RC car using Raspberry Pi with object-detection

Project Report

Microprocessors and Interfacing– CSE2006

16BCB0073	Pranav Jain
16BCB0082	Kalyani Asthana
16BCB0088	B.Jaivarsan

B.Tech

in

Computer Science and Engineering

School of Computer Science & Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

1. Introduction

- **Motivation**

  Raspberry Pi is a computer that fits in your hand. It is a an extremely powerful tool because of its price and its capabilities. It can be used as a Desktop Computer as well an electronic device for implementing all kind of circuits, very similar to the Arduino series.
  We selected this project because we were curious about learning how to use the Raspberry Pi and testing it to its full capabilities.
  We know since our childhood that remotely controlled devices have existed since a long time. It's just that they have become more sophisticated over the years. We have easy to use platforms like Raspberry Pi and Arduino to make things better. We will be using a Raspberry Pi 3 model in out project.
  We tried to implement an object detection algorithm along with a simple RC car because of its far reached applications.

- **Significance of the project**

  The number of people that die because of terrorist attacks, earthquakes, fire hazard etc. is enormous. Because of its size, this RC car can reach small places, hidden passages, etc. and help save people.
  It will be remotely accessed by a computer through the SSH protocol on the same local area network. The range of its access can be increased based on the kind of protocol being used.
  It can find places where terrorists hide without human intervention and give back useful information about their base to governments across the world.

- **Scope of the project**

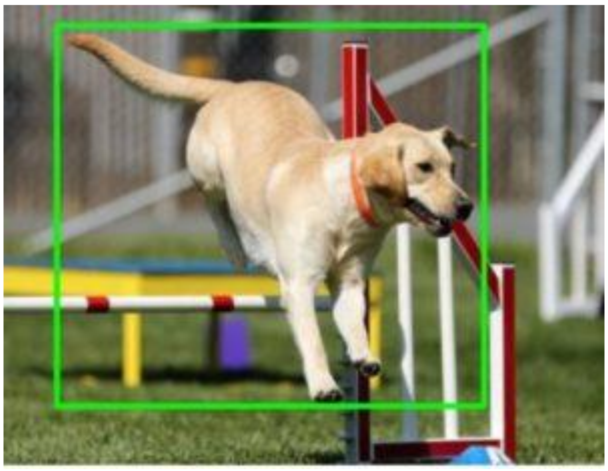
  Scope of the project can be extended to other physical designs such as a quadcopter implementing object-detection with Raspberry Pi.
  It can also be extended to swarm robots and other kinds of drones.

- **Application**

  One major application of this project is that it can reach places where humans can't and give important information back to the controller, so the controller can move it in any direction accordingly.
  For example, if there is an earthquake and people are stuck, the car can be moved to inaccessible places and aid rescue workers help people.

2. Literature Survey

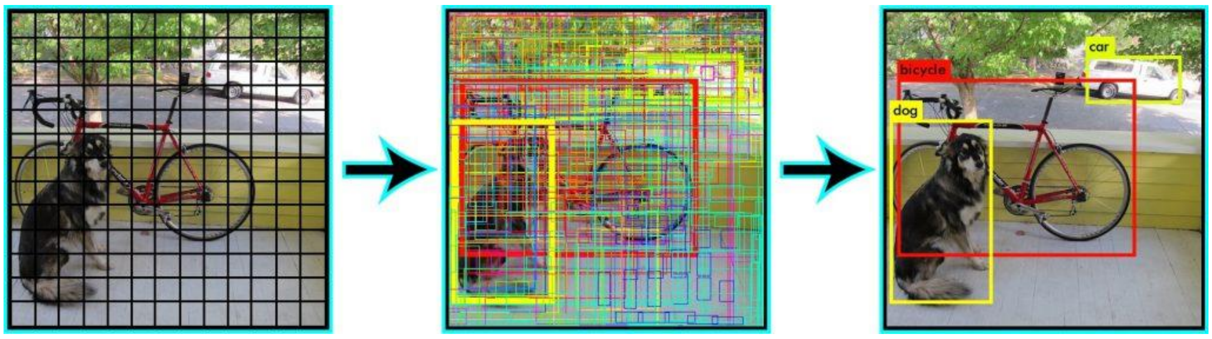
- **Existing Survey**



YOLO(You only Look Once)

For YOLO, detection is a simple regression problem which takes an input image and learns the class probabilities and bounding box coordinates. Sounds simple?

YOLO divides each image into a grid of S x S and each grid predicts N bounding boxes and confidence. The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an object(regardless of class). YOLO also predicts the classification score for each box for every class in training. You can combine both the classes to calculate the probability of each class being present in a predicted box.
So, total SxSxN boxes are predicted. However, most of these boxes have low confidence scores and if we set a threshold say 30% confidence, we can remove most of them as shown in the example below.



Notice that at runtime, we have run our image on CNN only once. Hence, YOLO is super fast and can be run real time. Another key difference is that YOLO sees the complete image at once as opposed to looking at only a generated region proposals in the previous methods. So, this contextual information helps in avoiding false positives. However, one limitation for YOLO is that it only predicts 1 type of class in one grid hence, it struggles with very small objects.

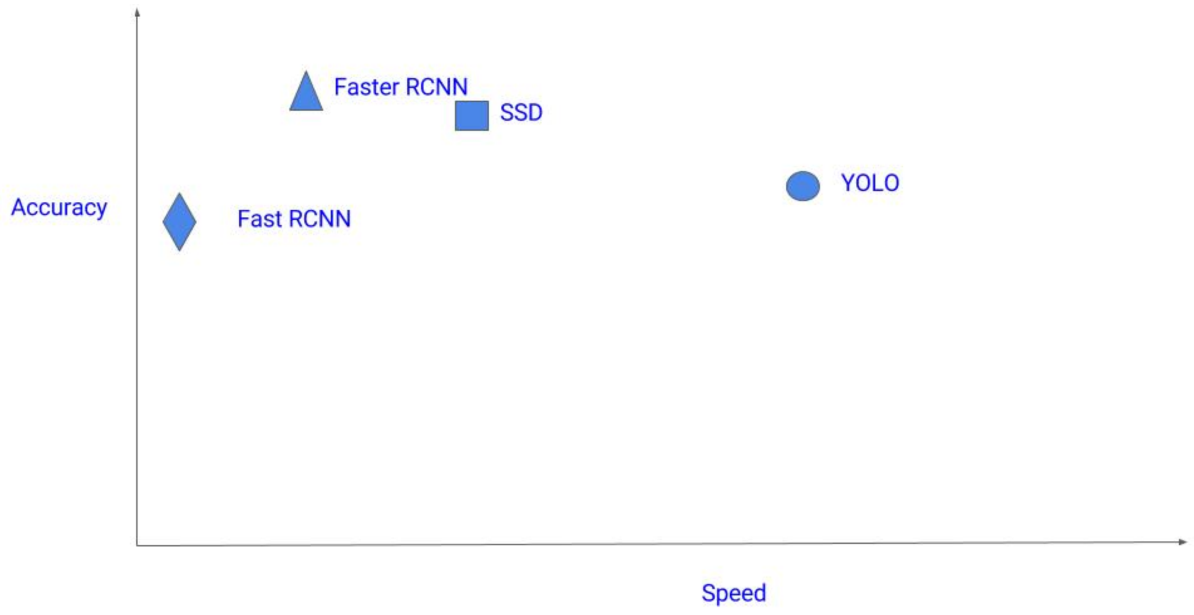
Single Shot Detector(SSD)

Single **S**hot **D**etector achieves a good balance between speed and accuracy. SSD runs a convolutional network on input image only once and calculates a feature map. Now, we run a small 3×3 sized convolutional kernel on this feature map to predict the bounding boxes and classification probability. SSD also uses anchor boxes at various aspect ratio similar to Faster-RCNN and learns the off-set rather than learning the box. In order to handle the scale, SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it is able to detect objects of various scales.

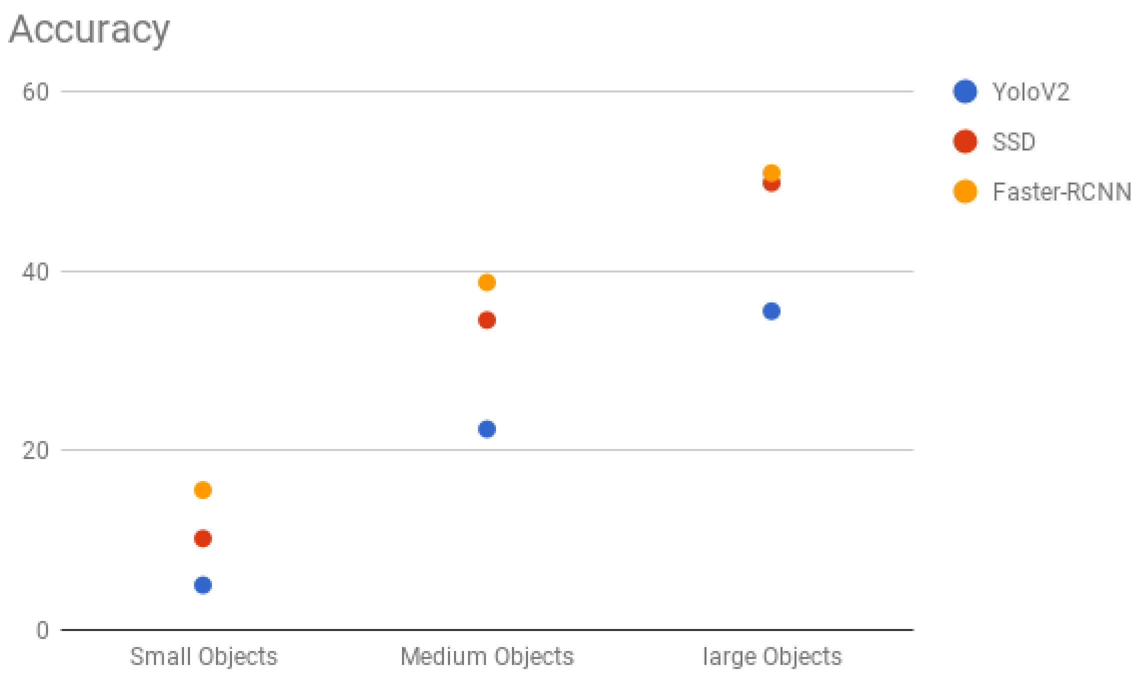
Choice of a right object detection method is crucial and depends on the problem you are trying to solve and the set-up. Object Detection is the backbone of many practical applications of computer vision such as autonomous cars, security and surveillance, and many industrial applications.

- **Gaps Identified**

That's a lot of algorithms. Which one should you use? Currently, Faster-RCNN is the choice if you are fanatic about the accuracy numbers. However, if you are strapped for computation(probably running it on Nvidia Jetsons), SSD is a better recommendation. Finally, if accuracy is not too much of a concern but you want to go super fast, YOLO will be the way to go. First of all a visual understanding of speed vs accuracy trade-off:



SSD seems to be a good choice as we are able to run it on a video and the accuracy trade-off is very little. However, it may not be that simple, look at this chart that compares the performance of SSD, YOLO, and Faster-RCNN on various sized objects. At large sizes, SSD seems to perform similarly to Faster-RCNN. However, look at the accuracy numbers when the object size is small, the gap widens.


Accuracy
60
40
20
0
Small Objects
Medium Objects
large Objects
YoloV2
SSD
Faster-RCNN

3. Design and Implementation

Hardware Components required

Raspberry Pi 3:

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside its target market or uses such as robotics. It does not include peripherals. However, some accessories have been included in several official and unofficial bundles.

DC motors:

A DC motor is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor.

Chassis and Wheels:

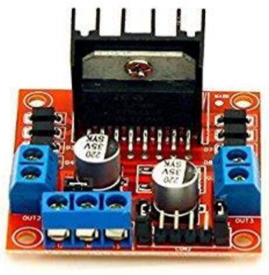
A chassis is the internal framework of an artificial object, which supports the object in its construction and use. An example of a chassis is a vehicle frame the underpart of a motor vehicle, on which the body is mounted; if the running gear such as wheels and transmission, and sometimes even the driver's seat, are included, then the assembly is described as a rolling chassis.

Jumper wires:

A jump wire (also known as jumper, jumper wire, jumper cable, wire, or DuPont cable – named for one manufacturer of them) is an electrical wire or group of them in a cable with a connector or pin at each end , which is normally used to interconnect the components of a breadboard or other prototype or test circuit, internally or with other equipment or components, without soldering.

Motor driver L298:

The *L298* is an integrated monolithic circuit in a 15-lead multi-1watt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors.



Power Bank(Output: 5V, 2.1A):

Portable *Power Banks* are comprised of a special battery in a special case with a special circuit to control *power* flow. They allow you to store electrical energy (deposit it in the *bank*) and then later use it to charge up a mobile device (withdraw it from the *bank*).

AAA Batteries(Output: 5V):

An AAA or triple-A battery is a standard size of dry cell battery commonly used in low-drain portable electronic devices. A zinc–carbon battery in this size is designated by IEC as "R03", by ANSI C18.1 as "24", by old JIS standard as "UM 4", and by other manufacturer and national standard designations that vary depending on the cell chemistry.

A triple-A battery is a single cell and measures 10.5 mm (0.41 in) in diameter and 44.5 mm (1.75 in) in length, including the positive terminal button, which is a minimum 0.8 mm (0.031 in). The positive terminal has a maximum diameter of 3.8 mm (0.15 in); the flat negative terminal has a minimum diameter of 4.3 mm (0.17 in).[1] Alkaline AAA batteries weigh around 11.5 grams , while primary lithium AAA batteries weigh about 7.6 g. Rechargeable nickel–metal hydride (NiMH) AAA batteries typically weigh 14–15 g.

Software Components required

Raspbian Operating system:

Raspbian is a Debian-based computer operating system for Raspberry Pi. There are several versions of Raspbian including Raspbian Stretch and Raspbian Jessie. Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers.

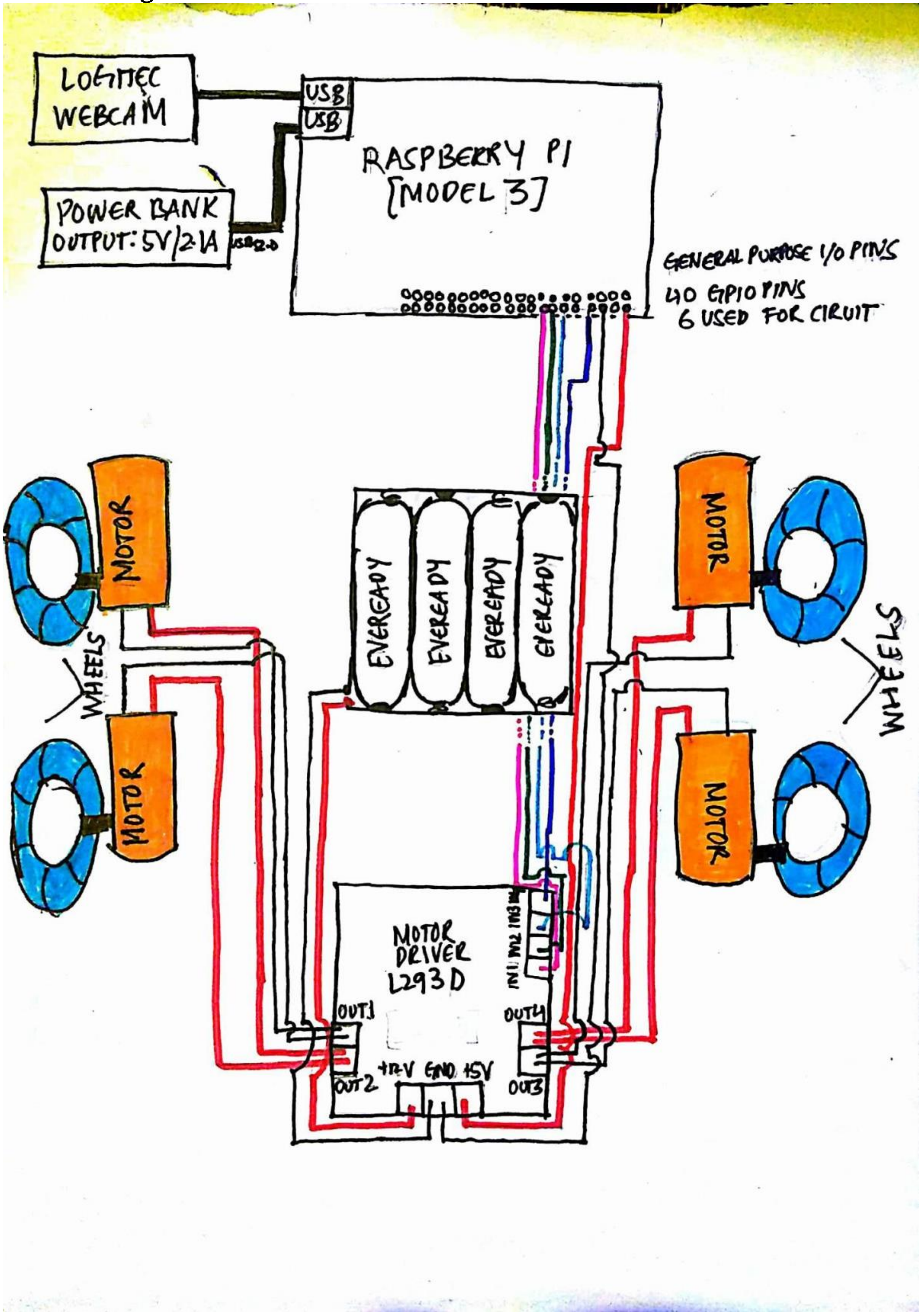
Python3 with the following libraries-RPI.GPIO, pygame, matplotlib, opencv, tensorflow, pillow:

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido von Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

VNC Viewer:

In computing, Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the Remote Frame Buffer Protocol(RBF) to remotely control another computer. It transmits the keyboard and mouse events from one computer to another, relaying the graphical screen updates back in the other direction, over a network.

Circuit Diagram

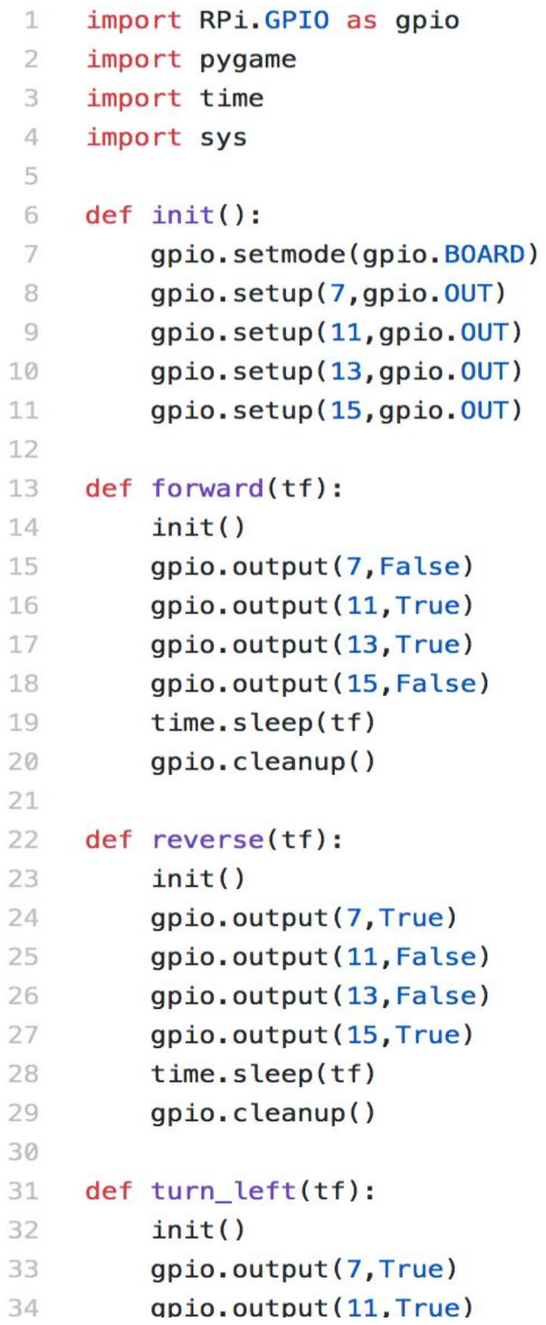


Implementation Details:

The connections are made as shown above.
Raspberry Pi command line interface is accessed via another machine using the SSH protocol.
It’s graphical user interface can be accessed as well using VNC viewer.
Functions for forward, backward, right, left, pivot right and pivot left movement of the car are written in a python script by using nano editor.
After testing each function separately, user control for executing the functions is added in the python script using the pygame library.
For executing this new script, Raspberry Pi is accessed via VNC viewer.
Up arrow is for forward motion, down arrow for backward, left arrow for left, right arrow for right, Q for pivot left and P for pivot right.
Code is written in python for object-detection using a webcam. This code requires the following libraries: tensorflow, opencv, matplotlib, pillow.
A separate terminal on the Raspberry Pi is opened to execute this code.
It will return a video feed while naming objects it sees.

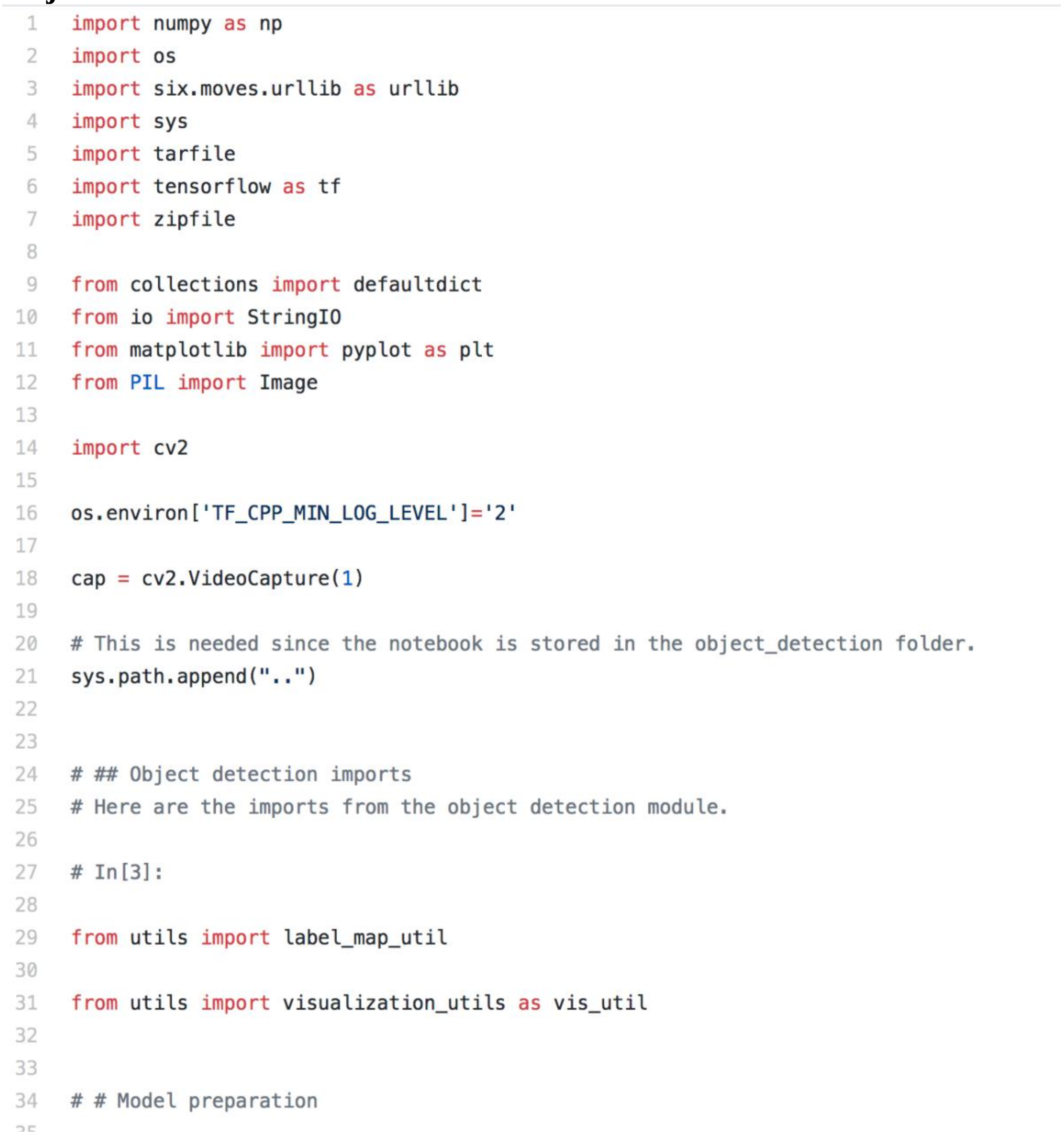
Code:

User Control:



```python
import RPi.GPIO as gpio
import pygame
import time
import sys

def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(7,gpio.OUT)
    gpio.setup(11,gpio.OUT)
    gpio.setup(13,gpio.OUT)
    gpio.setup(15,gpio.OUT)

def forward(tf):
    init()
    gpio.output(7,False)
    gpio.output(11,True)
    gpio.output(13,True)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()

def reverse(tf):
    init()
    gpio.output(7,True)
    gpio.output(11,False)
    gpio.output(13,False)
    gpio.output(15,True)
    time.sleep(tf)
    gpio.cleanup()

def turn_left(tf):
    init()
    gpio.output(7,True)
    gpio.output(11,True)
    gpio.output(11,True)
    gpio.output(13,True)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()

def turn_right(tf):
    init()
    gpio.output(7,False)
    gpio.output(11,True)
    gpio.output(13,False)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()

def pivot_left(tf):
    init()
    gpio.output(7,True)
    gpio.output(11,False)
    gpio.output(13,True)
    gpio.output(15,False)
    time.sleep(tf)
    gpio.cleanup()

def pivot_right(tf):
    init()
    gpio.output(7,False)
    gpio.output(11,True)
    gpio.output(13,False)
    gpio.output(15,True)
    time.sleep(tf)
    gpio.cleanup()

pygame.init()
size = (300,400)
pygame.display.set_mode(size)
```

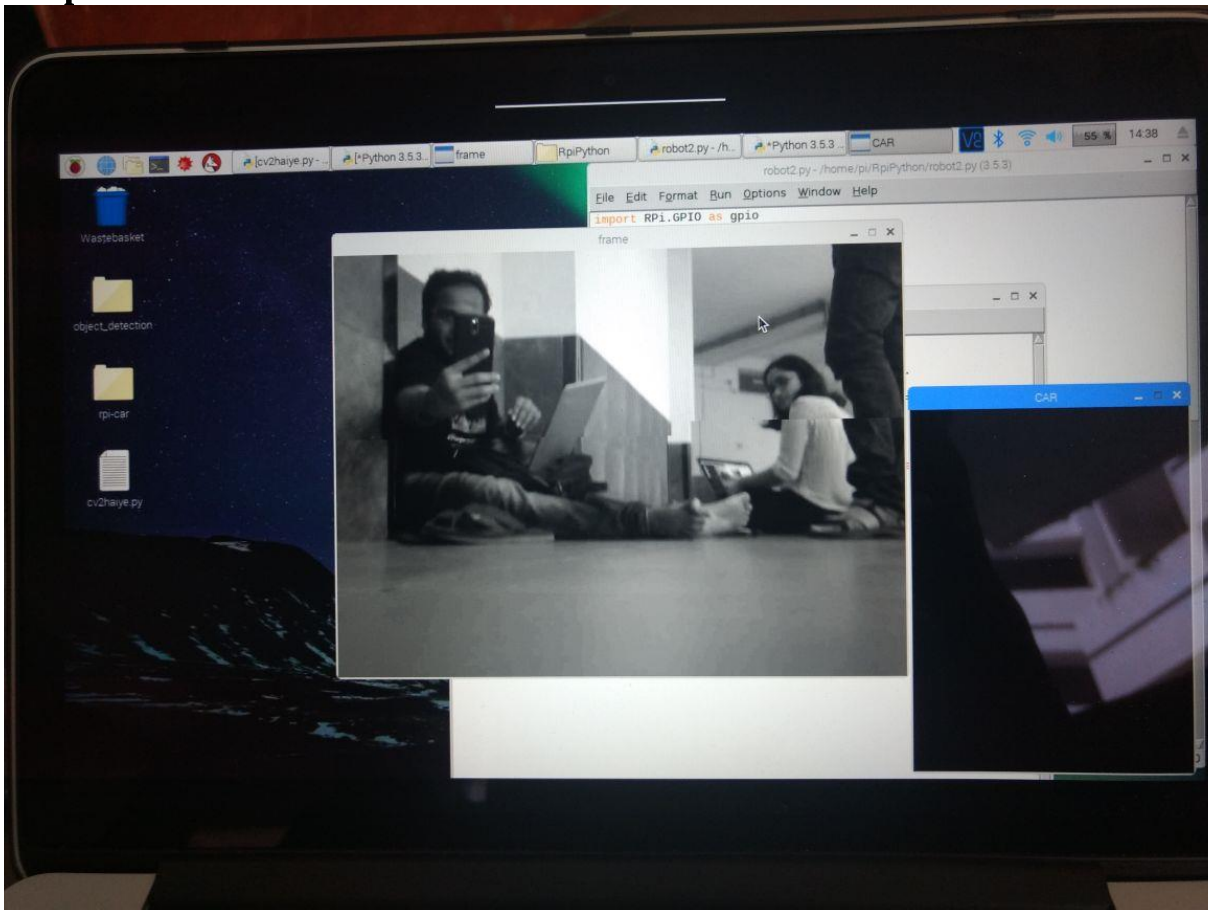
Object-detection:



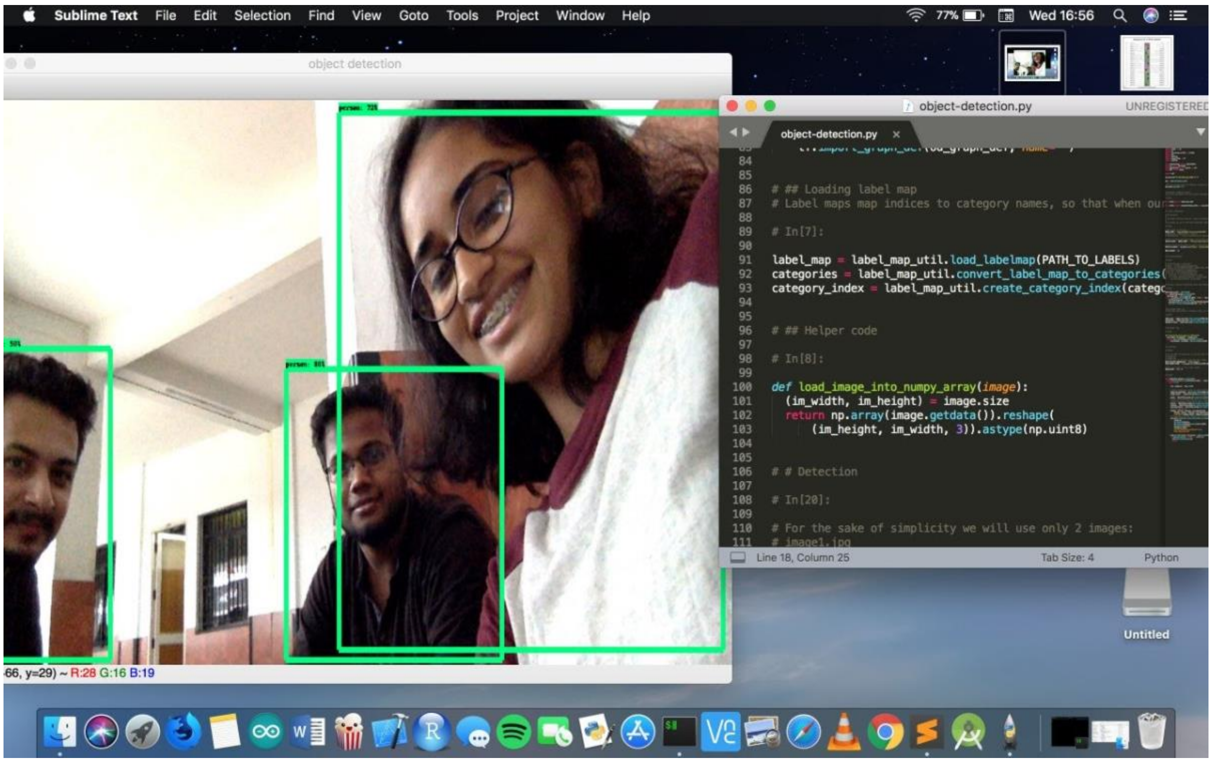
```python
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

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

import cv2

os.environ['TF_CPP_MIN_LOG_LEVEL']='2'

cap = cv2.VideoCapture(1)

# 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
# # Model preparation

# ## Variables
#
# Any model exported using the `export_inference_graph.py` tool can be loaded here simply by chan
#
# By default we use an "SSD with Mobilenet" model here. See the [detection model zoo](https://git

# 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]:

# print('Started to download')
# opener = urllib.request.URLopener()
# opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
# # print('Successfully downloaded')
# 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:
```

```
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

# 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)

PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(1, 3) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)


# In[21]:

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),
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
```

4. Snapshots


Wastebasket
object_detection
rpi-car
cv2haiye.py
frame
CAR
robot2.py - /home/pi/RpiPython/robot2.py (3.5.3)
File Edit Format Run Options Window Help

```
import RPi.GPIO as gpio
```



```
# ## Loading label map
# Label maps map indices to category names, so that when ou

# In[7]:

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(
category_index = label_map_util.create_category_index(catego


# ## 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[20]:

# For the sake of simplicity we will use only 2 images:
# image1.jpg
```

logitech

5. Conclusion

This is the very first step to autonomous cars or also known as self-driving cars. Leading tech giants like Uber[4], Tesla[5], NVIDIA[6], Google's Waymo[7] are already at the stage of done with trials and are already on the roads. Recently the first death because of a self-driving car was registered[8] and has caused a widespread impact on the autonomous cars. The problem wasn't because of the sensors[9] but it was just the object-detection algorithm, we can't expect it to work 100% but it will sooner or later happen. A self-driving car better than a human driving car isn't a dream that would never turn out true.

6. References

1. Wikipedia - https://www.wikipedia.com/
2. Joseph Redmon, Ali Farhadi, "YOLO9000: Better, Faster, Stronger", University of Washington
3. Gurjashan Singh Pannu, Mohammad Dawud Ansari, Pritha Gupta, "Design and Implementation of Autonomous car using Raspberry Pi", International Journal of Computer Applications, 2015.
4. Self Driving Cars in Pittsburgh - https://www.uber.com/cities/pittsburgh/self-driving-ubers/
5. Autopilot- Tesla - https://www.tesla.com/autopilot?redirect=no
6. Self Driving Cars Technology and Solutions by NVIDIA Automotive - https://www.nvidia.com/en-us/self-driving-cars/
7. Waymo - https://waymo.com/
8. Uber's Self-Driving Car Just Killed Somebody. Now What? - Aarian Marshall, WIRED
9. Sensor supplier says Lidar tech isn't to blame in fatal Arizona crash - Kyle Hyatt, CNET