University of Colorado - Boulder

MCEN 5115 Mechatronics

May 4th, 2020

Implementing Machine Learning Models with Luxonis Depthai Camera

Author:
Riley Kenyon[a]

[a]MEID: 272-513

Professor:
Derek Reamon

Mechanical Engineering
UNIVERSITY OF COLORADO BOULDER

I. Introduction

This report is intended to fulfill the graduate section requirements of the course Mechatronics. In the report, the USB3 Depthai camera from Luxonis will be used to implement machine learning models via the Intel Movidius Myriad X VPU on the camera. In particular, the the report will serve as a guide to

1) Setup the Luxonis on aarch64 and armhf architectures
2) Calibrate the stereo cameras for differential depth
3) Develop a custom model using tensor flow for target recognition
4) Offload TensorFlow and Caffe models to the Luxonis camera

In the process of implementing custom machine learning models, the OS will be geared towards implementing embedded computer vision and ROS. Scripts relevant to the project will be located in the repository:

`https://github.com/RileyKenyon/Mechatronics`

II. Experimental Setup

The hardware being used for the project is a Raspberry Pi 3 B+. The single board computer utilizes light-weight operating systems and is useful as an inexpensive Linux/GNU environment. To setup the Raspberry Pi, additional software and toolkits will be installed to facilitate machine learning model implementation. The steps needed to build the experimental setup are

1) Select and flash the chosen operating system (OS) to the Raspberry Pi
2) Download and install the Intel OpenVino toolkit
3) Download and install the robotics operating system, ROS (optional)

Note this is only for running existing models, the computational resources of the Raspberry Pi are not significant enough to train a model. Training options are primarily through a web service such as Google Collab, with limited success on a smaller embedded GPU platform such as the Nvidia Jetson Nano.

A. Operating System

There are two operating system distributions outlined in this document: Raspbian (buster), which is derived from Debian and is the official operating system from the Raspberry Pi founation, and Ubuntu Mate, which is a light weight distribution of Ubuntu 18.04. The links to download the OS images are found below:

Rasbian Buster	`https://www.raspberrypi.org/downloads/raspbian/`
Ubuntu Mate 18.04	`https://ubuntu-mate.org/download/`

Note that Raspbian is 32 bit hard float and Ubutnu Mate can be either 32 bit or 64 bit. This can be verified by using the command: **`uname -m`** from the terminal once the OS is installed. If the value of `armv7l` is returned, the OS is 32 bit; if `aarch64` is returned, the OS is 64 bit. For the purposes of the project, there are two compatible OS for the Raspberry Pi that may be use with OpenVino, the visual computing toolkit offered from *intel*. This includes the Raspberry Pi Buster installation and the 64 bit experimental Ubuntu Mate OS. A more complete description of OpenVino can be found from Intel's website:

> "The openVINO toolkit is a free toolkit facilitating the optimization of a Deep Learning model from a framework and deployment using an inference engine onto Intel hardware."

In essence, in order to communicate with the Myriad X Visual Processing Unit (VPU), the main chip on Luxonis cameras, it is necessary to install the toolkit. The links to the openVino installation procedures are below:

Rasbian & Ubuntu Mate 18.04	`https://docs.openvinotoolkit.org/latest/index.html`

Our team is interested in using Robotic Operating System (ROS) to control processes involved the operation of the project. From community members, it was recommended to utilize Ubuntu Mate in order to more easily implement ROS. It is feasible to install ROS with Raspbian, however not recommended due to conflicts when installing new packages. An alternative is to use the threading python module to communicate between processes rather than installing ROS if the system only requires a couple threads. Regardless, the links to the ROS Melodic installation tutorials can be found below:

Rasbian	`http://wiki.ros.org/melodic/Installation/Debian`
Ubuntu Mate 18.04	`http://wiki.ros.org/melodic/Installation/Ubuntu`

B. Flashing Operating System

More documentation can be found online if necessary, however the process of flashing an image to a Micro-SD card is straight forward.

1) Download the image from the respective location (either Raspbian or Ubuntu in this report)
2) Download a flashing software such as Balena Etcher: `https://www.balena.io/etcher/`
3) Download an SD card formatter: `https://www.sdcard.org/downloads/formatter/`
4) Format SD card, select the OS image and SD in flashing software
5) Remove once completed and insert into hardware

C. Preliminary Package Install

After the first boot and sufficiently updating and upgrading the system, the installation of future packages requires some base dependencies. The command line utilities `apt, apt-get` are common for installing packages through the default package manager with apt software repositories. The location of these on Ubuntu and Debian are likely found in the `/etc/apt/sources.list` file or in separate files under the `/etc/apt/sources.list.d/`. If it is necessary to add more packages to the list, the methods to incorporate the additional repositories into apt is to utilize the add-apt-repository command. The format is:

```
add-apt-repository [options] repository
```

As with most command line tools, the `--help` option lists the arguments of the tool. The following commands were utilized in the obtaining base packages required for openVino

```
sudo apt-get update && sudo apt-get upgrade
sudo apt install python3-pip
sudo apt install git
sudo apt install cmake libusb-1.0-0-dev
sudo apt install python3-numpy
sudo apt install python3-opencv
```

If problems arise, try installing the package `gfortran`. Note that the apt repositories are usually more stable but older versions of the packages. `pip` is an alternative that uses the python package index (pypi) and wheels to install packages. This method was tested on the Raspbian distribution, but using pip with Ubuntu had issues with finding the packages built for the specific architecture. At this point the necessary packages should be installed for use with OpenVino

D. OpenVino Install

Installing OpenVino on Ubuntu Mate and Rasbian follow the same process, but is more complicated on Ubuntu Mate. Using the utility: `wget` which downloads data from an internet address, the version of OpenVino can be obtained for arm architectures. Utilizing the command

```
wget https://download.01.org/opencv/2020/openvinotoolkit/2020.1/\
l_openvino_toolkit_runtime_raspbian_p_2020.1.023.tgz
```

in the desired directory of installation, such as `~/Downloads/`. This will download the .tgz file and place the file in the working directory. Next the contents will be extracted and placed in the directory `/opt/intel/openvino`. Note the backslash in the previous command is used for readability and only separates the command into two lines. In order to extract the tarball (.tgz), use the commands

```
sudo mkdir -p /opt/intel/openvino
sudo tar -xf l_openvino_toolkit_runtime_raspbian_p_<version>.tgz --strip 1 -C /opt/intel/openvino
```

to create a directory and the parent directories (-p) and extract the contents of the tarball. The rest of the install can be followed from the link from Intel. Once complete, when a new instance of the terminal is launched or `~/.bashrc` is sourced, the command line should be preceded by the message:

```
[setupvars.sh] OpenVINO environment initialized
```

An alternate method from the pre-built binaries is to build the toolkit from source, this can be useful if using a system architecture that is not currently supported through the default installation methods. The project can be built using the instructions provided in the `build-instruction.md` file. A link to the project and the Github repository can be found at

Project	`https://01.org/openvinotoolkit`
Github	`https://github.com/openvinotoolkit/openvino`

E. Depthai Install

Depthai is the python package that is used to communicate with the Luxonis camera. Although the package is currently not hosted by the python package index, the company is planning on offering an official install via pip in the future. In the interim, there is an excellent walk through for how to install their packages linked below:

`https://docs.luxonis.com/api/#install`

The process is straight forward if you are planning on using Raspbian, however there are some tricks involved in getting it working properly with Ubuntu Mate. It is also important to note that the install is an editable install, and at the time of this writing was not recognized as a python package to be used with ROS.

1. Ubuntu Mate Installation

In order to communicate withe the Luxonis camera, the python modules of `depthai` have to be built for the system and architecture. By default the python modules in the download from the Luxonis Github include the `.so` files for the Rasbian operating system with `armv7l`. However, in order to work with `aarch64` Ubuntu 18.04 they will need to built using the `depthai-api`. This can be downloaded from the Luxonis github. This should be placed in the folder `<DownloadLocation>/depthai-api/`. Refer to the `README.md` for more information on the details. There is a shell script in the `depthai-api` download that will install the dependencies and build the modules for the system architecture. These are executed by using the `./<script>.sh` operator to execute, however on Ubuntu Mate the memory restriction causes the operation to hang and for the system to freeze. If such as scenario arises, hold `alt-printscreen` and perform the command "R E I S U B". This will restart the system without having to perform a hard boot. A work around for the memory issue is to boot the Raspberry Pi into the command line interface (CLI), which will reduce memory consumption and allow for the python modules to be built for the system. To accomplish this, run the following sequence of commands

```
sudo systemctl start multi-user.target
.<DownloadLocation>/depthai-api/build_py_module
```

This starts a command line interface on next boot; in order to switch back to graphical use `sudo systemctl start graphical.target`. Once the script succeeds in building the python modules, the module

`depthai.cpython-36m-aarch64-linux-gnu.so`

will be in the depthai directory. When attempting to run the `test.py` script for the Luxonis, there was a conflict with the openCV version. To list where the python packages are install, use the command `python3 -m site`. The openCV package for this system was in `/usr/lib/python3/dist-packages/` and needed to be copied over to the installation directory of OpenVINO.

```
cp cv2.cpython-36m-aarch64-linux-gnu.so <OpenVINODirectory>/python/python3/cv2.ab3.so
```

This will overwrite the openCV package used by OpenVINO to the one installed by apt. Afterwards, the depthai package needed relinking with the command

```
pip3 install --user -e depthai-python-extras/
pip3 install --user -e .
```

After running `python3 test.py` as a functional test, the system is capable of communicating with the Luxonis camera.

F. ROS Install

At this point, it is advisable to save the OS image in case anything goes wrong. Run win32 disk imager if using windows, select the SD card, and select the filename to read, and select read. The process takes approximately 12 minutes - note this will be a large file (30Gb). To further target the Raspberry Pi as a capable machine in Mechatronics, ROS will now be installed for Ubuntu Mate. In the installation of ROS, the system needed an additional package and a source directory.

```
sudo apt-get install python-rosdep
mkdir <ROSDIR>/ src/
```

The tutorials from the ROS website are beneficial in setting up the catkin workspace and understanding the fundamentals of the robotic operating system.

`http://wiki.ros.org/ROS/Tutorials`

III. Calibrate Camera

To user stereo depth measurements, the camera needs to be calibrated to account for its unique optic parameters. Once determined, these parameters can be used to undo the distortions naturally present in the camera. More information can be found by reading the openCV page for stereo calibration

`https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`

To aid in the process, Luxonis provides a calibration utility to perform the operation. Note that currently the checkerboard used to calibrate the camera is based on an 7x11 grid. It is possible to change the default values using a stream editor such as `sed` with the following series of commands.

```
cd <INSTALLDIR>/depthai-python-extras/depthai_helpers/
cp calibration_utils.py calibration_utils_py.bak
sed -i '/self.objp\|Chessboard/{
  N
  s/9/<WIDTH>/g
  s/6/<HEIGHT>/g
}' calibration_utils.py
```

to verify the correct lines were modified, run the command

```
diff -u calibration_utils.py.bak calibration_utils.py
```

Otherwise if using the default chessboard provided in the install of the `depthai` python package, the script requires no adjustments. The process of calibration can be followed from the Luxonis documentation found here

`https://docs.luxonis.com/tutorials/stereo_calibration/`

The checkerboard can either be printed or displayed on a flat monitor for calibration. After taking the images, the result is a custom configuration file placed in the directory `depthai/resources/depthai.calib`. This will be used for any future depth measurements.

IV. Developing a Model

Before implementing a model on the Luxonis Camera, a custom model has to be built for the purpose of identifying the targets correctly. To develop a pipeline for training and development, this report outlines the process of setting up TensorFlow on an NVIDIA Jetson Nano and a reccommended method of using Google Colaboratory to develop the neural network that will be run on the Movidius X chip. Although the NVIDIA Jetson Nano is a powerful embedded system, the processing requirements to train a neural network proved to be at the limits of the machine. An alternative approach is to use a Google Colab notebook to develop the model. The service is free and allows access to GPUs to accelerate the training process. In particular, Luxonis has a notebook that was used for training which was helpful in getting started

`https://docs.luxonis.com/tutorials/object_det_mnssv2_training/`

A. Jetson Nano Setup (optional)

If setting up a Jetson Nano from scratch with all the tools needed for machine learning, this guide

`https://www.pyimagesearch.com/2020/03/25/how-to-configure-your-nvidia-jetson-nano-for-computer-vision-and-deep-learning/`

is helpful in getting everything operating. Another resource if not building from source is a link to getting started with the Jetson Nano, and the dependencies needed to install TensorFlow can be found from NVIDIA's website below:

`https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html`

In the pip install, the package scipy failed and instead was replaced with the apt package `python3-scipy`. The tutorials for installing additional dependencies can be found in the tensorflow object detection api tutorial:

`https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html#cudnn-install`
`https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html`

The installation to the protocol buffers library which is necessary for optimizing tensorflow on the Jetson Nano can be found here:

`https://github.com/protocolbuffers/protobuf/blob/master/src/README.md`

As noted in the tensorflow documentation, it is optional to install cocotools if using it with the coco platform.

B. Image Collection

A python script can be used to collect images from the Luxonis camera using a hotkey and saved to a local directory. A good number for training and testing is 200 images to start, with a split of 80% to 20% respectively. I adjusted the test script from Luxonis to capture images using spacebar, which can be found on Github

`https://github.com/RileyKenyon/Mechatronics`

It is best practice that the images should very closely reflect the environment which the model will be implemented in. For the purposes of this document, the images were taken on a desk because the model is a proof of concept, but if it is needed in a test course or in other lighting conditions the training images should reflect that environment.

C. Image Annotation

The most time consuming portion of creating a object detection model is the annotation process. In order to annotate the images, the software package `labelImg` is used to create bounding boxes around the object of interest and save the metadata to an .xml file. The link to the `labelImg` annotation software can be downloaded from the source repository

`https://github.com/tzutalin/labelImg`

Although for this, the training will not be performed on the Jetson Nano, the setup and annotation process is the same as the link below.

`https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/v1.9.1/install.html`

Using the separation script to make two data sets, one for training and one for testing was useful from the link, rather than manually performing the process. If following the tutorial from `pyimagesearch.com`, in order to get `labelImg` working it was necessary to create a virtual environment with site packages using the command:

```
mkvirtualenv labelImg --system-site-packages
```

where `labelImg` was the name of the virtual environment. After the images are annotated and separated, upload the files to Google Drive in order to mount to a Google Colab notebook. It is advisable to create a new Google Drive under another email address in case an erroneous call is made such as `rm -r` which will remove everything in a given directory.

D. Training the Model

The training can be accomplished on Google Colab thorough a notebook from the Luxonis website. Currently present at the link

`https://drive.google.com/open?id=1p1KEb37RS3h5HvjxSzcByeCmWdhdYBOD`

The pipeline for training requires several conversions to work. At this point the files of the object being detected are image files *.jpg and data files *.xml. To create tfrecord files, the inputs are converted from *.xml to *.csv. The outputs are converted to a tfrecord file and label map file (*.pbtxt) using the images and *.csv files. This process is repeated for the training and test data. Training is accomplished by using a pre-trained model, in this case the Mobilenet SSD v2 Model and incorporating the new data for the custom object. This process takes a while, but after the loss of the model has settled to below 2, model has been trained. The output is an inference graph which is exported as `frozen_inference_graph.pb`.

E. Converting and Implementing the Model

The output of the Tensorflow model is a inference graph *.pb file. Using this file, the information can be converted to a .bin (weights) and .xml (graph) using the model optimizer offered by the OpenVino toolkit. This optimizer can be found by running the command

```
find <OPENVINO_DIR> -iname mo.py
find <OPENVINO_DIR> -iname myriad_compile
```

The python file is the model optimizer which performs the conversion. The second function, `myriad_compile` is used to convert the *.xml file into a .blob that is offloaded to the Myriad X chip. In addition to this file, the Myriad X chip requires a *.json file to inform the chip what outputs are expected from the model. In the case of the object detection model, the outputs are the top left corner position of the bounding box, as well as the width and height of the box. If using a pre-trained model such as anything in the OpenVino model zoo, the model outputs can be found

`https://docs.openvinotoolkit.org/latest/usergroup3.html`

The conversion for the test new model is accomplished via the Google Colab script provided by Luxonis as well, the `ssd_v2_support.json` file is adjusted for the new model. The `myriad_compile` command is used to perform the last conversion to a *.blob that can be used as a configuration file for the Luxonis. The command can be found at the Luxonis Github

`https://github.com/luxonis/depthai#conversion-of-existing-trained-models-into-intel-movidius-binary-format`

At the time of writing, OpenVino 2020.1.023 was used for the model conversion and compilation. For the custom model, the compilation option `-ip U8` did not yield the correct inference. In place, the option `-ip FP16` was used. As a result the image data did not convert correctly, but the metadata and bounding box were placed correctly, seen in figure 2.

V. Results

The model was able to be trained to identify the target in tensorflow. a sample test image is shown in figure 1. The bounding box is drawn around the target with a probability of 79%. Upon converting the model to a blob, the preview image has become distorted but the metadata remains informative. Similar to a PIXY camera, the output of the Luxonis would return the position of the bounding box which could be used for targeting or locating an object.

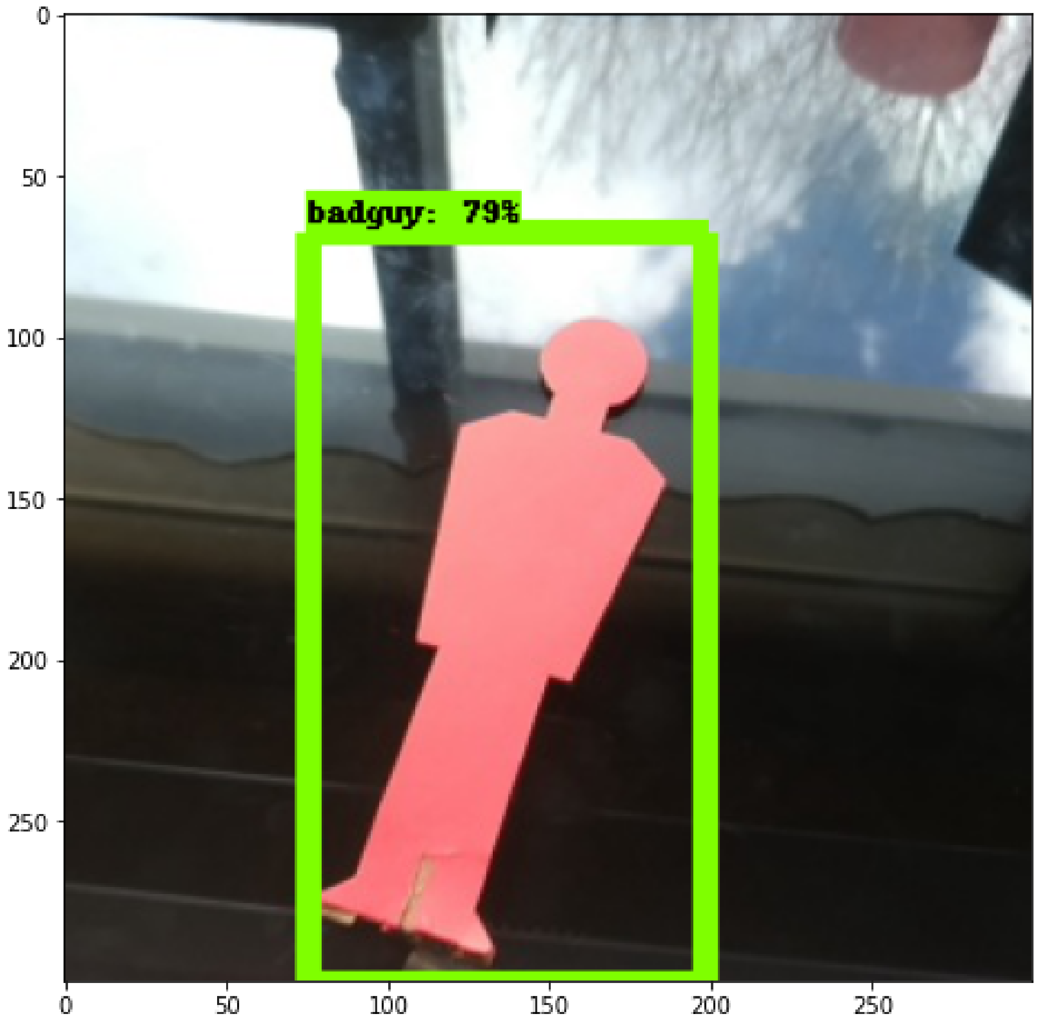

Fig. 1 The target was identified with a probability of 79%, and drew a bounding box accordingly.

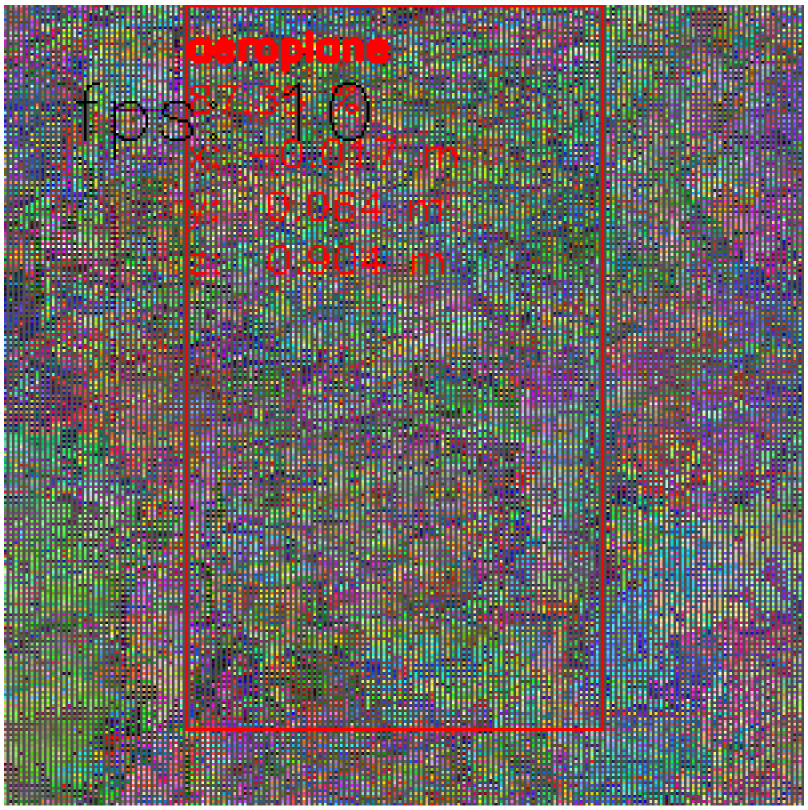

Fig. 2 The outcome of the model being converted to a blob for the Luxonis depthai camera. The background is indistinguishable, however the bounding box of the object was placed in the correct orientation and followed the target. The label is also referring to a different entry of the Mobilenet SSD v2 labels because the entry of the target was not introduced.

VI. Conclusion

Luxonis is working on a product that is rapidly being developed and is incredible for edge computing. Along with the rapid development, the code outlined here will likely need adjusting due to the pace at which the software is changing. In the future, the python module should be offered through a python package manager such as pip, and the main scripts should stabilize. However, at the time of this writing, the interfaces are changing daily. The uses for the camera are numerous and the functionality outlined here has barely scratched the surface. In the future, the stereo depth measurements could be utilized in ROS to perform simultaneous localization and mapping (SLAM) to navigate, as well as a more exhaustive object detection model which could be offloaded to the camera for inference. The hardware allows for complex operations without using the CPU of the embedded system. In the case of the Raspberry Pi, this frees up the resources to perform other tasks such as running ROS.

VII. References

1) `https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units`
2) `https://github.com/luxonis/depthai-python-extras`
3) `https://download.01.org/opencv/2020/openvinotoolkit/2020.1/`
4) `https://www.cyberciti.biz/faq/save-file-in-vi-vim-linux-apple-macos-unix-bsd/`
5) `https://www.pyimagesearch.com/2019/04/08/openvino-opencv-and-movidius-ncs-on-the-raspberry-pi/`
6) `https://www.raspberrypi.org/forums/viewtopic.php?t=176612`
7) `https://superuser.com/questions/1106174/boot-ubuntu-16-04-into-command-line-do-not-start-gui`
8) `https://wiki.ubuntu.com/Win32DiskImager`
9) `https://www.reddit.com/r/ROS/comments/e9e45p/ros_in_ubuntu_mate/`
10) `https://linuxize.com/post/how-to-add-apt-repository-in-ubuntu/`
11) `http://texdoc.net/texmf-dist/doc/latex/listings/listings.pdf`
12) `https://www.overleaf.com/articles/git-and-overleaf-integration/qmdncpnqwfxx`
13) `http://www.ee.surrey.ac.uk/Teaching/Unix/`
14) `https://www.linuxuprising.com/2018/07/mounting-google-drive-on-xfce-or-mate.html`
15) `https://github.com/NVIDIA/jetson-gpio`
16) `https://github.com/sparkfun/3D_Models`

VIII. Acknowledgments

I would like to acknowledge the help of the course instructors including the professor, Professor Derek Reamon. Furthermore, I would like thank the University of Colorado Boulder school of Engineering for providing phenomenal facilities to further my education.