www.raspberrypi.org/forums/viewtopic.php?t=190525

Lovely, P. B. S. (2018, September 19). How to use your Raspberry Pi as a wireless access point. Retrieved from https://thepi.io/how-to-use-your-raspberry-pi-as-a-wireless-access-point/

Connecting RPi to 2 different networks. (2018, June 7). Retrieved from https://raspberrypi.stackexchange.com/questions/84748/connecting-rpi-to-2-different-networks