Tetrix Vision Robot with Python and OpenCV

Eric Liu, CJ Chung

Table of Contents

1. Tetrix Vision Robot - EduBot

1.1 Why Tetrix Platform?

1.2 Building the Tetrix Vision Robot - EduBot

2. PRIZM Controller

2.1 What is PRIZM?

2.2 Setting up Program Environment - Arduino IDE

2.3 Getting Started with PRIZM

2.4 DC Motor Control with PRIZM (Arduino sketch driven program)

3. Python for EduBot Programming

3.1 Why Python?

3.2 Python and PRIZM Communication

3.3 DC Motor Control with Python

4. EduBot Programming with Vision

4.1 Python OpenCV Introduction

4.2 OpenCV GUI Featueres

a) Getting Started with Images

b) Getting Started with Videos

c) Drawing Functions in OpenCV

d) Trackbar for Threshold

4.3 Image Processing in OpenCV

a) Image Thresholding

b)White Pixel Counting

c) Colored Pixel Counting

d)Changing Colorspaces

4.4 Follow a White Line

4.5 Follow an Edge

References

Acknowledgments

Appendix A - How to Install Arduino IDE and PRIZM Arduino Libarary

Appendix B - How to Install Python, OpenCV

Appendix C - EduBot (Model B) Building Instructions

1. Tetrix Vision Robot - EduBot

1.1 Why Tetrix Platform?

The Tetrix building system is one of the most widely used robotics building system. It is also one of the building systems for First Tech Challenge (FTC), a robotics challenge for middle and high schoolers in the United States. The features of this building system include:

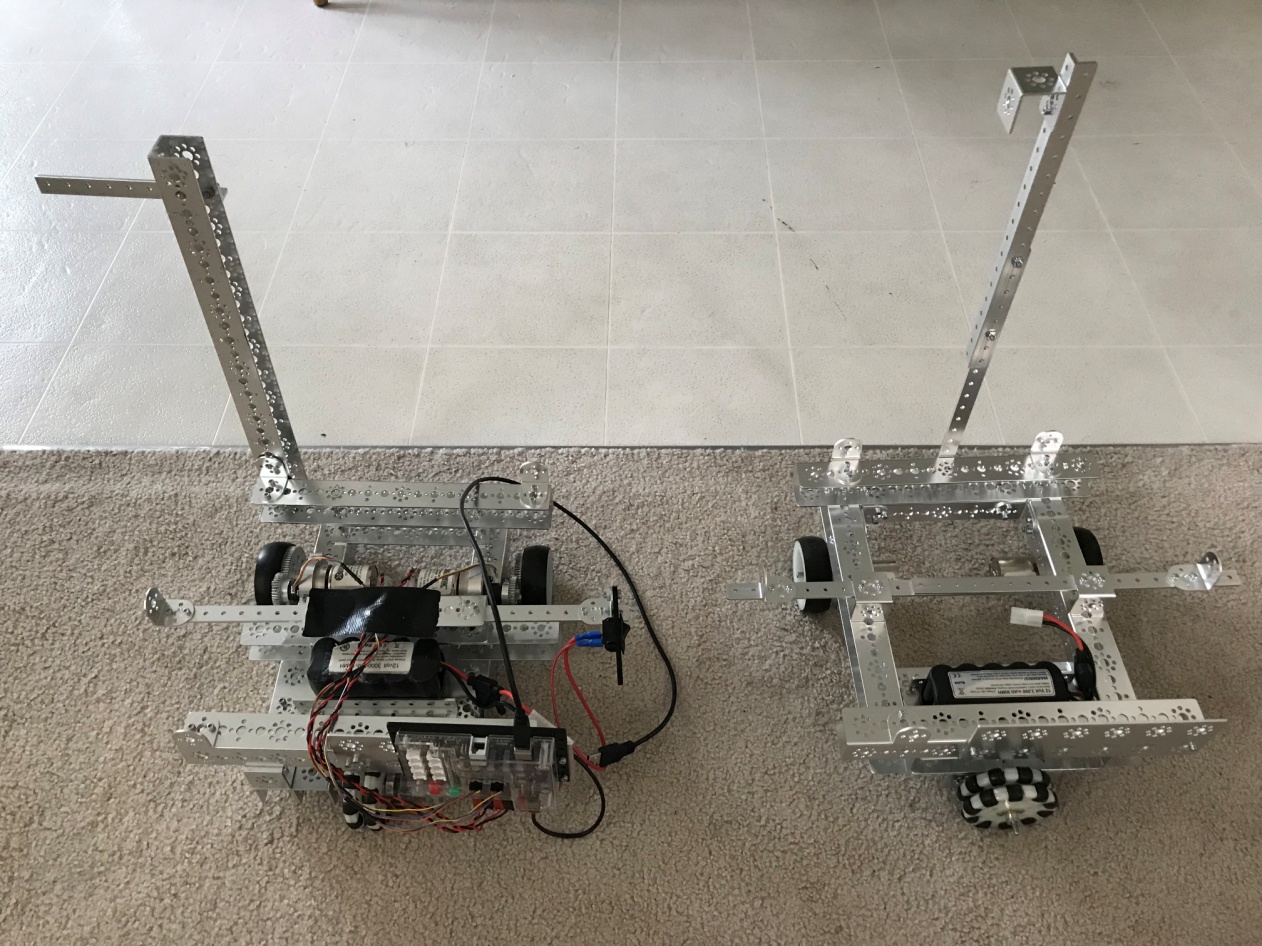
* Uses elements made from heavy-duty, aircraft-grade aluminum to maximize stability and reliability
* Uses powerful drive motors that drastically increase the capabilities of TETRIX robots
* Offers flexibility in build design and can be expanded using additional materials
* Gives users the opportunity to master the concepts of wiring, motor control, and much more
* Fosters creativity and ingenuity amongst students
* Develops engineering and problem-solving skills
* Can be extended for vision and machine learning applications

1.2 Building the Tetrix Vision Robot - EduBot

Our purpose in this e-book is to develop a Tetrix Vision Robot. This robot will be able to hold a laptop with Python and OpenCV, and mount a webcam as a sensor. We will call this robot EduBot because this robot can be used for general purpose STEM education as well as for more advanced robotics learning such as vision and machine learning. Some of the considerations we had while building the robot were:

* able to hold the laptop and camera
* castor bot structure for enhanced maneuverability
* able to move slowly and precisely
* easy access to battery and controller
* very sturdy and reliable structure
* wires and cables all neatly tucked away

We developed two versions of the EduBot, Model A and Model B as shown in picture one.

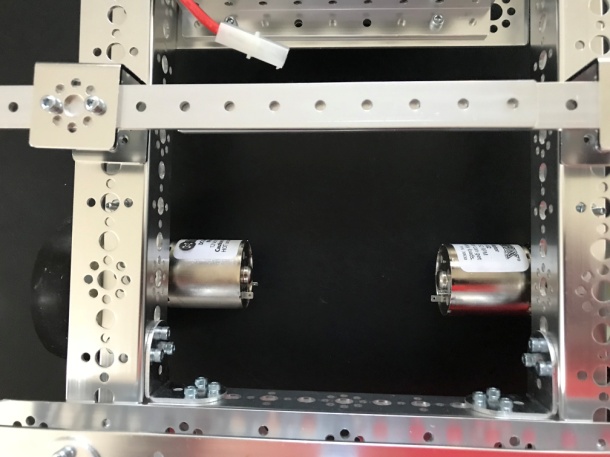
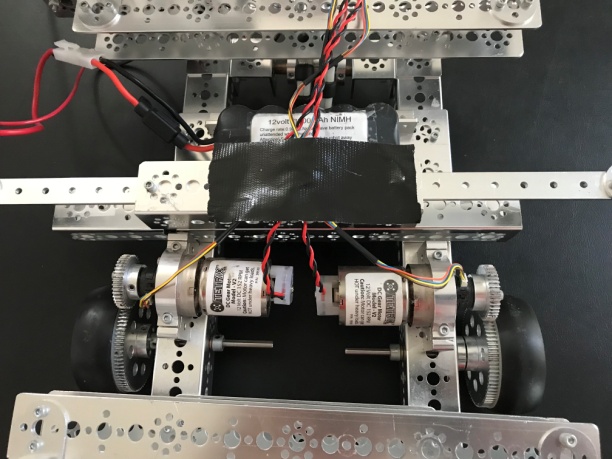


**Tetrix Vision Robot -EduBot**

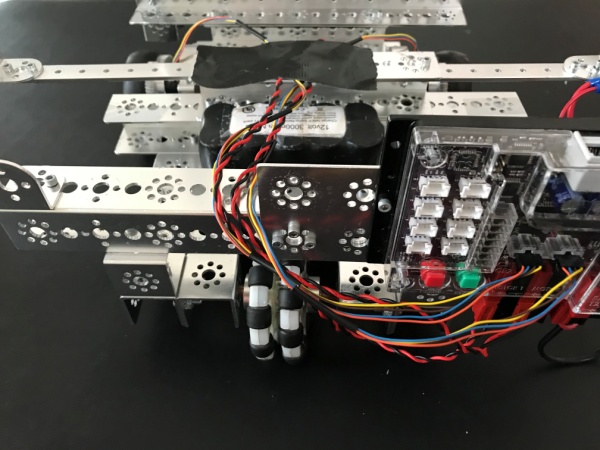
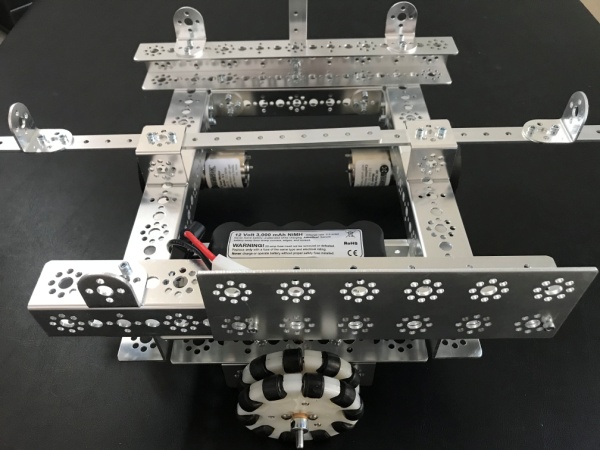
**Model B**

**Model A**

Picture 1.1

Picture 1.2a Picture 1.2b

Picture 1.3a Picture 1.3b

The main differences between the two models are:

a) model A uses a speed reduction gear system while model B has the wheel attached straight onto the axle, model A can move slower and more controlled but the exposed gears pose a safety hazard (see pictures 1.2a and 1.2b)

b) the rear wheel (omni wheels) in Model B is mounted parallel to the back channel to allow for better lateral maneuverability (see pictures 1.3a and 1.3b)

c) model B is easier to build and requires less building parts

Detailed building instructions for Model B can be found in Appendix C of this e-book.

2. PRIZM Controller

(The content in this section is edited based on Pitsco's "tetrix-prizm-programming-guide.pdf". Pls. check the document on the Pistsco's website if you need more detailed information )

2.1What is PRIZM?

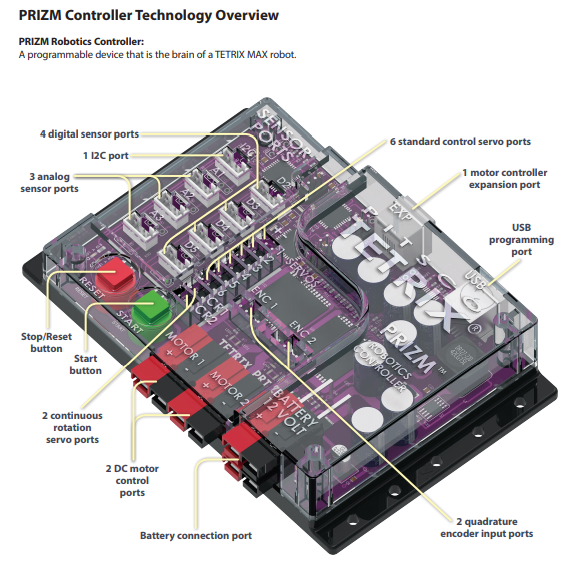


Fig2.1 PRIZM Controller

TETRIX® PRIZM® is a programmable target designed to enable autonomous control of robotics and mechatronic systems constructed using TETRIX MAX building elements. The design architecture of PRIZM is Arduino compatible, using the same ATmega328P program memory chip and preprogrammed Optiboot bootloader as a genuine Arduino UNO.

PRIZM offers a fully integrated brain for your robot that is powerful yet easy to use and, when combined with the Arduino Software (IDE), can provide an ideal learning tool for teaching engineering design and coding. The TETRIX PRIZM Robotics Controller features the following:

• 2 high-current DC motor control ports, 6 standard control servo ports, and 2 continuous rotation (CR) servo ports

• 4 digital sensor ports (D2 can be used as a serial port; D2-D5 can be configured as digital input or output) and 3 analog sensor ports (A1-A3 can be configured as analog input or digital input or output)

• 1 I2C port, 1 USB programming port, 1 motor controller expansion port, and 2 quadrature encoder input ports

• 2 battery connection ports (additional port used to daisy-chain power to other devices)

2.2 Setting up Program Environment - Arduino IDE

For PRIZM software setup, we need to do two things:

(1) Install the Arduino Software (IDE)

The Arduino Integrated Development Environment (IDE) is open-source software that compiles code and sends it to the hardware. As a text-based and developer friendly software with a huge existing user base, the Arduino Software (IDE) is supported for a variety of Windows, Macintosh, and Linux devices. This makes the Arduino Software (IDE) flexible, classroom friendly, and competition ready and offers all users, even the hobbyist, a low entry and high ceiling. All these factors make it perfect for beginner and veteran coders alike. The Arduino Software (IDE) uses a C-based programming language to communicate with the PRIZM. Within the Arduino Software (IDE) an individual program is referred to as a sketch. Each of the activities in this guide will involve creating a sketch that gives instructions to the robot.

(2) Installing the PRIZM Controller Arduino Library.

Adding custom libraries can expand the usability of the Arduino Software (IDE). Libraries are collections of code that make it easier to create a program, or, as Arduino calls it, a sketch. After you have successfully installed the Arduino Software (IDE), you can add the Arduino PRIZM controller library. The PRIZM controller library contains special functions written for the TETRIX PRIZM controller.

**Detailed software installation steps can be found in Appendix A of this e-book.**

2.3 Getting Started with PRIZM

We will use a simple sketch to get familiar with PRIZM controller by blinking the onboard PRIZM red LED. This sketch is the PRIZM equivalent of a Hello World! program, which is usually the intro activity for any beginning programmer. The sketch we will create is one of the simplest and most basic PRIZM functions and requires only the PRIZM, a power source, and a USB connection to the computer.

Here are what we need:

• PRIZM controller

• USB cable

• A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack

• PRIZM Controller On/Off Battery Switch Adapter

• Computer

**Important Safety Information: Caution: Use only a TETRIX battery pack that is equipped with an in-line safety fuse. Failure to do so could result in damage or injury. Connect the TETRIX battery pack to either the top or bottom red/black power inlet row at the battery connection port. Do not connect two battery packs to the PRIZM controller.**

The complete code is listed in code listing 2.1. You can type in the code in the IDE as a practice if time permits, or simply open it by selecting File > Examples > TETRIX\_PRIZM > GettingStarted\_Act1\_Blink\_RedLED. A new sketch window will open titled GettingStarted\_Act1\_Blink\_RedLED .

Code listing 2.1: PRIZM\_Blink\_RedLED

/\* PRIZM controller example program

\* Blink the PRIZM red LED at a 1-second flash rate.

\* author PWU on 08/05/2016

\*/

#include <PRIZM.h> // include the PRIZM library

PRIZM prizm; // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {

prizm.PrizmBegin(); // initialize the PRIZM controller

}

void loop() { // repeat this code in a loop

prizm.setRedLED(HIGH); // turn the red LED on

delay(1000); // wait here for 1,000 ms (1 second)

prizm.setRedLED(LOW); // turn the red LED off

delay(1000); // wait here for 1,000 ms (1 second)

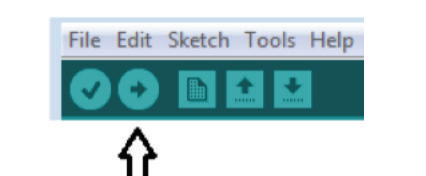
}

Check followings before we can upload the sketch to the PRIZM:

* make sure the PRIZM has power
* PRIZM is connected to the computer, and is detected bythe computer
* When the PRIZM is connected, power on the PRIZM with the on/off switch.

You will know the PRIZM has power by the glowing blue light. To see if the PRIZM is detected by the computer, check the port as we did in the Configuring USB Communication section.

To upload the sketch to the PRIZM, click Upload as shown below:



Press the Green button on the PRIZM controller to run the program

As hands on practices, you can change some parameters in our code and see how it affects the behavior of the red LED. According to the comments in the example, the delay function defines the duration the LED is on or off. This is a parameter we can change in our code. Experiment with changing those values to create new blinking behaviors for the LED. Try making the LED blink faster or slower.

With the example as a reference, you can try creating the blinking LED in a new sketch. Instead of just blinking the red LED, try to blink the green LED too. Flashing or blinking lights have a rich tradition as a method of signaling or long-distance communication. You could challenge yourself to communicate “Hello World!” in blinking Morse code.

2.4 DC Motor Control with PRIZM (Arduino sketch driven program)

For our second test, we want to keep things simple but add an element of motion. We will create a sketch that will rotate a TETRIX DC motor.

Here are what we need:

•TETRIX DC motor

• PRIZM controller

• USB cable

• A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack

• PRIZM Controller On/Off Battery Switch Adapter

• Computer

Code listing 2.2: PRIZM\_Move\_DCMotor

Example 2: GettingStarted\_Act2\_Move\_DCMotor

You can type in the code in the IDE as a practice if time permits, or simply open it by selecting File > Examples > TETRIX\_PRIZM > GettingStarted\_Act2\_Move\_DCMotor

/\* PRIZM controller example program

\* Spin DC Motor Channel 1 for 5 seconds and then coast to a stop.

\* After stopping, wait for 2 seconds and then spin in opposite direction.

\* Continue to repeat until red Reset button is pressed.

\* author PWU on 08/05/2016

\*/

#include <PRIZM.h> // include the PRIZM library in the sketch

PRIZM prizm; // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {

prizm.PrizmBegin(); // initialize the PRIZM controller

}

void loop() { // repeat in a loop

prizm.setMotorPower(1,25); // spin Motor 1 CW at 25% power

delay(5000); // wait while the motor runs for 5 seconds

prizm.setMotorPower(1,0); // stop motor (coast to stop)

delay(2000); // after stopping, wait here for 2 seconds

prizm.setMotorPower(1,-25); // spin Motor 1 CCW at 25% power

delay(5000); // wait while the motor runs for 5 seconds

prizm.setMotorPower(1,0); // stop motor (coast to stop)

delay(2000); // after stopping, wait here for 2 seconds and then repeat

}

3.Python for EduBot Programming

3.1 Why Python?

Python is a general-purpose language which is designed to be simple to read and write. It is

a versatile, easy to learn, and easy to use scripting language. Its power, and huge library of user-created modules make it an ideal language for a wide verity of computer side tasks.

A brief list of advantages of Python:

• Clear, readable syntax

• Obvious simplicity

• Quick progression

• Versatility

• Cross-Platform run ability

• Widely available resources

• Supportive community

• Widely available resources

• One of [the most popular languages used](https://dzone.com/articles/which-are-the-popular-languages-for-data-science) in data science, machine learning and AI systems

• Python is free

When it comes to robotics, here are some examples how Python is used:

* Python can be used to control GPIO pins in Raspberry Pi, or the pins on your Arduino (your Arduino robot is connected to PC) to control your physical robots, and these are two of the most popular robotic platforms.
* There are also other aspects of robotics where Python is very useful. You can use OpenCV for any image processing or computer vision related tasks. These are awesome libraries that allows you to use almost any popular image processing or computer vision algorithm using just a few lines of codes. This e-book will focus on this application in the next section.
* You can use machine learning libraries like TensorFlow, Scikit-learn to create a neural network using less than 20 lines of Python code. You can use any of the other scores of ML algorithms that come bundled with these libraries.
* The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. You can write the algorithm using Python for ROS and use the algorithm instantly on all sorts of robot (real or simulated) that ROS supports.

3.2 Python and PRIZM Communication

We will use USB port to establish Python and PRIOZM communication like Fig.3.1.

**See Appendix B for Serial Installation steps**

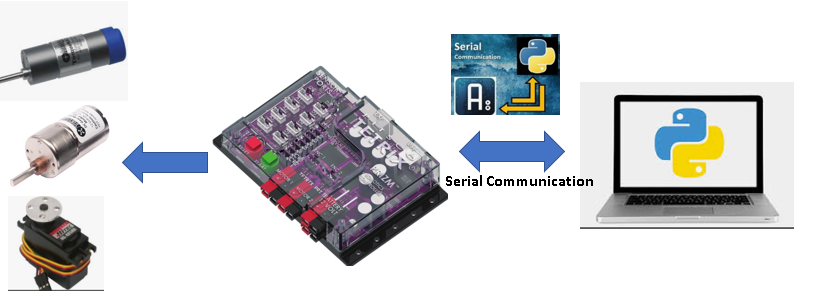


Fig.3.1 Python and PRIZM Communication

1. PRIZM side

The PRIZM USB port (SEE Fig.34) is used for communication between PRIZM and a

Windows, Mac, or Linux computer. Its main use is for downloading program code to the

program memory processor. It can also be used to transfer data between PRIZM and a

computer using serial communications protocol. For example, the Arduino Software (IDE)

includes a built-in serial monitor display window used to display data values received from

the PRIZM controller. This feature can be used to display sensor or encoder data or even

other types of program data as needed.

Oftentimes, viewing data values can be very helpful when debugging program code. The

serial monitor can also be used to send information to the PRIZM controller. An example of

this would be to have the PRIZM controller moving DC motors or servos in response to

keyboard entries. Users with advanced programming knowledge could create graphical

interfaces using Java, Python, or another coding platform for monitoring data or controlling

functions.

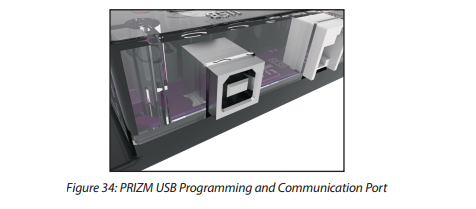


Figure 3.2: PRIZM USB Programming and Communication Port

1. Laptop/Python side:

On the Laptop/Python side, we will be using a Python module called [PySerial](http://pyserial.sourceforge.net/) which allows for the use of serial connections with Python.

(3) Steps for establishing the communication between Python and PRIZM:

1.Installing PySerial on Windows. This is a pretty simple process. To install on Windows, simply visit [PySerial's Download Page](http://pyserial.sourceforge.net/pyserial.html" \l "installation), download the Windows binary, and run it. Afterwards, test your installation by opening up a new instance of the Python interpreter, and running:

import serial

2. Initial Connection

To initiate a connection with the PRIZM from Python, we first have to figure out which *COM Port* the PRIZM is on. This task is luckily made simple by the Ardunio programming environment.

Simply look in the bottom right corner of your Arduino IDE, and you will see some text containing the COM Port number. We will use this to initiate our Python serial connection, like so:

arduino = serial.Serial('COM1', 115200, timeout=.1)

The above code will create a new serial object called "ardunio" on "COM1" with a "115200" [baud-rate](https://en.wikipedia.org/wiki/Baud) and a .1 second timeout. It is extremely important that you keep the chosen baud-rate on hand, as it must match exactly with the baud-rate on the Ardiuno side.

3. A simple code to test communication between Python and PRIZM

The listed code below can be used to test Python/PRIZM communication by controlling the LED light on PRIZM.

Code listing 3.1a: Controlling LED light through Communication (Python side)

import serial # add Serial library for Serial communication

import time

Arduino\_Serial = serial.Serial('com3',9600) #Create Serial port object called arduinoSerialData

#print Arduino\_Serial.readline() #read the serial data and print it as line

def lightOn():

Arduino\_Serial.write(str.encode('y'))

print("Turn light on")

def lightOff():

Arduino\_Serial.write(str.encode('n'))

print("Turn light off")

while True:

lightOn()

time.sleep(2)

lightOff()

time.sleep(2)

As you can see, the code creates a new serial object called Arduino\_Serial on "com3" with a "9600" [baud-rate](https://en.wikipedia.org/wiki/Baud).

You can change the baud rate in the Python and Arduino programs as long as they stay the same.

Here is the code that is used on the PRIZM side.

Code listing 3.1b: Controlling LED light through Communication (PRIZM Side)

#include <PRIZM.h> // include the PRIZM library

PRIZM prizm;

const int led=13;

int value=0;

void setup()

{

prizm.PrizmBegin();

Serial.begin(9600);

pinMode(led, OUTPUT);

digitalWrite (led, LOW);

Serial.println("Connection established...");

}

void loop()

{

value = 10;

while (Serial.available())

{

value = Serial.read();

}

if (value == 'y'){

prizm.setRedLED(HIGH);

}

else if (value == 'n'){

prizm.setRedLED(LOW);

}

}

3.3 DC Motor Control with Python

For vision robot like EduBot, what we need is to use Python code to control DC motors through the serial communication. That is the code we will study here. The refined version of this code is similar to LOCOMOCO, a C# motor drive class used in the past VCC activities.

Code listing 3.2a: Driving DC Motors Using Python Code (Python side)

import serial # add Serial library for Serial communication

import time

Arduino\_Serial = serial.Serial('com3',9600) #Create Serial port object called arduinoSerialData

#print Arduino\_Serial.readline() #read the serial data and print it as line

def motorForward():

Arduino\_Serial.write(str.encode('y'))

print("Turn motor forwards")

def motorBackward():

Arduino\_Serial.write(str.encode('n'))

print("Turn motor backwards")

while True:

motorForward()

time.sleep(2)

motorBackward()

time.sleep(2)

Code listing 3.2b: Driving DC Motors Using Python Code (PRIZM side)

#include <PRIZM.h> // include the PRIZM library

PRIZM prizm;

const int led=13;

int value=0;

void setup()

{

prizm.PrizmBegin();

Serial.begin(9600);

pinMode(led, OUTPUT);

digitalWrite (led, LOW);

Serial.println("Connection established...");

prizm.setMotorInvert(2,1); // invert the direction of DC Motor 1 to harmonize the direction

}

void loop()

{

value = 10;

while (Serial.available())

{

value = Serial.read();

}

if (value == 'y'){

prizm.setRedLED(HIGH);

prizm.setMotorPower(1,25); // spin Motor 1 CW at 25% power

prizm.setMotorPower(2,30);

}

else if (value == 'n'){

prizm.setRedLED(LOW);

prizm.setMotorPower(1,-25); // spin Motor 1 CW at 25% power

prizm.setMotorPower(2,-28);

}

}

4. EduBot Programming with Vision

In this section, we will add a webcom and Python/OpenCV to EduBot to make it a vision robot, which is the main purpose of this e-book.

**See Appendix B for Python , OpenCV Installation steps**

4.1 Python OpenCV Introduction

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library is used extensively in companies, research groups and by governmental bodies.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS.

EmguCV and C# WFA were used for Robofest VCC challenges in previous years. Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++,etc..

OpenCV-Python is a library of Python bindings designed to solve computer vision problems, and it is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language. Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules this gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax.

4.2 OpenCV GUI Featueres

a) Getting Started with Images

Code listing 4.1: Getting Started with Images

The following program loads an image in grayscale, displays it, save the image if you press

‘s’ and exit, or simply exit without saving if you press ESC key.

import numpy as np

import cv2

img = cv2.imread('messi5.jpg',0)

cv2.imshow('image',img)

k = cv2.waitKey(0)

if k == 27: # wait for ESC key to exit

cv2.destroyAllWindows()

elif k == ord('s'): # wait for 's' key to save and exit

cv2.imwrite('messigray.png',img)

cv2.destroyAllWindows()

A few notes:

(1) Use the function **cv2.imread()** to read an image, **cv2.imshow()** to display an image in a window, and **cv2.imwrite()** to save it back.

(2)**cv2.waitKey()** is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If **0** is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key a is pressed etc which we will discuss below.

(3)**cv2.destroyAllWindows()** simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argument.

b) Getting Started with Videos

Code listing 4.2: Getting Started with Videos

The following program will show how to capture a video from the camera convert it into

grayscale video and display it. Just a simple task to get started.

import numpy as np

import cv2

cap = cv2.VideoCapture(0)

while(True):

# Capture frame-by-frame

ret, frame = cap.read()

# Our operations on the frame come here

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Display the resulting frame

cv2.imshow('frame',gray)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# When everything done, release the capture

cap.release()

cv2.destroyAllWindows()

A few notes:

(1) Use the function **cv2.VideoCapture()** to to capture a video

(2) cap.read() returns a bool (True/False). If frame is read correctly, it will be True. So you can check end of the video by checking this return value.

c) Drawing Functions in OpenCV

Code listing 4.3: Drawing Functions in OpenCV

Here are some simple codes showing how to use Drawing Functions that can be used in

OpenCV to to draw different geometric shapes .

**Drawing Line**

To draw a line, you need to pass starting and ending coordinates of line. We will create a black image and draw a blue line on it from top-left to bottom-right corners.

import numpy as np

import cv2

# Create a black image

img = np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px

img = cv2.line(img,(0,0),(511,511),(255,0,0),5)

**Drawing Rectangle**

To draw a rectangle, you need top-left corner and bottom-right corner of rectangle. This time we will draw a green rectangle at the top-right corner of image.

img = cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)

**Drawing Circle**

To draw a circle, you need its center coordinates and radius. We will draw a circle inside the rectangle drawn above.

img = cv2.circle(img,(447,63), 63, (0,0,255), -1)

Note:

In all the drawing functions like **cv2.line()**,  **cv2.circle()** ,  **cv2.rectangle()**,  **cv2.ellipse()**,  **cv2.putText()** etc, you will see some common arguments as given below:

* img : The image where you want to draw the shapes
* color : Color of the shape. for BGR, pass it as a tuple, eg: (255,0,0) for blue. For grayscale, just pass the scalar value.
* thickness : Thickness of the line or circle etc. If **-1** is passed for closed figures like circles, it will fill the shape. default thickness = 1
* lineType : Type of line, whether 8-connected, anti-aliased line etc. By default, it is 8-connected. cv2.LINE\_AA gives anti-aliased line which looks great for curves.

d) Track bar for Threshold

Code listing 4.4: Track Bar for Threshhold

import numpy as np

import cv2

img=cv2.imread("p2.jpg",0)

def nothing(x):

pass

cv2.namedWindow('image')

cv2.createTrackbar('min','image',0,255,nothing)

cv2.createTrackbar('max','image',0,255,nothing)

while(1):

a = cv2.getTrackbarPos('min','image')

b = cv2.getTrackbarPos('max','image')

#ret,thresh=cv2.threshold(img,a,b,cv2.THRESH\_BINARY\_INV)

ret,thresh=cv2.threshold(img,a,b,cv2.THRESH\_BINARY)

cv2.imshow("output",thresh)

k = cv2.waitKey(10) & 0xFF

if k == 27:

break

print (cv2.countNonZero(thresh))

cv2.destroyAllWindows()

4.3 Image Processing in OpenCV

a) Image Thresholding

## In simple Thresholding, If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is **cv2.threshold**. First argument is the source image, which **should be a grayscale image**. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

* cv2.THRESH\_BINARY
* cv2.THRESH\_BINARY\_INV
* cv2.THRESH\_TRUNC
* cv2.THRESH\_TOZERO
* cv2.THRESH\_TOZERO\_INV

Two outputs are obtained. First one is a **retval** which will be explained later. Second output is our **thresholded image**.

Code listing 4.5: Simple Threshholding in OpenCV

This code shows how to perfom simple thresholding by using function **cv2.threshold**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('gradient.png',0)

ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV)

ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH\_TRUNC)

ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO)

ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO\_INV)

cv2.imshow('frame1', thresh1)

cv2.imshow('frame2', thresh2)

cv2.imshow('frame3', thresh3)

cv2.imshow('frame4', thresh4)

cv2.imshow('frame5', thresh5)

k = cv2.waitKey(5) & 0xFF

if k == 27:

break

b)White Pixel Counting and Obtain Image Information

Code listing 4.6: White Pixel Counting and Obtain Image Information

This example code shows (a) how to obtain image information (b) how to obtain pixel

number and (c) how to define ROI in an image etc.

import numpy as np

import cv2 as cv

# import and show images

img = cv.imread('p2.jpg',0) # read as gray image

img2 = cv.imread('p2.jpg',1) # read as colored image

cv.imshow('image\_BW',img) # display BW image

cv.imshow('image\_colored',img2) # display colored image

# obtain image information

#create binalized BW image

ret,thresh = cv.threshold(img,150,230, cv.THRESH\_BINARY)

#BW image height, width and pixel number

print ("img.shape")

print ("thresh.shape")

height, width = img.shape

print ("height and width : ", height, width)

print ("img.size")

print ("thresh.size")

size = img.size

print ("size of the image in number of pixels", size)

# To obtain white pixel number by using countNonZero(thresh)

print ("white pixel number", cv.countNonZero(thresh))

# black pixel number = total -white

print ("black pixel number", size-cv.countNonZero(thresh))

# How to define ROI in an image

"""

Note:

If (x1,y1) and (x2,y2) are the two opposite vertices of plate you obtained, then

simply use function:

roi = image[y1:y2, x1:x2]

that is the image ROI.

"""

ROI1 = img2[200:260, 530:590]

ROI2 = img2[400:460, 200:260]

cv.imshow('image\_ROI1',ROI1)

cv.imshow('image\_ROI2',ROI2)

print (" ROI1 shape and size", ROI1.shape, ROI1.size)

print (" ROI2 shape and size", ROI2.shape, ROI2.size)

# Modify original img2 using ROIs

img2[400:460, 265:325] = ROI1

cv.imshow('new\_img\_with\_ROI1',img2)

img2[300:360, 220:280] = ROI2

cv.imshow('new\_img2\_with\_ROI2',img2)

k = cv.waitKey(0) & 0xFF

if k == 27:

#break

cv.destroyAllWindows()

c) Colored Pixel Counting

For black images you get the total number of pixels (rows\*cols) and then

subtract it from the result you get from countNonZero(mat).

For other values, you can create a mask using inRange() to return a binary

mask showing all the locations of the color/label/value you want and then

use countNonZero to count how many of them there are.

Code listing 4.7: Colored Pixel Counting

This example code shows how to obtain image information (b) how to obtain pixel

number and (c) how to define ROI in an image etc.

import numpy as np

import cv2 as cv

#img2 = cv.imread('RGB\_1.jpg',1)

#img2 = cv.imread('RGB\_1.jpg',1)

#img2 = cv.imread('RGB\_3.png',1)

#img2 = cv.imread('HSV\_2.jpg',1)

#img2 = cv.imread('red.png',1) # read as colored image

img2 = cv.imread('green.png',1) # read as colored image

#img2 = cv.imread('blue.png',1) # read as colored image

cv.imshow('original BGR',img2)

# Convert BGR to HSV

hsv\_image = cv.cvtColor(img2, cv.COLOR\_BGR2HSV)

cv.imshow('converted HSV',hsv\_image)

a1, a2=cv.meanStdDev(img2)

b1, b2=cv.meanStdDev(hsv\_image)

print ('a1: mean for imag2-BGR', a1)

print ('a2: deviation',a2)

print ('b1: mean for img-HSV', b1)

print ('b2: deviation',b2)

# define range of blue color in HSV

#lower\_blue = np.array([110,50,50])

"""

#For red:

lower\_blue = np.array([-5,250,253])

upper\_blue = np.array([10,290,260])

"""

#For green:

lower\_blue = np.array([50,200,250])

upper\_blue = np.array([65,290,260])

"""

#For blue:

lower\_blue = np.array([90,150,240])

upper\_blue = np.array([124,250,250])

"""

# Threshold the HSV image to get only blue colors

mask = cv.inRange(hsv\_image, lower\_blue, upper\_blue)

cv.imshow('mask',mask)

no\_blue = cv.countNonZero(mask)

print('The number of blue pixels is: ' + str(no\_blue))

# Bitwise-AND mask the original image

res = cv.bitwise\_and(img2,img2, mask= mask)

cv.imshow('res',res)

k = cv.waitKey(0) & 0xFF

if k == 27:

#break

cv.destroyAllWindows()

d) Changing Colorspaces

There are more than 150 color-space conversion methods available in OpenCV. But we will look into only two which are most widely used ones, BGR \leftrightarrow Gray and BGR \leftrightarrow HSV.

For color conversion, we use the function cv2.cvtColor(input\_image, flag) where flag determines the type of conversion.

For BGR \rightarrow Gray conversion we use the flags cv2.COLOR\_BGR2GRAY. Similarly for BGR \rightarrow HSV, we use the flag cv2.COLOR\_BGR2HSV.

## Note:

## For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. Different softwares use different scales. So if you are comparing OpenCV values with them, you need to normalize these ranges

Code listing 4.8: Changing colorspace in OpenCV

In this example, we will show how to convert BGR image to HSV to extract a colored object. In HSV, it is more easier to represent a color than RGB color-space. In our application, we will try to extract a blue colored object. So here is the method:

* Take each frame of the video
* Convert from BGR to HSV color-space
* We threshold the HSV image for a range of blue color
* Now extract the blue object alone, we can do whatever on that image we want.

import cv2

import numpy as np

cap = cv2.VideoCapture(0)

while(1):

# Take each frame

\_, frame = cap.read()

# Convert BGR to HSV

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

# define range of blue color in HSV

lower\_blue = np.array([110,50,50])

upper\_blue = np.array([130,255,255])

# Threshold the HSV image to get only blue colors

mask = cv2.inRange(hsv, lower\_blue, upper\_blue)

# Bitwise-AND mask and original image

res = cv2.bitwise\_and(frame,frame, mask= mask)

cv2.imshow('frame',frame)

cv2.imshow('mask',mask)

cv2.imshow('res',res)

k = cv2.waitKey(5) & 0xFF

if k == 27:

break

cv2.destroyAllWindows()

4.4 Follow a White Line

Code listing 4.9: Following an Edge of a Black Line

import numpy as np # Numerical calculation class

import serial # the port class

import cv2 as cv

#establish communication between python and PRIZM

Arduino\_Serial = serial.Serial('com10',9600)

VIDEO\_SRC = 0

running=False

cap = cv.VideoCapture(VIDEO\_SRC)

def nothing(x):

pass

cv.namedWindow('image')

cv.createTrackbar('min','image',0,255,nothing)

cv.createTrackbar('max','image',0,255,nothing)

while(cap.isOpened()):

# Capture frame-by-frame

ret1, img\_BGR = cap.read()

cv.imshow('img\_BGR',img\_BGR)

if not ret1:

print ('Video is off')

break

a = cv.getTrackbarPos('min','image')

b = cv.getTrackbarPos('max','image')

img\_gray = cv.cvtColor(img\_BGR, cv.COLOR\_BGR2GRAY)

ret,img\_BW=cv.threshold(img\_gray,a,b,cv.THRESH\_BINARY)

cv.imshow("image",img\_BW)

# RECTS for actual sensors

img\_zone\_left=cv.rectangle(img\_BW, (0, 0), (213, 240), (125,125,0), 2)

img\_zone\_middle=cv.rectangle(img\_BW, (213, 0), (426, 240), (125,125,0), 2)

img\_zone\_right=cv.rectangle(img\_BW, (426, 0), (640, 240), (125,125,0), 2)

cv.imshow('image',img\_zone\_left)

cv.imshow('image',img\_zone\_middle)

cv.imshow('image',img\_zone\_right)

if (running==False):

ch = cv.waitKey(10) & 0xFF

if ch == ord('i'): # start FiCO mission

running=True

else:

#define ROIs images for 4 zones in RECTs

ROI\_zone\_left=img\_BW[0:240, 200:280]

ROI\_zone\_middle=img\_BW[0:240, 280:360]

ROI\_zone\_right=img\_BW[0:240, 360:440]

zone\_Left\_white=cv.countNonZero(ROI\_zone\_left)

zone\_Middle\_white=cv.countNonZero(ROI\_zone\_middle)

zone\_Right\_white=cv.countNonZero(ROI\_zone\_right)

total5=80\*240

zone\_Left\_whiteP=float(cv.countNonZero(ROI\_zone\_left))/float(total5)

zone\_Middle\_whiteP=float(cv.countNonZero(ROI\_zone\_middle))/float(total5)

zone\_Right\_whiteP=float(cv.countNonZero(ROI\_zone\_right))/float(total5)

if ((zone\_Middle\_whiteP<0.85) and (zone\_Middle\_whiteP>0.05)):

Arduino\_Serial.write(str.encode('f'))

elif ((zone\_Left\_whiteP<0.95)):

Arduino\_Serial.write(str.encode('l'))

elif ((zone\_Right\_whiteP>0.05) and (zone\_Right\_whiteP<0.75)):

Arduino\_Serial.write(str.encode('r'))

if cv.waitKey(1) & 0xFF == ord('q'):

Arduino\_Serial.write(str.encode('s'))

#mc.stop()

break

cv.waitKey(0)&0xFF

cap.release()

cv.destroyAllWindows()

4.5 Follow An Edge (Change to Follow a white line and stop at a colored paper)

Code listing 4.10: Following an White Line

References:

1. PITSCO PRIZM, https://asset.pitsco.com/sharedimages/resources/tetrix-prizm-

programming- guide.pdf

1. PITSCO/PRIZM, <https://asset.pitsco.com/sharedimages/resources/43167_prizm_>

quickstartguide.pdf

1. Python, https://www.python.org/
2. https://spectrum.ieee.org/static/interactive-the-top- programming-languages-2017

4. Open-CV, <https://en.wikipedia.org/wiki/OpenCV>

5. About Open-CV, <https://opencv.org/about.html>

6. OpenCV-Python Tutorials,<https://opencv-python>tutroals.readthedocs.io/en/latest/py\_tutorials/py\_tutorials.html

7. Robofest VCC, <https://www.robofest.net/index.php/current-competitions/vision-centric-challenge>

8. “Introduction to L2Bot” CJ Chung, Professor of Computer Science Founder & Director of Robofest

Acknowledgments:

We would like to thank PITSCO for sponsoring the EduBot development.

We would like to thank DENSIN for sponsoring Robofest VCC challenge event.

We would like to thank LTU/Robofest for hosting Robofest events.

We would like to thank Professor and Robofest Founder CJ Chung for supporting the work

on EduBot development.

Appendix A - How to Install Arduino IDE and PRIZM Arduino Libarary

**Installing the Arduino Software (IDE)**: In order to upload programs to the TETRIX PRIZM Robotics Controller, you need to install the Arduino Software (IDE). Full and comprehensive instructions for installing the software on Windows, Mac, and Linux computers can be found at the Arduino website at www.arduino.cc.

**To install**:

1. Go to [www.aruino.cc](http://www.aruino.cc)
2. From the homepage, click Download to access the software downloads section.
3. Download Arduino IDE
4. Open Aduino software application
5. After the Arduino Software is opened, select Arduino/Genuino Uno from the Tools > Board menu to enable communication with PRIZM.
6. Select the COM port where PRIZM has been installed by your computer system.

A more detailed explanation of software installation is included in the full TETRIX PRIZM Programming Guide available for download at <https://www.pitsco.com/TETRIX-PRIZM-Robotics-Controller>. Here, you will also find the latest updates on supported software platforms for the TETRIX PRIZM Robotics Controller.

**Installing the PRIZM Robotics Controller Arduino Library:** After you have successfully installed the Arduino Software (IDE), you can add the PRIZM Robotics Controller Library for Arduino to your programming pallet. The library is a specialized set of functions written specifically for the controller that greatly simplify the creation of programming applications to control your robot’s motors and sensors. The PRIZM library is distributed as a zip file: TETRIX\_PRIZM.zip.

**To install**:

(1) Download the PRIZM library from <https://www.pitsco.com/sharedimages/resources/TETRIX_PRIZM_01_03_18.zip>. After it is downloaded, save it to a location on your computer so you can install it into the Arduino Software (IDE)

(2) To install the PRIZM library into the Arduino Software (IDE), one way is to import it using the Arduino Software (IDE) Add .ZIP Library menu option: in the Arduino Software (IDE), navigate to Sketch > Include Library. From the dropdown menu, select Add .ZIP Library (Figure A1.1).

(3) You will be prompted to select the library you would like to add. Navigate to the location where you saved the TETRIX\_PRIZM.zip library file, select it, and open it (Figure A1.2).

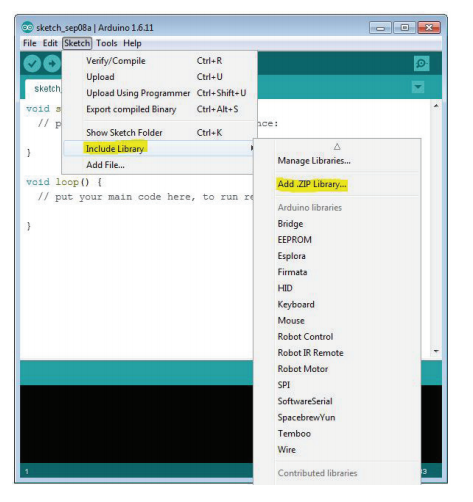


Fig.A1.1

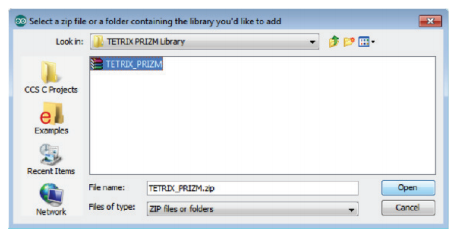


Fig.A1.2

**Configuring USB Communication**

PRIZM and the Arduino Software (IDE) will communicate with each other through the

computer’s USB port. Therefore, before we can begin programming, we first need to be

sure that the PRIZM controller is properly set up in the Arduino Software (IDE) for

communication over the USB port. The easiest way to do this is to first start the Arduino

Software (IDE) and navigate to Tools > Board and select Arduino/Genuino Uno (Figure A1.3).

The PRIZM controller uses the same processor chip as a genuine Arduino UNO, so this is the

board you will select.

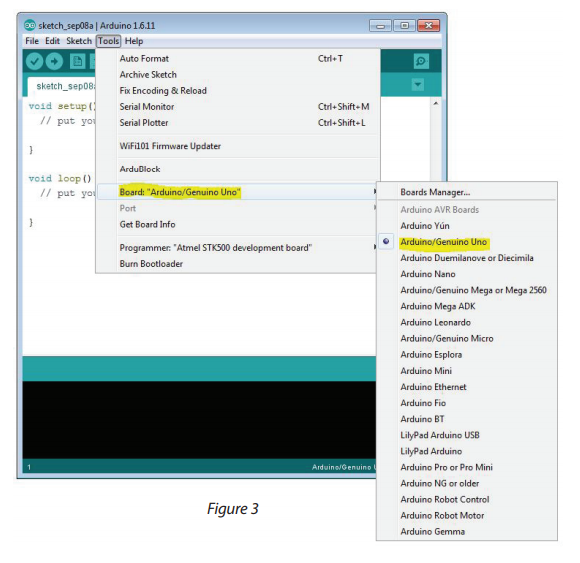
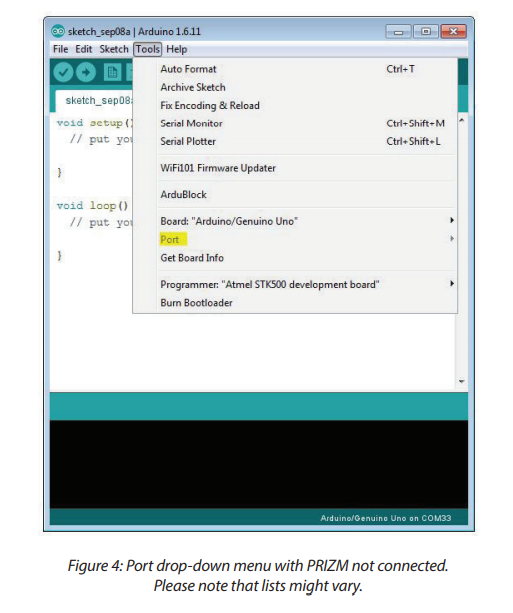


Fig.A1.3

Next, without the PRIZM connected, navigate to Tools > Port (Figure 4) and check the

current connections. If there are no current connections detected, the word Port will be

grayed out. If there are connections detected, take note of the COM ports that are listed.



Next, plug the PRIZM controller into a USB port and power it up by connecting the TETRIX

battery pack and turning the power switch on.

Finally about Arduino Code Examples: The PRIZM library includes several code examples

that will assist you in understanding how to program PRIZM using the library functions.

After the library has been installed, be sure to restart the Arduino Software. After the

software has restarted, the code examples will be located in the File > Examples drop-down

menu.

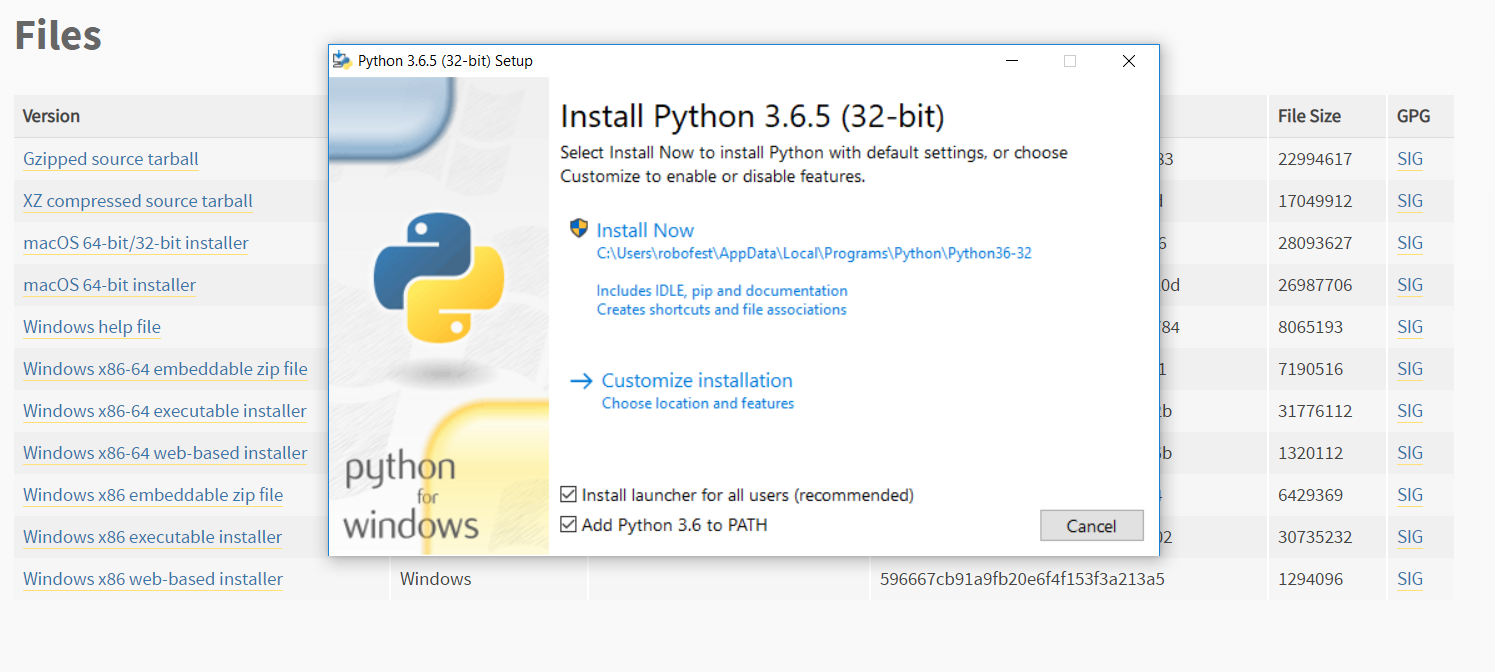
Appendix B - How to Install Python­ V3, OpenCV, Numpy and Matplotlib

Installation Steps:

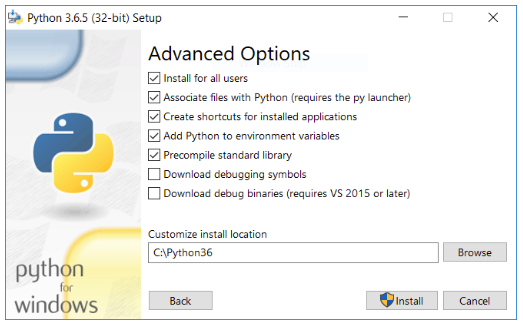
1. Download Python version 3.6.5 by going to the following link:

<https://www.python.org/downloads/release/python-365/>

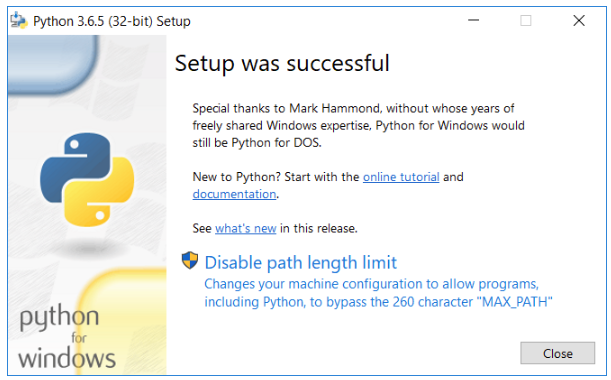
Select Windows x86 executable installer. Make sure that the Add Python to PATH checkbox is selected



Click on custom installation. Click on Advanced Options and check all necessary boxes and then customize installation location to C:\Python36



2. You should see this prompt after installation is complete. Make sure to click on Disable path length limit.



3. Go to all programs (located in bottom right corner for windows 10) and click IDLE(Python 3.6 32-bit) to launch the Python shell

4. Type print ("Hello World") to make sure that Python works

5. Install Python modules as needed:

Examples of OpenCV, Numpy, Serial (must have for vision usage) as well as matplotlib (optional) are shown below.

This can be done through the following method:

1. open command prompt

2. cd into C:\Python36\Scripts>

3. Type the commands:

For OpenCV:

pip install opencv-python==3.4.0.14 (or your choice of version)

For Numpy:

pip install numpy

For Serial:

pip install serial

For Matplotlib:

pip install matplotlib

6. Go to Python shell and type the following commands to make sure the modules were installed successfully

For OpenCV:

import cv2

For Numpy:

import numpy

For Serial:

import serial

For Matplotlib:

import matplotlib

If there are no error messages, then the installations were successful.

Appendix C - EduBot Building Instruction (See separate document)