ModBot Software Documentation

4CAD – April 30, 2018

Password to the Raspberry Pi: 4cadinc

Main Installations

- In Pi BIOS, enable i2c and camera
- Ruby Version Manager (downloadable at rvm.io)
- MySQL server and client
- Ruby 2.4.2
- Ruby on Rails 5.1.5
- OpenCV 2.3
- TensorFlow and TensorFlowGPU 1.7
    - Use GPU version (if applicable) to train faster
    - Consider using Anaconda to set up TensorFlow environment

When booting the robot, wait until the robot gives its initialization movement (dance) before adding any stimulus.

All of the code described below can be found in a directory called robo_code in the Raspberry Pi's home directory.

Main.py

Epoll (Linux kernel system call) loop is located in this program. This checks all of the file descriptors for the sockets of each of the other programs, and if any of the sockets is triggered, Main.py puts the task associated with that triggered socket into a thread pool. This functionality branches depending on which socket was triggered (i.e. if Sensors.py was triggered, Main.py will tell the robot to back up to avoid collision).

Robot.py

Robot.py contains the functions to interact with the PiHat. This includes being able to turn on/off pins that power the motors in different configurations (i.e. move forward, turn right, etc.). All locomotive calls go through Robot.py.

Sensors.py

Sensors.py keeps track of the pins associated with the encoders and those associated with the sonar sensors. In an infinite loop, this program tells the sensors to ping the environment and

calculates the distance generated from the pulses. If any of those distances are less than our threshold value, Sensors.py sends an alert to Main.py over sockets, where Main.py will handle the alert as already described.

Camera.py

This program first loads in the TensorFlow model. Then, it initializes the camera. Finally, it enters a loop of capturing an image with the camera, classifying the image to attempt to detect the green square or red circle; if it detects the green square or red circle, Camera.py calculates which direction the robot must turn to align itself with that signpost. It sends this direction, as well as the distance the robot needs to move in that direction to align itself (within a 10% tolerance), to Main.py via sockets. More signposts and/or objects can be trained through TensorFlow.

Web Application

There is a script on the Raspberry Pi called start_web_app that will reboot the web application. The Web App is using EngineX and Puma web server. It has a MySQL database as well, with no tables at this time. When the web application reboots, it connects to Main.py. From there, the user can connect to the web application and use the arrow keys or WASD keys to control the robot. In the terminal, the user can type “crontab -e” at reboot, which will initiate the script to start all of the programs previously described in this document.

TensorFlow Installation and Configuration

The following instructions to install and configure TensorFlow can be followed using either a native or virtual environment (Anaconda, Docker, Virtualenv, etc.).

The following steps outline the process I used to set up the TensorFlow environment on my system. The attached instructions are specific to a UNIX-based system since you will be setting this up on the Pi. The final product should be the same thing you saw on my system last week when we tested the object detection on any random object.

First, install the CPU version of TensorFlow:

```
pip install tensorflow
```

Then, install the following dependencies:

```
pip install pillow
```

```
pip install lxml
pip install jupyter
pip install matplotlib
```

After this, clone the following repo at https://github.com/tensorflow/models:

```
git clone https://github.com/tensorflow/models
```

Install the python version of protoc. I am using the 3.4.0 version. Once protoc is installed, run:

```
protoc object_detection/protos/*.proto --python_out=.
```

Followed by

```
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

If the above commands don't work, reinstall protoc (python version) and run:

```
sudo ./configure
sudo make check
sudo make install
```

Next, open terminal/cmd.exe from the models/object_detection directory and open the Jupyter Notebook with jupyter notebook. From here, choose the object_detection_tutorial.ipynb. From here, you should be able to cell in the main menu, and choose run all.

Now for VIDEO!

To begin, modify the notebook first by converting it to a .py file. To convert, you can go to file > save as > python file. Once that's done, you're going to want to comment out the get_ipython().magic('matplotlib inline') line. Also, make sure OpenCV is installed on the system as well.

Then, replace all of the code with the following:

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

import cv2
cap = cv2.VideoCapture(0)

# This is needed since the notebook is stored in the object_detection
folder.
sys.path.append("..")



# ## Object detection imports
# Here are the imports from the object detection module.


# In[3]:
```

```
from utils import label_map_util


from utils import visualization_utils as vis_util



# # Model preparation


# ## Variables

#

# Any model exported using the `export_inference_graph.py` tool can be
loaded here simply by changing `PATH_TO_CKPT` to point to a new .pb file.

#

# By default we use an "SSD with Mobilenet" model here. See the [detection
model
zoo](https://github.com/tensorflow/models/blob/master/object_detection/g3d
oc/detection_model_zoo.md) for a list of other models that can be run out-
of-the-box with varying speeds and accuracies.


# In[4]:


# What model to download.

MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'

MODEL_FILE = MODEL_NAME + '.tar.gz'

DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'


# Path to frozen detection graph. This is the actual model that is used
for the object detection.

PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'


# List of the strings that is used to add correct label for each box.

PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')


NUM_CLASSES = 90
```

```
# ## Download Model


# In[5]:


opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())



# ## Load a (frozen) Tensorflow model into memory.


# In[6]:


detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')



# ## Loading label map
# Label maps map indices to category names, so that when our convolution
network predicts `5`, we know that this corresponds to `airplane`.  Here
we use internal utility functions, but anything that returns a dictionary
mapping integers to appropriate string labels would be fine


# In[7]:
```

```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)

categories = label_map_util.convert_label_map_to_categories(label_map,max_
num_classes=NUM_CLASSES, use_display_name=True)

category_index = label_map_util.create_category_index(categories)



# ## Helper code


# In[8]:


def load_image_into_numpy_array(image):

  (im_width, im_height) = image.size

  return np.array(image.getdata()).reshape(

      (im_height, im_width, 3)).astype(np.uint8)



# # Detection


# In[9]:


# For the sake of simplicity we will use only 2 images:

# image1.jpg

# image2.jpg

# If you want to test the code with your images, just add path to the
images to the TEST_IMAGE_PATHS.

PATH_TO_TEST_IMAGES_DIR = 'test_images'

TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.f
ormat(i)) fori in range(1, 3) ]


# Size, in inches, of the output images.

IMAGE_SIZE = (12, 8)



# In[10]:
```

```
with detection_graph.as_default():
  with tf.Session(graph=detection_graph) as sess:
    while True:
      ret, image_np = cap.read()
      # Expand dimensions since the model expects images to have shape:
[1, None, None, 3]
      image_np_expanded = np.expand_dims(image_np, axis=0)
      image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
      # Each box represents a part of the image where a particular object
was detected.
      boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
      # Each score represent how level of confidence for each of the
objects.
      # Score is shown on the result image, together with the class label.
      scores = detection_graph.get_tensor_by_name('detection_scores:0')
      classes = detection_graph.get_tensor_by_name('detection_classes:0')
      num_detections = detection_graph.get_tensor_by_name('num_detections:
0')
      # Actual detection.
      (boxes, scores, classes, num_detections) = sess.run(
          [boxes, scores, classes, num_detections],
          feed_dict={image_tensor: image_np_expanded})
      # Visualization of the results of a detection.
      vis_util.visualize_boxes_and_labels_on_image_array(
          image_np,
          np.squeeze(boxes),
          np.squeeze(classes).astype(np.int32),
          np.squeeze(scores),
          category_index,
          use_normalized_coordinates=True,
          line_thickness=8)

      cv2.imshow('object detection', cv2.resize(image_np, (800,600)))
```

```
if cv2.waitKey(25) & 0xFF == ord('q'):
  cv2.destroyAllWindows()
  break
```

TensorFlow Training

The following TensorFlow tutorial will walk through installing TensorFlow if you need further assistance, as well as explaining how to train a custom model:

https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/