Robotõ

Integrating a Raspberry Pi 2 onto a Thymio 2 robot with Python 3 and D-bus
(He's Spanish)

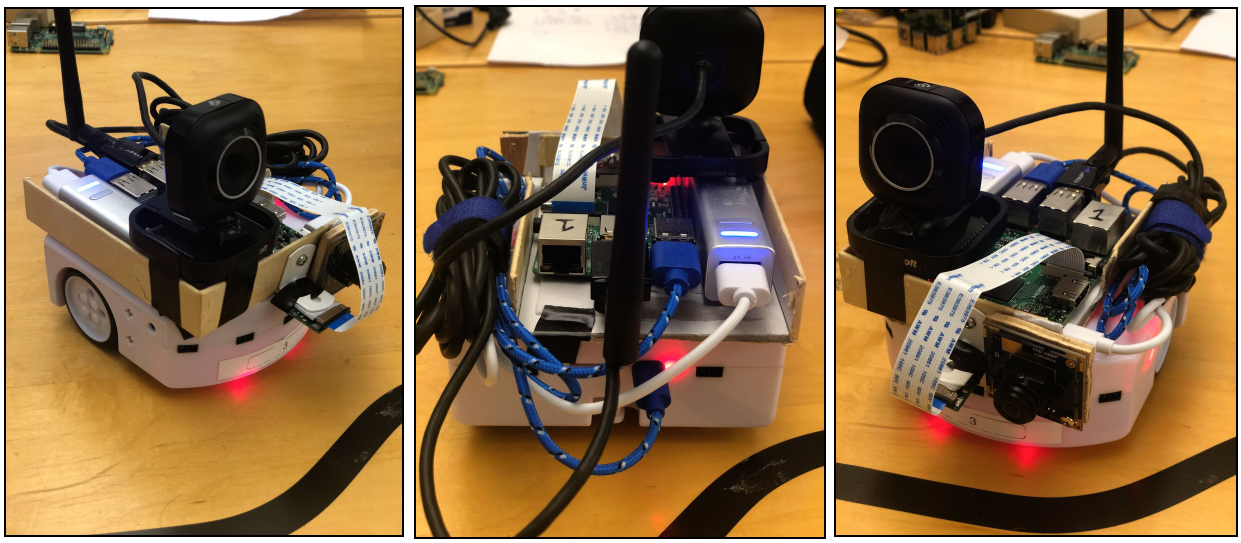

Figure 1: Front, Back, and Side view of Robotõ

1 Introduction:

The Thymio 2 is a small, 2 wheel drive robot that comes with its own unique event-based programming language *Aseba*. Also provided is the utility *Asebamedulla*, which allows for the communication between external devices and the Thymio using D-bus, a communication software for distinct programs. In the following project, these available tools are used to integrate a Raspberry Pi 2 with the Thymio, opening the door for a higher level of robotics such as: running other programming languages like Python and C++, adding additional sensors, and communicating with other devices via an online network.

Demonstrated in this project are some possible applications made available by integrating a Raspberry Pi 2 with the Thymio. This includes Line Following via color filtering + a PD controller, line following with a convolutional neural network (CNN) as the controller, color following using the top camera, as well as a Caffe tracking model using an external computer to do the processing by sending raw video data via ZMQ.

2 Installation:

1. Download and install Raspbian Jessie onto the Raspberry Pi 2:

At the time of writing, the latest version Aseba is only compatible with Raspbian Jessie. To install this version of Raspbian on the Raspberry Pi 2, the Raspbian Jessie image needs to be flashed onto the SD card that will be used. Click here for more details on how to flash an SD card, and here to download the image.

2. Install Aseba and other necessary packages:

Before anything else, update and upgrade the available packages on the Raspberry Pi 2:

```
$sudo apt-get update
$sudo apt-get upgrade
```

Update the Raspberry Pi's firmware, this includes a necessary update for a v4l2 driver used to operate both cameras

```
$sudo rpi-update
```

Download Aseba, following the prompted installation instructions. Note that the latest install for Raspberry Pi 3 architecture also works on the Raspberry Pi 2 model.

Install Packages:

```
$sudo apt-get install python3-dbus
$sudo apt-get install PiCamera
$sudo apt-get install python3-numpy
$sudo apt-get install python3-matplotlib
$sudo apt-get install xrdp
$sudo apt-get install tightvncserver
```

3. Enable D-Bus connection to the Thymio:

Before executing any programs, the connection between the Thymio and the Raspberry Pi 2 needs to be established. This requires a terminal to be constantly open, running a program initiated by the following command:

```
$asebamedulla "ser:debvice=/dev/ttyACM0
```

Note that the device name above is entered as 'ttyACM0'. This may not be the case in some situations, but can be determined by opening Aseba Studio and inspecting the list of available devices. For more details on using D-bus with asebamedulla, click here.

4. (Optional) Install OpenCV

For some applications including some of those described in this project, a computer vision library such as OpenCV may be required. To install OpenCV on Raspbian Jessie, it requires the user to build the package from source. A detailed step-by-step install for Raspbian Jessie can be found here.

3 Example Applications:

All executable applications are available for download here. In order to initialize the connection between the Thymio and the Raspberry Pi, make the file *Initialize.txt* executable upon downloading for the first time:

$chmod +x Initialize.txt

Thereafter, *Initialize.txt* can then be used to establish the connection, by simply executing the following command in the Roboto directory:

$./Initialize.txt

3.1 Line Follow: *Roboto/Line_follow_v8.py*

For a video demonstration, click here.

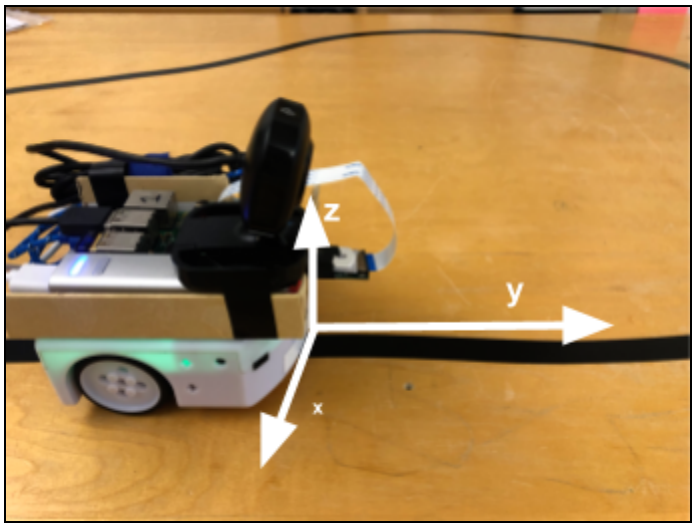

Figure 2: Line following Implementation

The objective of this application is to follow a black line as fast as possible while taking a path that minimizes the time spent where the line is not centered. This is achieved by using OpenCV to read the image, convert to an HSV format, and apply a binary mask. This means that each individual pixel value will then become true for black pixels(1), and false for beige(0). The mask applied, in this case, is one specifically calibrated to differentiate between black and beige colors.

Figure 3:RGB Image Taken During Line Following :320x160

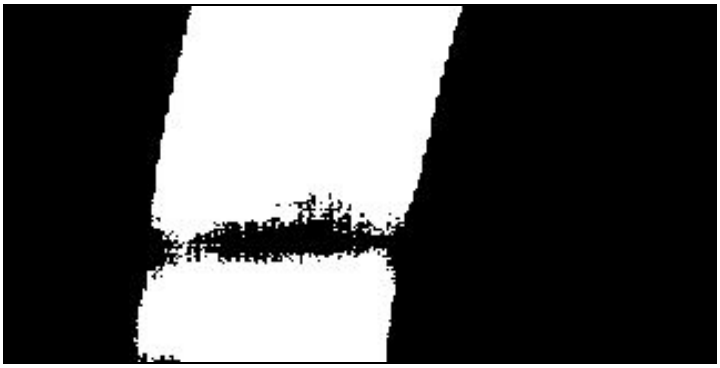

Figure 4: RGB Image After The Binary Mask Is Applied: 320x160

After applying the given binary mask to the camera output, the 2D array is then summed along the y-axis, producing a 1D array representing a distribution of true pixels (pixels that returned true after the binary mask). This process allows for the calculation of the mean pixel index (position with reference to the x-axis) since the total number of pixels can be calculated by summing the entire array. The corresponding distribution of true pixels to the real image and mask shown previously is provided in Figure 5 below, where the mean index can be calculated to be 128.89, out of 320 possible values.

After watching the video demonstration, the viewer may note that there is a brief moment where Robotõ loses the line, despite the raw RGB frames showing a black line in the window. This is due to the mask's inability to return what we can see are the black pixels, as true. This is presumed to be the result of different lighting conditions. Finding a generalized solution to differ between the black and beige colors is difficult, and using the mask as described previously is even more so. This issue, along with errors, is one that is aimed to be solved by finding a close-to optimal algorithm to follow a line using a convolutional neural network.

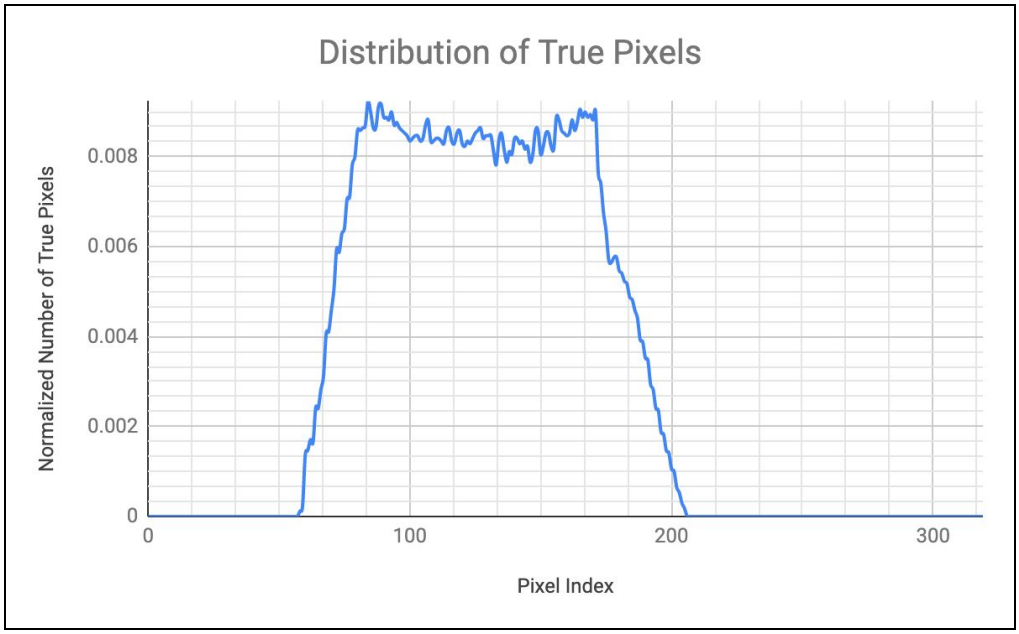

Figure 5: Distribution of True Pixels: 320x160

The index of the mean position is used to indicate the physical location of where to follow along the x-axis. A PD controller then uses this distance over some time to specify the left and right wheel velocities. This is done by adding a user enter bias for the speed, and combining this on top of the value calculated by the PD controller. For the left wheel velocity, the value returned by the PD controller is combined negatively, whereas for the right wheel velocity it is combined positively. This method of choosing the left and right wheel velocity is simple and produces a time-invariant linear velocity (see Equation 1 below)

$$Equation\ 1 : v_{linear} = (v_{left} + v_{right})/2 \ \ for\ a\ differential\ drive$$

$$Example\ Code :$$
$$Left\ speed\ =\ speed\ +\ (P * error\ +\ D * error)$$
$$Right\ speed\ =\ speed\ -((P * error\ +\ D * error))$$

The result is a line following algorithm that is both resistant to the noise produced by the color filter and using Numpy, computationally inexpensive compared to other explored methods such as applying convolution operations to filter noise.

To increase the speed at which the Thymio is capable of line following, many further improvements are possible. Some ideas for improvements include: slowing down near sharp turns, or on frames of high change in the mean index, utilizing the top camera to predict future actions, and adding a factor proportional to the sharpness of the distribution of true pixels (for example, standard deviation). This adds a stronger bias for an action when the mean is near centered, but the line is at an angle, see Figure 6 below.

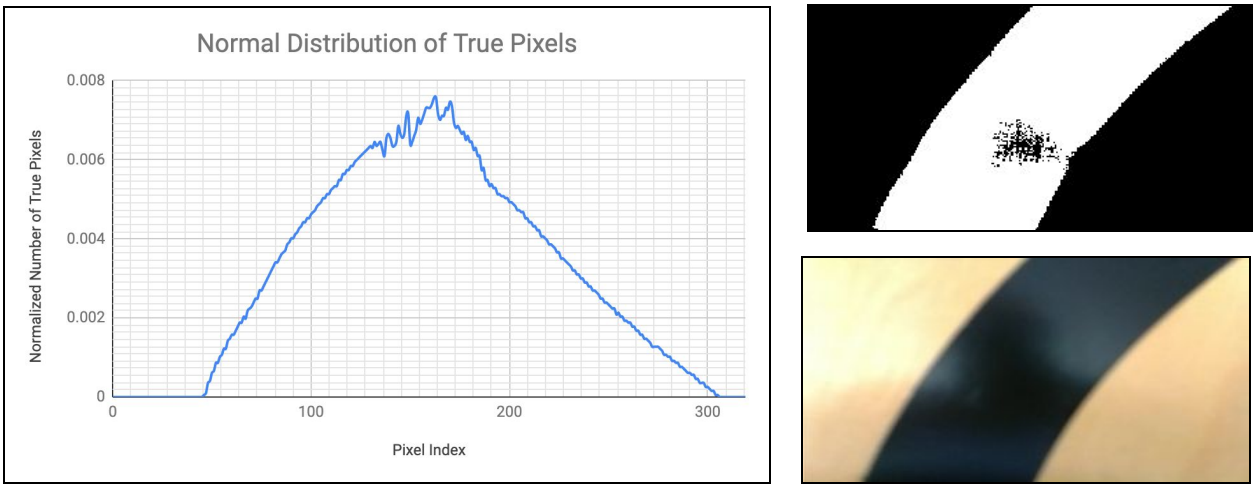

Figure 6: Near-Centred Mean With a Non-Zero Action Required

Another possibility to increase the speed is to combine data from both the top and downward-facing cameras. The hope being that the top camera can be used to add a bias towards actions that it will predict to be more likely, such as a sharp turn that the top camera can see, but the downward-facing camera can not.

3.2 Line Follow - CNN: *Roboto/Line_follow_CNN.py*

[Email jt.tam514@gmail.com to be added to the Google Colab]

The objective of this application is to use the predictions made by the line following algorithm in section 3.1, to train a Convolutional Neural Network(CNN) to follow the line, using raw RGB pictures from the downward facing camera.

1. Dense Layer - 320 neurons (hope to achieve a trivial solution, similar to classification)
2. Google Colab link
3. Must be Light Weight
4. VGG or another pre-trained network performance
5. Classification
6. Dropout effect on generalization accuracy of labels
7. Pooling
8. Evaluation

3.3 Color Follow: *Roboto/Color_follow_v10.py*

Implements the same strategy as described in Section 3.1: Line Follow, with the exception of using the top, front-facing camera. The user can then move around an object of the specified color, and Robotõ will follow accordingly.

3.4 Find Color: *Roboto/Webcam_color_test.py*

A simple program that allows the user to take a picture, click anywhere on the taken photo, and receive the RGB values of that exact point as output. This program can be used with the Line/Color Follow programs to click on a point of the image, and see its RGB or HSV value. This will hopefully make the process of creating a color filter much easier.

3.5 ZMQ Protocol: *Roboto/ZMQ_Protocal/*

Credit:
https://www.pyimagesearch.com/2019/04/15/live-video-streaming-over-network-with-opencv-and-imagezmq/

ZMQ is a message library, that with the addition of ImageZMQ will allow the Raspberry Pi 2 to publish raw image data over a local network. This application explores this method by capturing RGB data using the attached webcam and sending it to an external computer over a local network. The external computer then uses the data as input to a pre-trained Caffe MobileNet SSD object tracker, then displaying the output. The result is shown below in Figure 7.

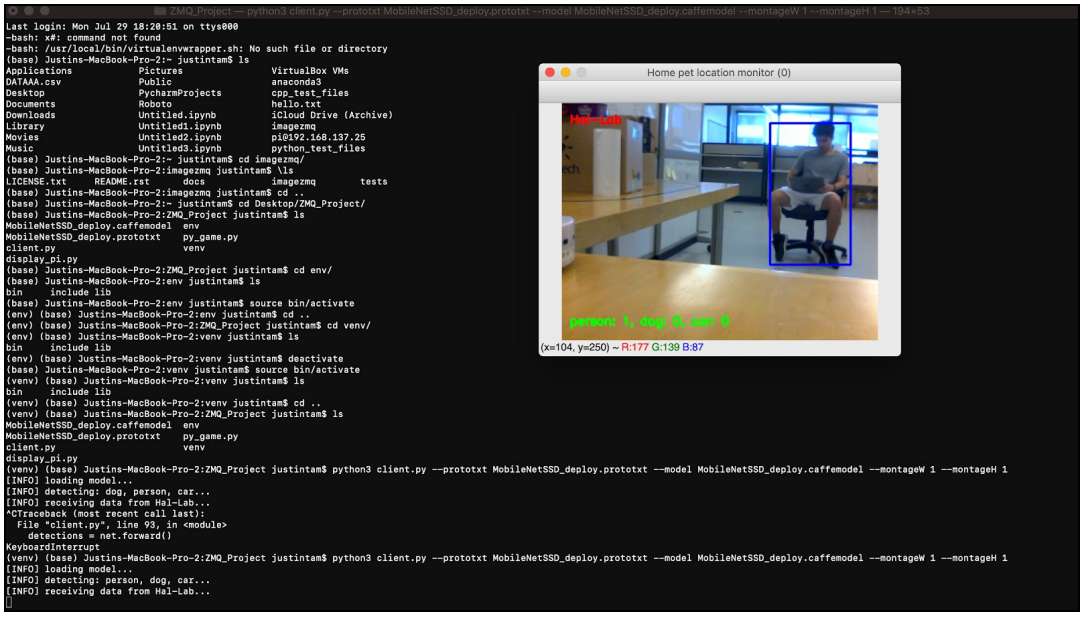

Figure 7: Screenshot of Caffe

On both the client and server the ZMQ and ImageZMQ packages need to be installed. Using a Macbook Pro as the client, and the Raspberry Pi 2 as the server:

Install ZMQ: https://github.com/MonsieurV/ZeroMQ-RPi

Install ImageZMQ

```
$cd ~
$git clone https://github.com/jeffbass/imagezmq.git
```

To publish the raw data from the server:

```
$python3 -s {IP Address of Client}
```

To run the model on the client:

```
$ python3 client.py --prototxt MobileNetSSD_deploy.prototxt --model
MobileNetSSD_deploy.caffemodel --montageW 1 --montageH 1
```

3.6 Braitenberg Vehicle: *Roboto/Braitenberg.py*

A **Python 2** implementation of a Braitenberg vehicle. For details on the Braitenberg implementation, follow this link: http://wiki.thymio.org/en:thymioraspyexample.

3.7 Ideas for Future Projects:

1. RL - Q-learning
2.

5 Troubleshooting:

- When connecting to the Thymio, error 'Cannot open serial port'. This means the Pi does not read Thymio, despite the Thymio indicating a red led for charging. This can be due to the micro USB cable used, and switching to a higher quality cable has solved this issue.
- When 2 or more cameras are connected, the index of each camera is assigned on startup and can differ from session to session. Because of this, errors may occur if the camera indexes are not first properly assigned.
- Depending on the camera used, there may be significant latency when using OpenCV and the Raspberry Pi 2. One option of reducing the effect of this issue is to create a threading function using python's Threading module. This function will continuously grab frames to clear the buffer, reducing the latency. The files demonstrated here implement this solution, however, the optimal rate at which the buffer is emptied will depend on the camera used as well as the performance of the program being executed.