A PROJECT REPORT ON

OBJECT DETECTION BY RASPBERRY PI 3 WITH OBSTACLE AVOIDING ROVER

SUBMITTED BY

ABHISHEK TYAGI

UNDER THE GUIDANCE OF

DR. SUSHIL CHANDRA

SCIENTIST 'G'

HOD BME

DEFARTMENT OF BIO-MEDICAL ENGINEERING

INSTITUE OF NUCLEAR MEDICINE AND ALLIED SCIENCES DEFENCE RESEARCH AND DEVELOPMENT ORGANISATION

<u>DECLARATION</u>

I hereby declare that the project work entitled **“Object detection by Raspberry Pi 3 with obstacle avoiding rover**” is an authentic record of my own work carried out at Institute of Nuclear Medicine and Allied Sciences (INMAS), Delhi as under the guidance of Dr. Sushil Chandra, Scientist ‘G’, Mr. Ram Singh, Scientist ‘C’, during June to July, 2019.

Date: - 07-2019

Certified that the above statement made by the student is correct to the best of my knowledge and belief.

Dr. Sushil Chandra
Scientist ' G ', (BME)
INMAS

ACKNOWLEDGEMENT

It gives me immense pleasure to express my sincerest regards and gratitude to **Dr. Ajay Kumar Singh, Director INMAS**, for providing us the opportunity to undergo our training at their prestigious organization. He and his team provided me with dedicated technical guidance and support at various stages of my training.

My humble and sincere thanks to my highly qualified and experienced guides **Dr. Sushil Chandra (Sci 'G' , Head of the department of BME), Mr. Ram Singh (Sci 'C') Mr. Manoj Prabhat (Technical Head) and Mr. Anmol Gupta (JRF)** and to all the members of Bio-Medical Engineering department (BME) for their guidance and direction, which helped me to work in my assigned field during the course of training.

My special thanks go to my friends whose support and encouragement have been a constant source of assurance, guidance, strength and inspiration to me. I am also immensely grateful to my family, who have always supported me and taught me the things that matter most in life and in supporting me to take up this training.

Abhishek Tyagi

DEFENCE RESEARCH AND DEVELOPMENT ORGANIZATION (DRDO)

The Defence Research and Development Organization (DRDO) is a defence initiative by the Republic of India, responsible for the military's innovative work in various fields, headquartered in New Delhi, India. It was framed in 1958 by the union of the Technical Development Establishment and the Directorate of Technical Development and Production with the Defence Science Organization. It works under the regulatory control of the Ministry of Defence, Government of India.

With a total of 52 labs, occupied with the advancement resistance innovation covering different fields, similar to flight, gadgets, arrive battle building, life sciences, materials, rockets, and maritime frameworks, DRDO is India's biggest and most vast research association. The association incorporates a total of more than 5,000 researchers with a place in Defence Research and Development Service (DRDS) and around 25,000 other logical, specialized and supporting work forces.

Vision

The key vision of DRDO is to work on making India prosperous by establishment of world-class science and technology bases and providing our Defence Services futuristic edge by equipping them with internationally competitive & technologically advanced systems and solutions.

Mission

Configuration, creation and prompt generation best in class sensors (various fields), weapon frameworks, stages and hardware for our Defence infrastructure. Provide innovative answers for the Defence Services for enhancement of battle viability and to advance prosperity of the troops. Create a secure foundation and conferred quality labor and assemble solid innovation base. 2

INSTITUTE OF NUCLEAR MEDICINE AND ALLIED SCIENCES (INMAS)

The Institute of Nuclear Medicine and Allied Sciences also known as INMAS, is a mandated multidisciplinary laboratory of DRDO engaged in R&D activities related to the field of Radiation, Imaging Sciences and CBRN technologies. INMAS is one of the very few Institutions that boast of having highly skilled and experienced scientists, clinical researchers and biomedical professionals working towards many common goals and objectives. One of the charters is the establishment of a center of excellence in biomedical & clinical research with special reference to neuro cognitive imaging, radiation and CBRN research. The main thrust areas of INMAS is the development of radio protectors, developing diagnostics and therapeutic approaches involving non-invasive imaging techniques, neuro cognitive and endocrine functional assessment of human body. INMAS has always taken a lead with academic developments in the fields of radiation and imaging sciences since its foundation. DRM course started under the aegis of Delhi University back in 1962 which was first of its kind in the world and continues so till date. Currently DNB course in Radiology and guest teaching of various courses at Delhi University are among the various additional academic activities that the institute is involved with. Academic accomplishments of INMAS have been continuously growing, with an increase in the number and high impact factor of research publications. A major development in the recent past years has been the establishment of NMR, PET Cyclotron facility that has made significant contributions in development of technologies for enhancing combat efficiency besides providing clinical research to the Armed forces. One of the charters is to establish as a center of excellence in biomedical & clinical research with special reference to radiation, neuro cognitive imaging and CBRN research. The main thrust areas of INMAS are development of radio protectors, development of diagnostics and therapeutic approaches using non-invasive imaging techniques, neuro cognitive and endocrine functional assessment of human body. INMAS has always taken a lead with academic developments in the areas of radiation and imaging sciences since its inception. The institute also pursues a strong research program with application in various stress related disorders using NMR techniques including metabolomics, in vivo spectroscopy and functional MRI (fMRI). INMAS has established a number of non-invasive and innovative Nuclear Medicine methods. For soldiers working at high altitudes a method for diagnosis, prevention and treatment of various ailments including cold injuries (frost bite) has been developed.

Deep Learning in Object Recognition, Detection, and Segmentation

Abstract As a major breakthrough in artificial intelligence, deep learning has achieved very impressive success in solving grand challenges in many fields including speech recognition, natural language processing, computer vision, image and video processing, and multimedia. This article provides a historical overview of deep learning and focus on its applications in object recognition, detection, and segmentation, which are key challenges of computer vision and have numerous applications to images and videos. The discussed research topics on object recognition include image classification on ImageNet, face recognition, and video classification. The detection part covers general object detection on ImageNet, pedestrian detection, face landmark detection (face alignment), and human landmark detection (pose estimation). On the segmentation side, the article discusses the most recent progress on scene labelling, semantic segmentation, face parsing, human parsing and saliency detection. Object recognition is considered as whole-image classification, while detection and segmentation are pixelwise classification tasks. Their fundamental differences will be discussed in this article. Fully convolutional neural networks and highly efficient forward and backward propagation algorithms specially designed for pixelwise classification task will be introduced. The covered application domains are also much diversified. Human and face images have regular structures, while general object and scene images have much more complex variations in geometric structures and layout. Videos include the temporal dimension. Therefore, they need to be processed with different deep models. All the selected domain applications have received tremendous attentions in the computer vision and multimedia communities. Through concrete examples of these applications, we explain the key points which make deep learning outperform conventional computer vision systems. (1) Different than traditional pattern recognition systems, which heavily rely on manually designed features, deep learning automatically learns hierarchical feature representations from massive training data and disentangles hidden factors of input data through multi-level nonlinear mappings. (2) Different than existing pattern recognition systems which sequentially design or train their key components, deep learning is able to jointly optimize all the components and crate synergy through close interactions among them. (3) While most machine learning models can be approximated with neural networks with shallow structures, for some tasks, the

expressive power of deep models increases exponentially as their architectures go deep. Deep models are especially good at learning global contextual feature representation with their deep structures. (4) Benefitting from the large learning capacity of deep models, some classical computer vision challenges can be recast as high-dimensional data transform problems and can be solved from new perspectives. Finally, some open questions and future works regarding to deep learning in object recognition, detection, and segmentation will be discussed.

Deep learning for Artificial Intelligence

We can now notice that with a massive amount of computational power, machines can now recognize objects and translate speech in real time, with this we can say artificial intelligence is getting better and smarter. The adaptations of machines and software to learn in a very real sense to recognize patterns in the digital representation of sounds, images or any type of data. when machine learning has many techniques in it, Deep learning stands out as very important one. There are many definitions of Deep learning, my favorite one being the formal definition for deep learning by Analytics Vidhya is defined as “Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.” Deep learning is a part of a family of machine learning, let's see how Deep learning came into existence and how it became more popular. As we know Machine learning uses algorithms to parse the data, memorizes it and learn from it, and then decide or prediction about the thing we are feeding the data in. It is very complex to hand code software with a specific set of instruction to accomplish a task in machine learning, so the machine is trained using copious amounts of data and algorithms that give it the ability to learn itself how to perform the task. The human brain contains billions of neurons which communicate each other for information sharing. With the same idea, the artificial neurons are created for the machine to make think and act like our brain. Using the neurons neural networks were created, the logic was to replicate the human brain in a machine. The representation of the biological neuron is given in the below fig: Machine learning directly enrooted from the minds who dreamt and gave birth of artificial intelligence. The logic was to replicate the human brain functionalities in the machine, so the concepts of neurons came into existence. The artificial neurons are represented as shown below: The algorithmic approaches over the years included decision tree learning, inductive logic programming are very initial ones. And there are algorithms like clustering, reinforcement learning, and Bayesian networks. These algorithms did not satisfy in achieving the goal of general AI. As we go more in-depth, we know that one of the best-suited application areas for machine learning for many years was computer vision, and though it still required a great deal of hand coding to get achieve the task. Somewhere even it was lacking the accuracy as an example of reading and recognizing the object during a sunny day and a foggy day. These layers learn through the models called ‘neural networks’, which is structured in layers one after another. Deep learning is called an artificial neural network (ANN), that is s network formed inspired by biological neural networks which can be used to approximate functions which have an enormous number of inputs that are rarely known The simple neural networks were lacking the accuracy, to make the system more robust and stronger in the aspect of human brain came to the multiple hidden layered networks called deep learning which is excellent till proven technique to implement machine learning. Most Deep learning methods use architecture as neural networks, so they are often referred to as a Deep neural network. The deep neural network is the name driven from the number of hidden layers in the network. The presentation of both simple and Deep learning is given in the fig. Deep learning models are trained on large sets of classified data and neural network architectures that learn features directly from the data without the need for manual feature extraction. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning.

Abstract

Efficient and accurate object detection has been an important topic in the advancement of computer vision systems. With the advent of deep learning techniques, the accuracy for object detection has increased drastically. The project aims to incorporate state-of-the-art technique for object detection with the goal of achieving high accuracy with a real-time performance. A major challenge in many of the object detection systems is the dependency on other computer vision techniques for helping the deep learning-based approach, which leads to slow and non-optimal performance. In this project, we use a completely deep learning-based approach to solve the problem of object detection in an end-to-end fashion. The network is trained on the most challenging publicly available dataset (PASCAL VOC), on which an object detection challenge is conducted annually. The resulting system is fast and accurate, thus aiding those applications which require object detection.

1 Introduction

1.1 Problem Statement

Many problems in computer vision were saturating on their accuracy before a decade. However, with the rise of deep learning techniques, the accuracy of these problems drastically improved. One of the major problems was that of image classification, which is defined as predicting the class of the image. A slightly complicated problem is that of image localization, where the image contains a single object and the system should predict the class of the location of the object in the image (a bounding box around the object). The more complicated problem (this project), of object detection involves both classification and localization. In this case, the input to the system will be an image, and the output will be a bounding box corresponding to all the objects in the image, along with the class of object in each box. An overview of all these problems is depicted in Fig. 1.

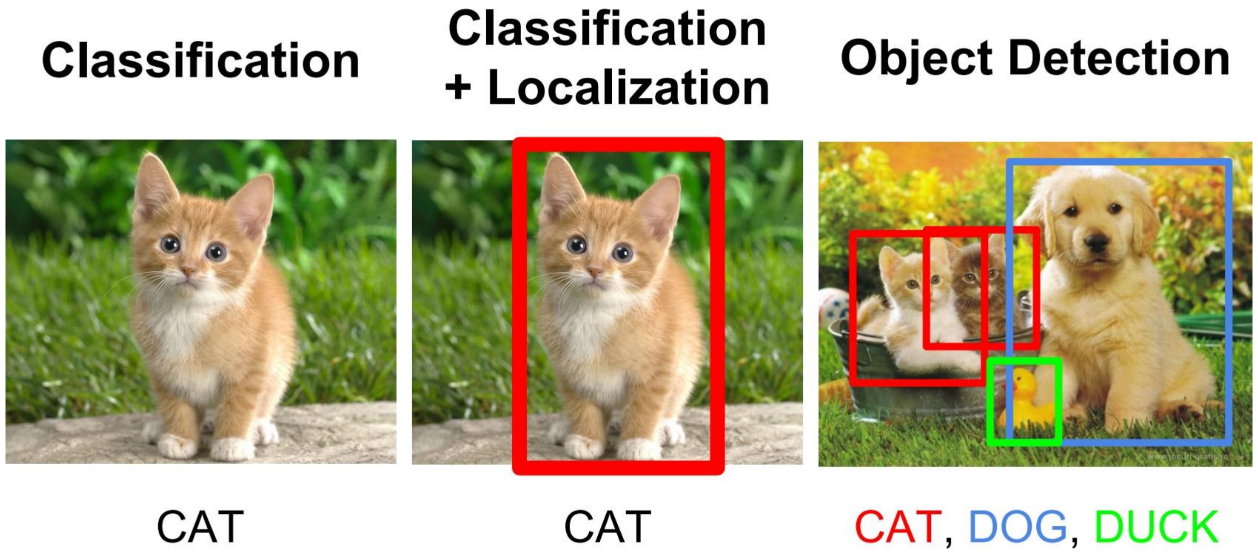
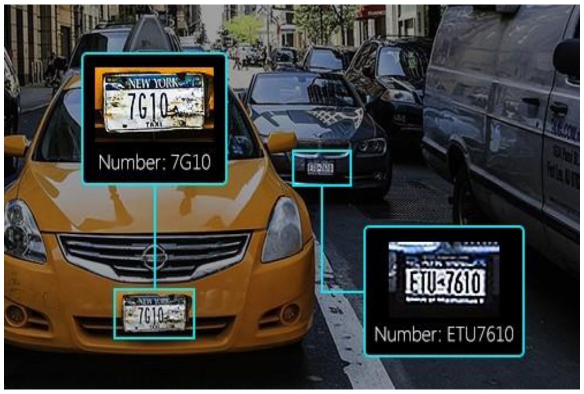


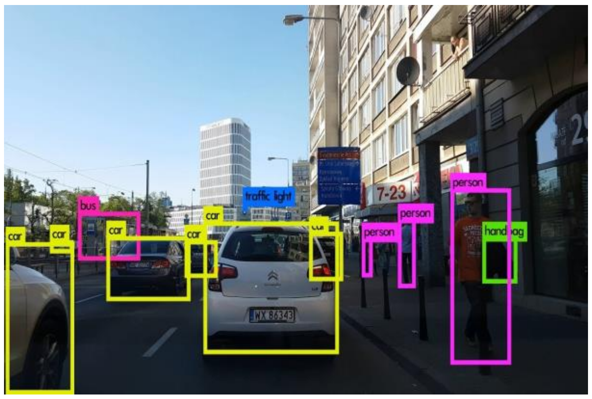
Figure 1: Computer Vision Tasks

1.2 Applications

A well-known application of object detection is face detection, that is used in almost all the mobile cameras. A more generalized (multi-class) application can be used in autonomous driving where a variety of objects need to be detected. Also, it has an important role to play in surveillance systems. These systems can be integrated with other tasks such as pose estimation where the first stage in the pipeline is to detect the object, and then the second stage will be to estimate pose in the detected region. It can be used for tracking objects and thus can be used in robotics and medical applications. Thus, this problem serves a multitude of applications.



(a) Surveillance



(b) Autonomous vehicles

Figure 2: Applications of object detections

1.3 Challenges

The major challenge in this problem is that of the variable dimension of the output which is caused due to the variable number of objects that can be present in any given input image. Any general machine learning task requires a fixed dimension of input and output for the model to be trained. Another important obstacle for widespread adoption of object detection systems is the requirement of real-time (30FPS) while being accurate in detection. The more complex the model is, the more time it requires for inference; and the less complex the model is, the less is the accuracy. This trade-off between accuracy and performance needs to be chosen as per the application. The problem involves classification as well as regression, leading the model to be learnt simultaneously. This adds to the complexity of the problem.

2 Related Work

There has been a lot of work in object detection using traditional computer vision techniques (sliding windows, deformable part models). However, they lack the accuracy of deep learning-based techniques. Among the deep learning based techniques, two broad class of methods are prevalent: two stage detection (RCNN , Fast RCNN , Faster RCNN) and unified detection (Yolo, SSD). The major concepts involved in these techniques have been explained below.

2.1 Bounding Box

The bounding box is a rectangle drawn on the image which tightly fits the object in the image. A bounding box exists for every instance of every object in the image. For the box, 4 numbers (center x, center y, width, height) are predicted. This can be trained using a distance measure between predicted and ground truth bounding box. The distance measure is a Jaccard distance which computes intersection over union between the predicted and ground truth boxes as shown in Fig. 3.

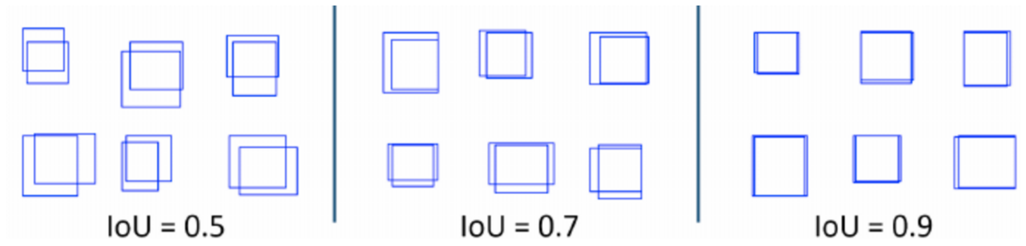


Figure 3: Jaccard distance

2.2 Classification + Regression

The bounding box is predicted using regression and the class within the bounding box is predicted using classification. The overview of the architecture is shown in Fig. 4

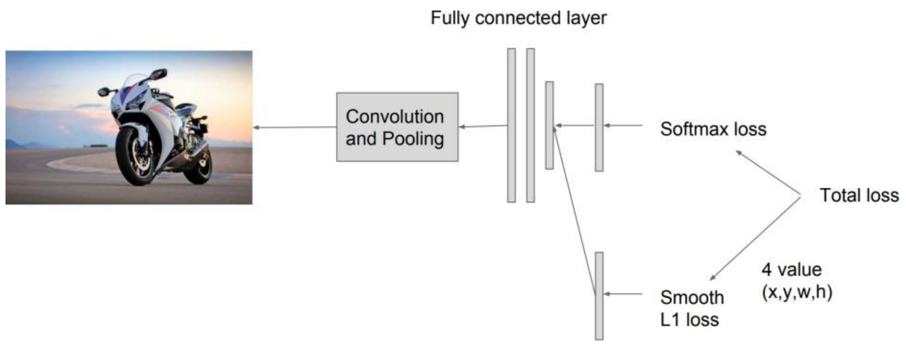
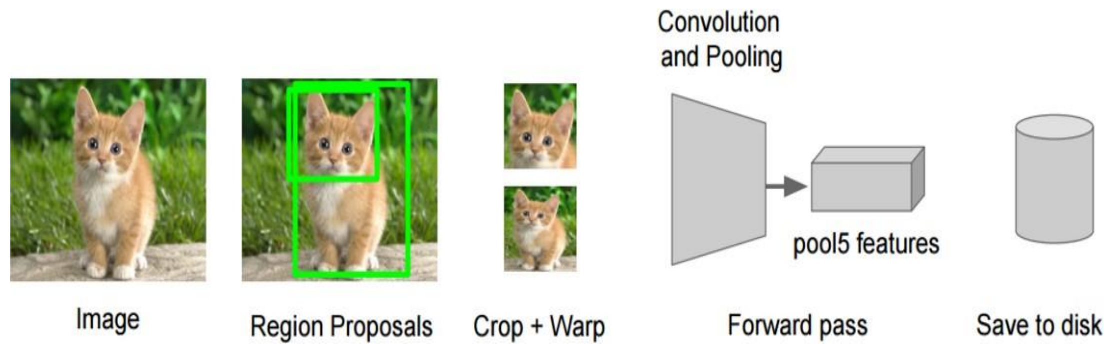


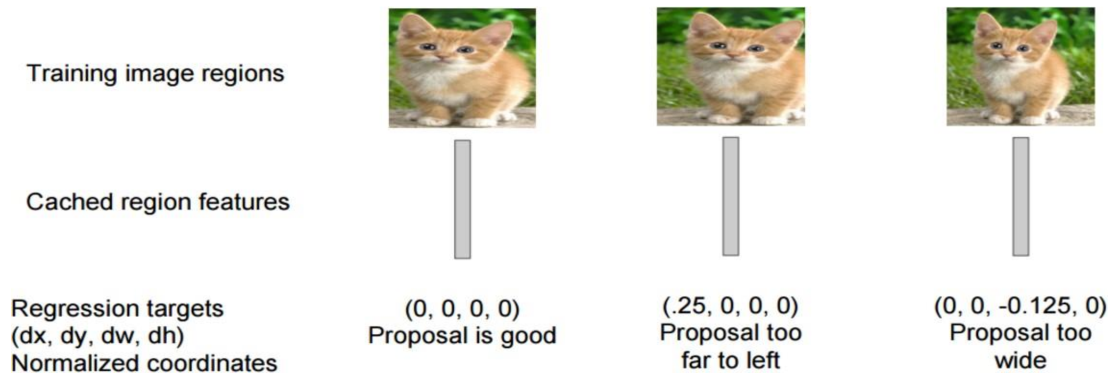
Figure 4: Architecture overview

2.3 Two-stage Method

In this case, the proposals are extracted using some other computer vision technique and then resized to fixed input for the classification network, which acts as a feature extractor. Then an SVM is trained to classify between object and background (one SVM for each class). Also, a bounding box regressor is trained that outputs some correction (offsets) for proposal boxes. The overall idea is shown in Fig. 5 These methods are very accurate but are computationally intensive (low fps).



(a) Stage 1


Training image regions
Cached region features
Regression targets (dx, dy, dw, dh) Normalized coordinates
(0, 0, 0, 0) Proposal is good
(.25, 0, 0, 0) Proposal too far to left
(0, 0, -0.125, 0) Proposal too wide

(b) Stage 2

Figure 5: Two stage method

2.4 Unified Method

The difference here is that instead of producing proposals, pre-define a set of boxes to look for objects. Using convolutional feature maps from later layers of the network, run another network over these feature maps to predict class scores and bounding box offsets. The broad idea is depicted in Fig. 6. The steps are mentioned below:

1. Train a CNN with regression and classification objective.
2. Gather activation from later layers to infer classification and location with a fully connected or convolutional layers.

3. During training, use Jaccard distance to relate predictions with the ground truth.
4. During inference, use non-maxima suppression to filter multiple boxes around the same object.

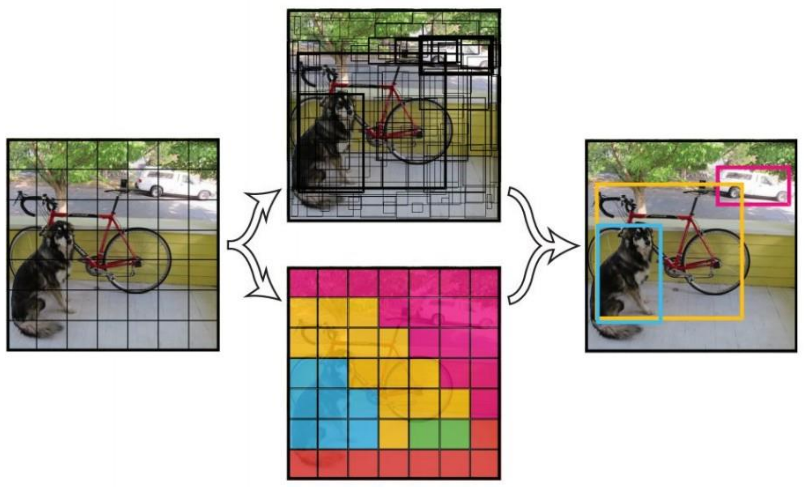


Figure 6: Unified Method

The major techniques that follow this strategy are: SSD (uses different activation maps (multiple-scales) for prediction of classes and bounding boxes) and Yolo (uses a single activation map for prediction of classes and bounding boxes). Using multiple scales helps to achieve a higher mAP(mean average precision) by being able to detect objects with different sizes on the image better. Thus, the technique used in this project is SSD.

3 Approach

The network used in this project is based on Single shot detection (SSD) . The architecture is shown in Fig. 7.

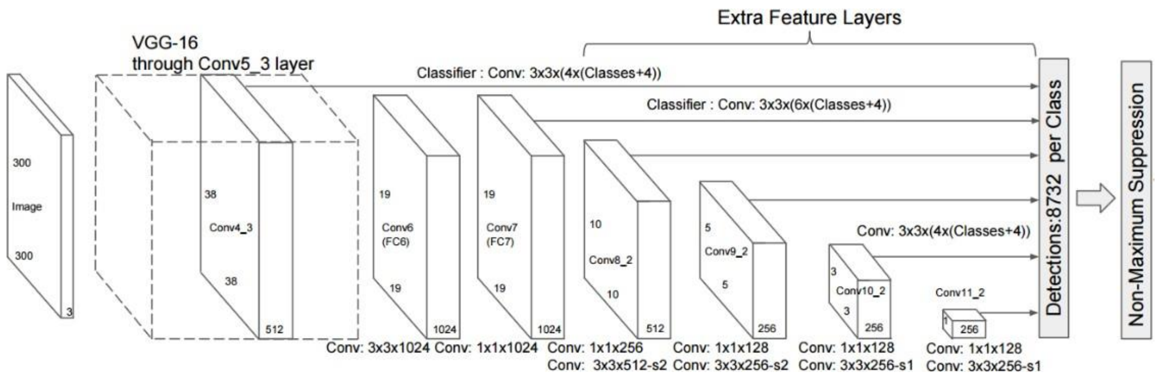


Figure 7: SSD Architecture

The SSD normally starts with a VGG [6] model, which is converted to a fully convolutional network. Then we attach some extra convolutional layers, that help to handle bigger objects. The output at the VGG network is a 38x38 feature map (conv4 3). The added layers produce 19x19, 10x10, 5x5, 3x3, 1x1 feature maps. All these feature maps are used for predicting bounding boxes at various scales (later layers responsible for larger objects).

Thus, the overall idea of SSD is shown in Fig. 8. Some of the activations are passed to the sub-network that acts as a classifier and a localizer.

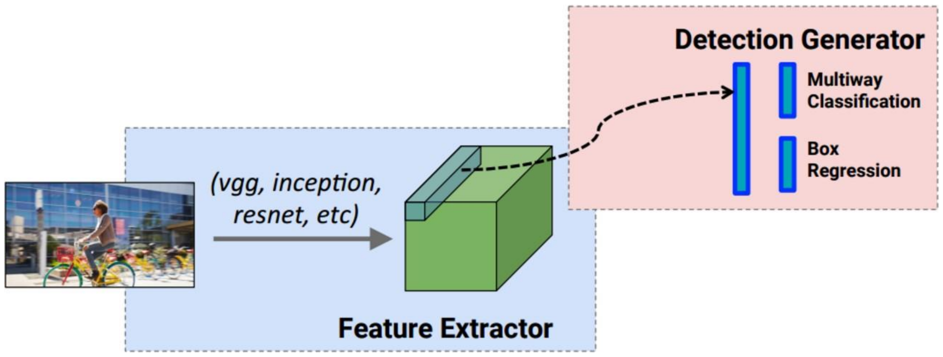


Figure 8: SSD Overall Idea

Anchors (collection of boxes overlaid on image at different spatial locations, scales and aspect ratios) act as reference points on ground truth images as shown in Fig. 9.

A model is trained to make two predictions for each anchor:

- A discrete class
- A continuous offset by which the anchor needs to be shifted to fit the ground-truth bounding box

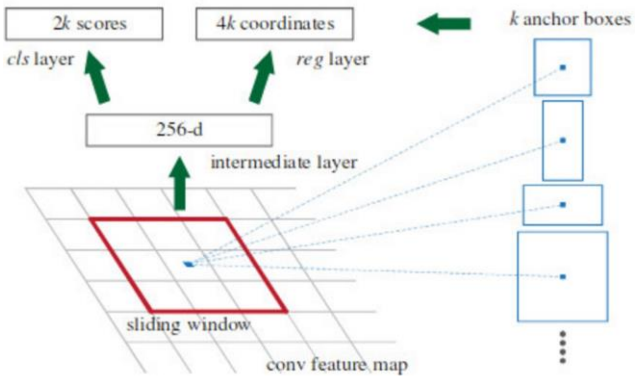


Figure 9: Anchors

During training SSD matches ground truth annotations with anchors. Each element of the feature map (cell) has a number of anchors associated with it. Any anchor with an IoU (Jaccard distance) greater than 0.5 is considered a match. Consider the case as shown in Fig. 10, where the cat has two anchors matched and the dog has one anchor matched. Note that both have been matched on different feature maps.

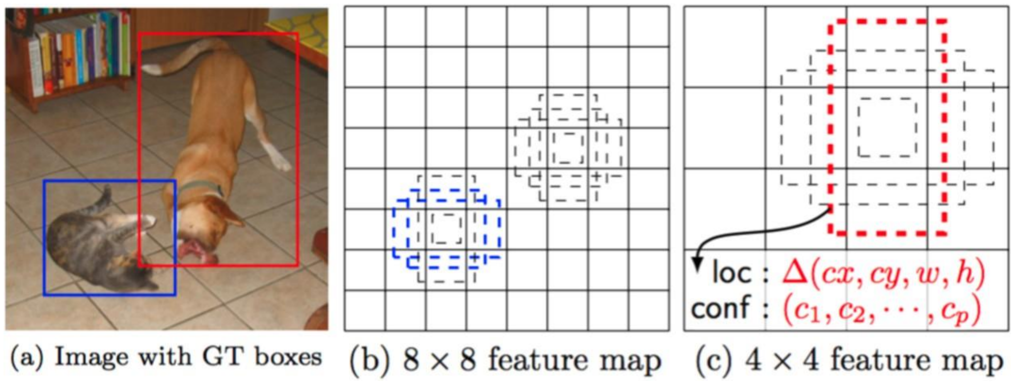


Figure 10: Matches

The loss function used is the multi-box classification and regression loss. The classification loss used is the SoftMax cross entropy and, for regression the smooth L1 loss is used.

During prediction, non-maxima suppression is used to filter multiple boxes per object that may be matched as shown in Fig. 11.

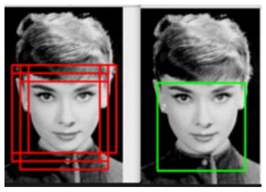


Figure 11: Non-maxima suppression

4 Experimental Results

4.1 Dataset

For the purpose of this project, the publicly available PASCAL VOC/SSD MobileNet V2 Lite (COCO) dataset will be used. It consists of 10k annotated images with 20 object classes with 25k object annotations (xml format). These images are downloaded from Flickr. This dataset is used in the PASCAL VOC Challenge which runs every year since 2006.

Visual Object Classes Challenge 2012 (VOC2012)
PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

Figure 12: Dataset

4.2 Implementation Details

The project is implemented in python 3. TensorFlow was used for training the deep network and OpenCV was used for image pre-processing.

The system specifications on which the model is trained and evaluated are mentioned as follows: Raspberry Pi 3 B model having 1 GB RAM, 1.4 Ghz Processor and 64GB HDD.

4.2.1 Pre-processing

The annotated data is provided in xml format, which is read and stored into a pickle file along with the images so that reading can be faster. Also, the images are resized to a fixed size.

4.2.2 Network

The entire network architecture is shown in Fig. 13. The model consists of the base network derived from VGG net and then the modified convolutional layers for fine-tuning and then the classifier and localizer networks. This creates a deep network which is trained end-to-end on the dataset.

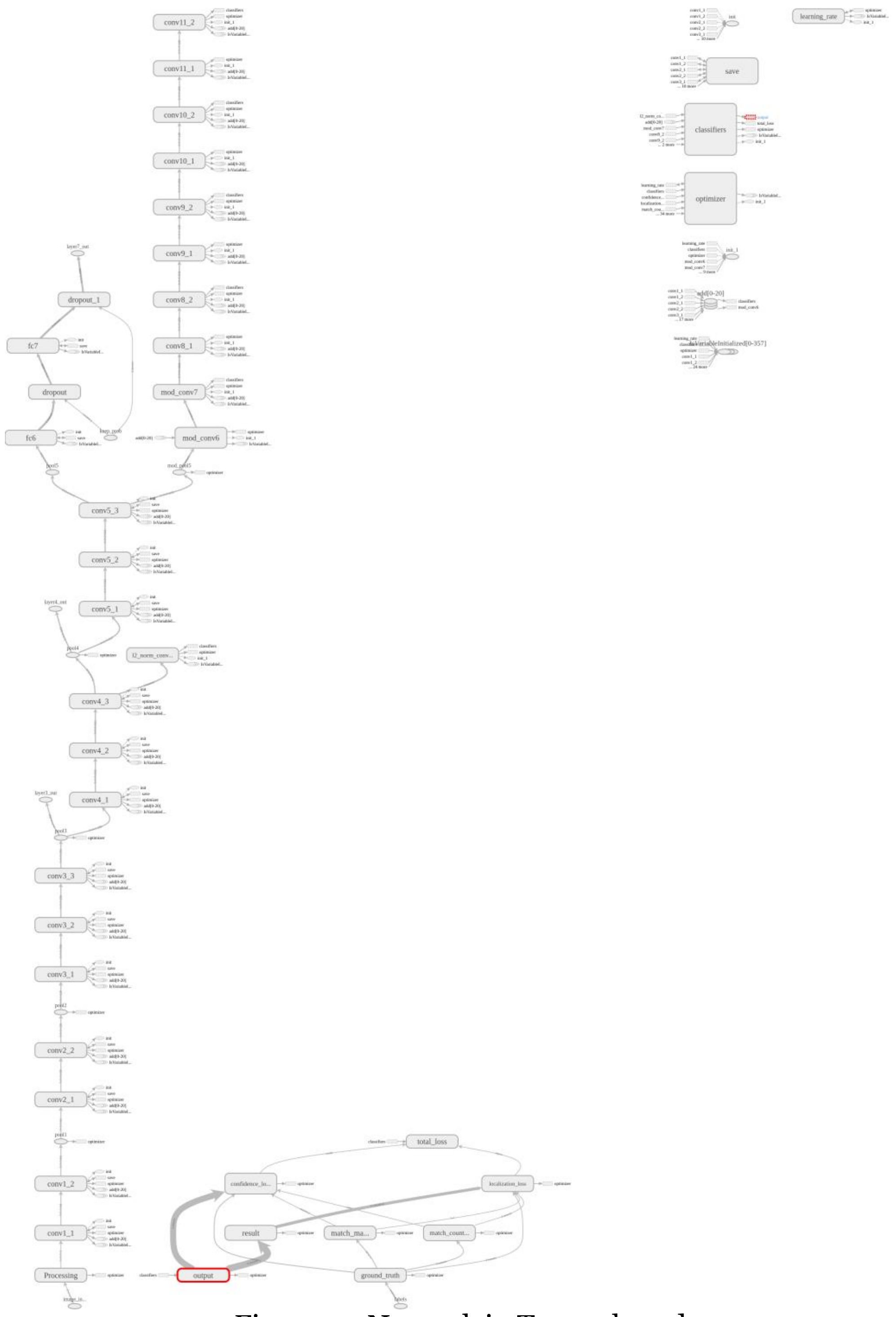


Figure 13: Network in Tensor board

4.3 Qualitative Analysis

The results from the PASCAL VOC dataset are shown in Table 1.

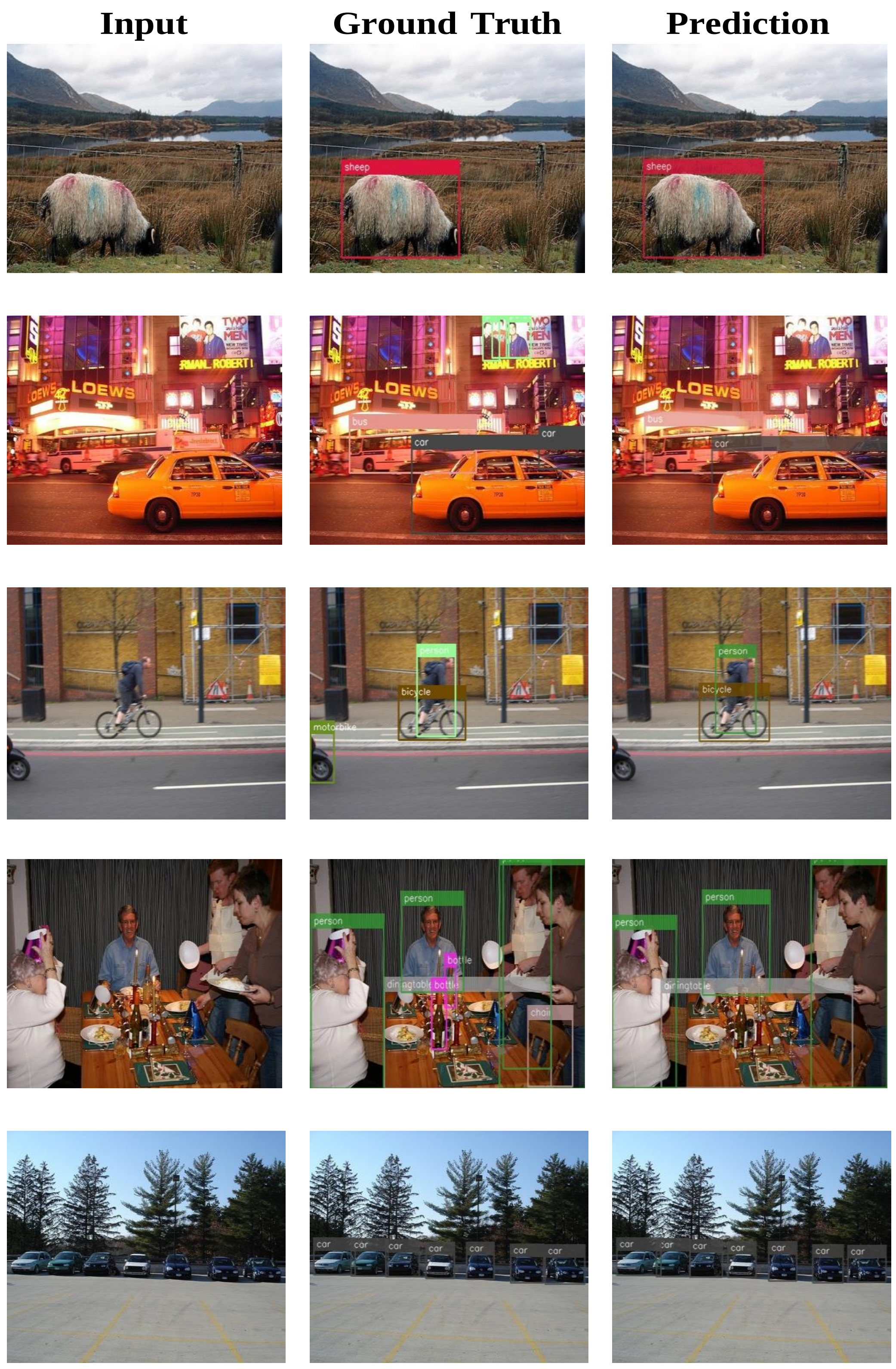


Table 1: Detection results on PASCAL VOC dataset.

The results on custom dataset are shown in Table 2.

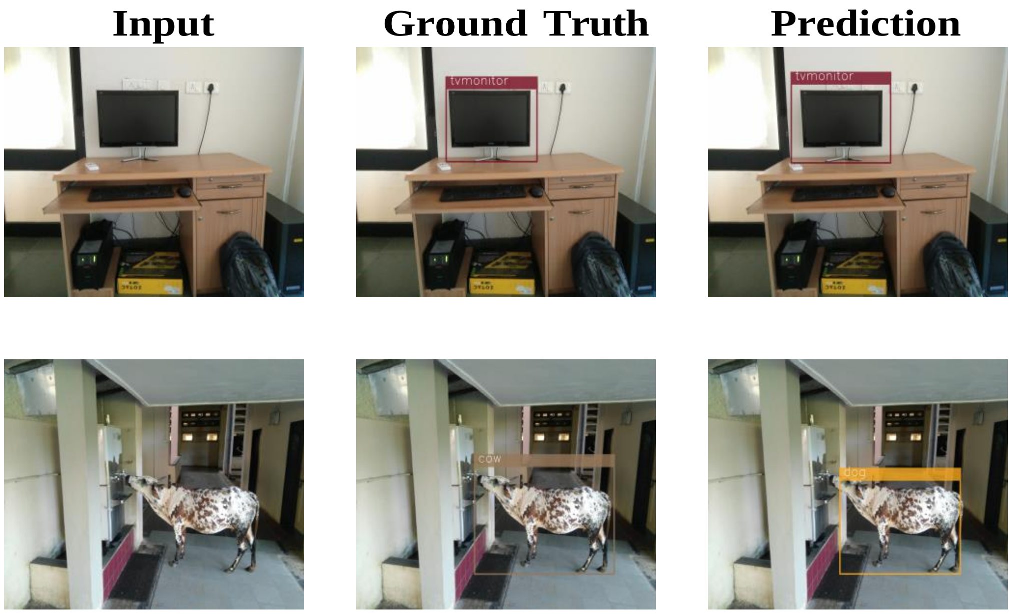
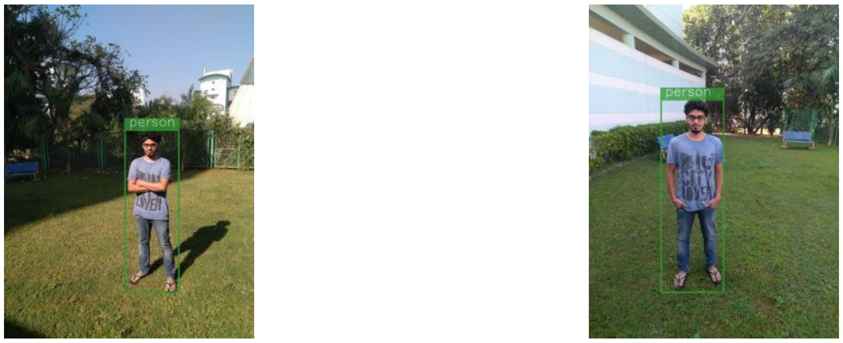


Table 2: Detection results on custom dataset.

The system handles illumination variations thus providing a robust detection. In Fig. 14 the same person is standing in the shade and then in the sunny environment.



(a) High illumination (b) Low illumination

Figure 14: Detection robust to illumination variation

However, occlusion creates a problem for detection. As shown in Fig. 15, the occluded birds are not detected correctly.

Also, larger object dominated when present along with small objects as found in Fig. 16. This could be the reason for the average precision of smaller objects to be less when compared to larger objects. This has been reported in the next section.

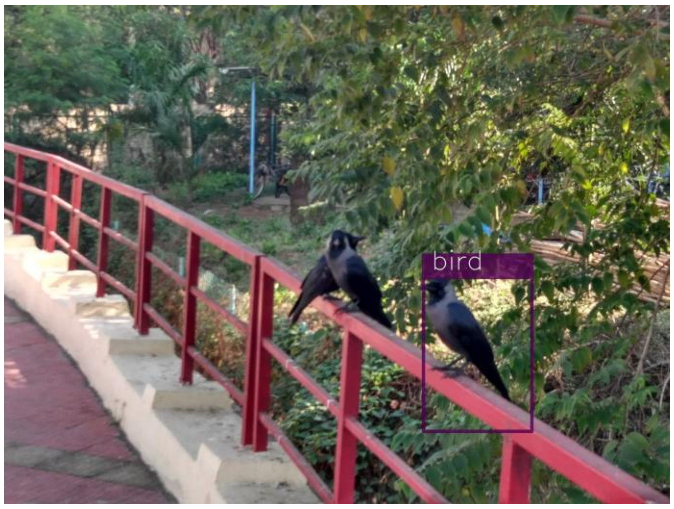
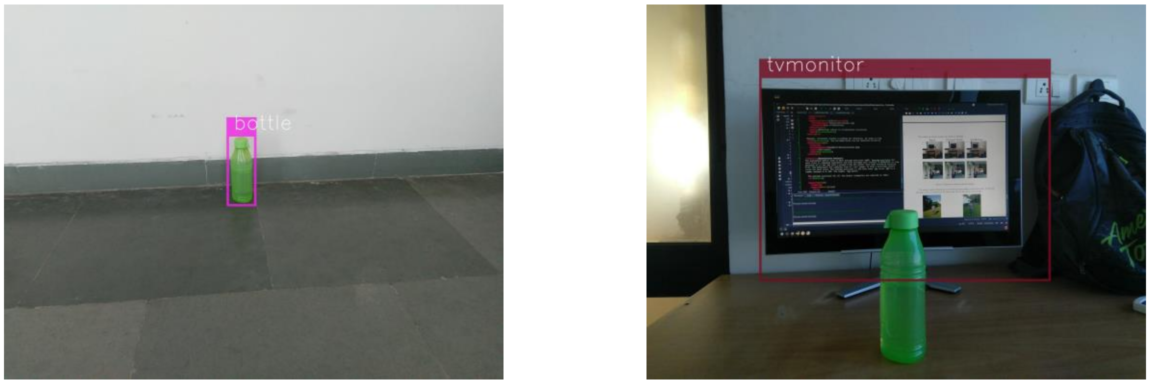


Figure 15: Occlusion



(a) Only small object in image (b) Small and large object in image

Figure 16: Domination of larger object in detection

4.4 Quantitative Analysis

The evaluation metric used is mean average precision (mAP). For a given class, precision-recall curve is computed. Recall is defined as the proportion of all positive examples ranked above a given rank. Precision is the proportion of all examples above that rank which are from the positive class. The AP summarizes the shape of the precision-recall curve, and is defined as the mean precision at a set of eleven equally spaced recall levels [0, 0.1, ... 1]. Thus, to obtain a high score, high precision is desired at all levels of recall. This measure is better than area under curve (AUC) because it gives importance to the sensitivity.

The detections were assigned to ground truth objects and judged to be true/false positives by measuring bounding box overlap. To be considered a correct detection, the area of overlap between the predicted bounding box and ground truth bounding box must exceed a threshold. The output of the detections assigned to ground truth objects satisfying the overlap criterion were ranked in order of (decreasing) confidence output. Multiple detections of the same object in an image were considered false detections, i.e. 5 detections of a single object counted as 1 true positive and 4 false positives. If no prediction is made for an image then it is considered a false negative.

The average precision for all the object categories is reported in Table 3. The mAP for the PASCAL VOC dataset was found to be 0.633. The current state-of-the-art best mAP value is reported to be 0.739.

Class	**Average Precision**
Motorbike	0.724
Bottle	0.272
Bird	0.635
Cat	0.909
Aero plane	0.727
Chair	0.360
Person	0.542
Dining table	0.534
Boat	0.544
Train	0.909
Sofa	0.710
Bicycle	0.636
Bus	0.726
Horse	0.726
TV monitor	0.633
Cow	0.632
Potted plant	0.359
Car	0.634
Dog	0.818
Sheep	0.633

Table 3: Average precision for all classes

5 Conclusion

An accurate and efficient object detection system has been developed which achieves comparable metrics with the existing state-of-the-art system. This project uses recent techniques in the field of computer vision and deep learning. Custom dataset was created using labelImg and the evaluation was consistent. This can be used in real-time applications which require object detection for pre-processing in their pipeline.

An important scope would be to train the system on a video sequence for usage in tracking applications. Addition of a temporally consistent network would enable smooth detection and more optimal than per-frame detection.

Programming Using Python (IDLE)-

TensorFlow object detection on Raspberry Pi 3 Model B using Pi-Camera/USB Camera

Python Program-

```
# Description:
# This program uses a TensorFlow classifier to perform object detection.
# It loads the classifier uses it to perform object detection on a Picamera feed.
# It draws boxes and scores around the objects of interest in each frame from
# the Picamera. It also can be used with a webcam by adding "--usbcam"
# when executing this script from the terminal.

## Some of the code is copied from Google's example at
##
https://github.com/tensorflow/models/blob/master/research/object_detection/object_detecti
on_tutorial.ipynb

## and some is copied from Dat Tran's example at
## https://github.com/datitran/object_detector_app/blob/master/object_detection_app.py

## but I changed it to make it more understandable to me.


# Import packages
import os
import cv2
import numpy as np
from picamera.array import PiRGBArray
from picamera import PiCamera
import tensorflow as tf
import argparse
import sys

# Set up camera constants
IM_WIDTH = 1280
IM_HEIGHT = 720
#IM_WIDTH = 640    Use smaller resolution for
#IM_HEIGHT = 480   slightly faster framerate

# Select camera type (if user enters --usbcam when calling this script,
# a USB webcam will be used)
camera_type = 'picamera'
parser = argparse.ArgumentParser()
```

```
parser.add_argument('--usbcam', help='Use a USB webcam instead of picamera',
action='store_true')
args = parser.parse_args()
if args.usbcam:
camera_type = 'usb'

# This is needed since the working directory is the object_detection folder.
sys.path.append('..')

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'ssdlite_mobilenet_v2_coco_2018_05_09'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'data','mscoco_label_map.pbtxt')

# Number of classes the object detector can identify
NUM_CLASSES = 90

## Load the label map.
# Label maps map indices to category names, so that when the 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
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)

# Load the Tensorflow model into memory.
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='')

sess = tf.Session(graph=detection_graph)


# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()
font = cv2.FONT_HERSHEY_SIMPLEX

# Initialize camera and perform object detection.
# The camera has to be set up and used differently depending on if it's a
# Picamera or USB webcam.

# I know this is ugly, but I basically copy+pasted the code for the object
# detection loop twice, and made one work for Picamera and the other work
# for USB.

### Picamera ###
if camera_type == 'picamera':
# Initialize Picamera and grab reference to the raw capture
camera = PiCamera()
camera.resolution = (IM_WIDTH,IM_HEIGHT)
camera.framerate = 10
rawCapture = PiRGBArray(camera, size=(IM_WIDTH,IM_HEIGHT))
```

```
rawCapture.truncate(0)

for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port=True):

t1 = cv2.getTickCount()
# Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]
# i.e. a single-column array, where each item in the column has the pixel RGB value
frame = np.copy(frame1.array)
frame.setflags(write=1)
frame_expanded = np.expand_dims(frame, axis=0)

# Perform the actual detection by running the model with the image as input
(boxes, scores, classes, num) = sess.run(
[detection_boxes, detection_scores, detection_classes, num_detections],
feed_dict={image_tensor: frame_expanded})

# Draw the results of the detection (aka 'visulaize the results')
vis_util.visualize_boxes_and_labels_on_image_array(
frame,
np.squeeze(boxes),
np.squeeze(classes).astype(np.int32),
np.squeeze(scores),
category_index,
use_normalized_coordinates=True,
line_thickness=8,
min_score_thresh=0.40)

cv2.putText(frame,"FPS:
{0:.2f}".format(frame_rate_calc),(30,50),font,1,(255,255,0),2,cv2.LINE_AA)

# All the results have been drawn on the frame, so it's time to display it.
cv2.imshow('Object detector', frame)

t2 = cv2.getTickCount()
time1 = (t2-t1)/freq
frame_rate_calc = 1/time1

# Press 'q' to quit
if cv2.waitKey(1) == ord('q'):
break

rawCapture.truncate(0)

camera.close()
```

```
### USB webcam ###
elif camera_type == 'usb':
# Initialize USB webcam feed
camera = cv2.VideoCapture(0)
ret = camera.set(3,IM_WIDTH)
ret = camera.set(4,IM_HEIGHT)

while(True):

t1 = cv2.getTickCount()

# Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]
# i.e. a single-column array, where each item in the column has the pixel RGB value
ret, frame = camera.read()
frame_expanded = np.expand_dims(frame, axis=0)

# Perform the actual detection by running the model with the image as input
(boxes, scores, classes, num) = sess.run(
[detection_boxes, detection_scores, detection_classes, num_detections],
feed_dict={image_tensor: frame_expanded})

# Draw the results of the detection (aka 'visulaize the results')
vis_util.visualize_boxes_and_labels_on_image_array(
frame,
np.squeeze(boxes),
np.squeeze(classes).astype(np.int32),
np.squeeze(scores),
category_index,
use_normalized_coordinates=True,
line_thickness=8,
min_score_thresh=0.85)

cv2.putText(frame,"FPS:
{0:.2f}".format(frame_rate_calc),(30,50),font,1,(255,255,0),2,cv2.LINE_AA)
# All the results have been drawn on the frame, so it's time to display it.
cv2.imshow('Object detector', frame)

t2 = cv2.getTickCount()
time1 = (t2-t1)/freq
frame_rate_calc = 1/time1

# Press 'q' to quit
if cv2.waitKey(1) == ord('q'):
```

```
Break

camera.release()

cv2.destroyAllWindows()
```

CONSTRUCTION OF OBSTACLE AVOIDING ROVER USING RASPBERRY PI 3 B WITH

Principle of Operation

The main principle in controlling a DC Motor with Raspberry Pi lies with the Motor Driver. A Motor Driver is a special circuit or IC that provides the necessary power (or rather the current) to the motor for smooth and safe operation.

Even a small 5V DC Motor draws a high initial current of around 300 – 400 mA. This current will then fall down 150 – 200 mA as the motor gains speed to around.

This is a huge current for devices like Microcontrollers, Arduino, Raspberry Pi etc. Hence, we should never connect a motor directly to Raspberry Pi (or any other microcontroller).

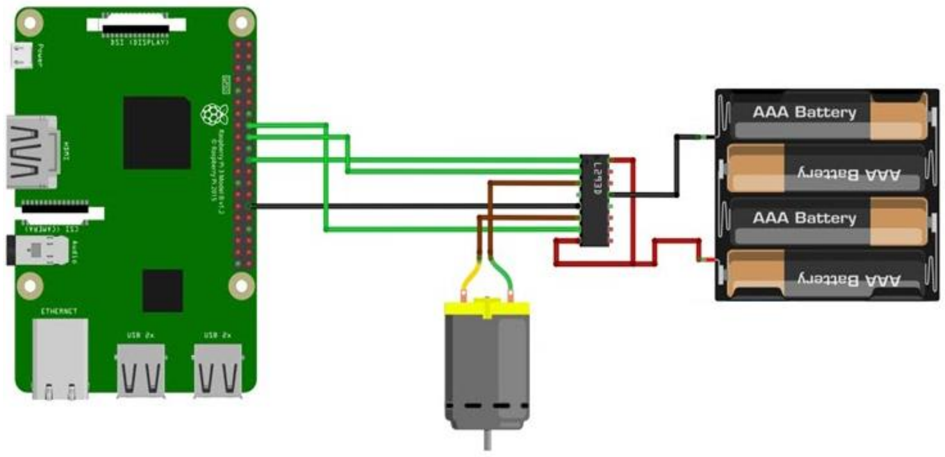
Motor Driver play an important role in this situation. They take the control signals from Raspberry Pi and provide the necessary drive current to the motor from the power supply.

In this project, the motor driver (L293D) is given with two control signals from Raspberry Pi through GPIO Pins. As per the Python Program, the motor will rotate in either forward or reverse direction.

Circuit Diagram

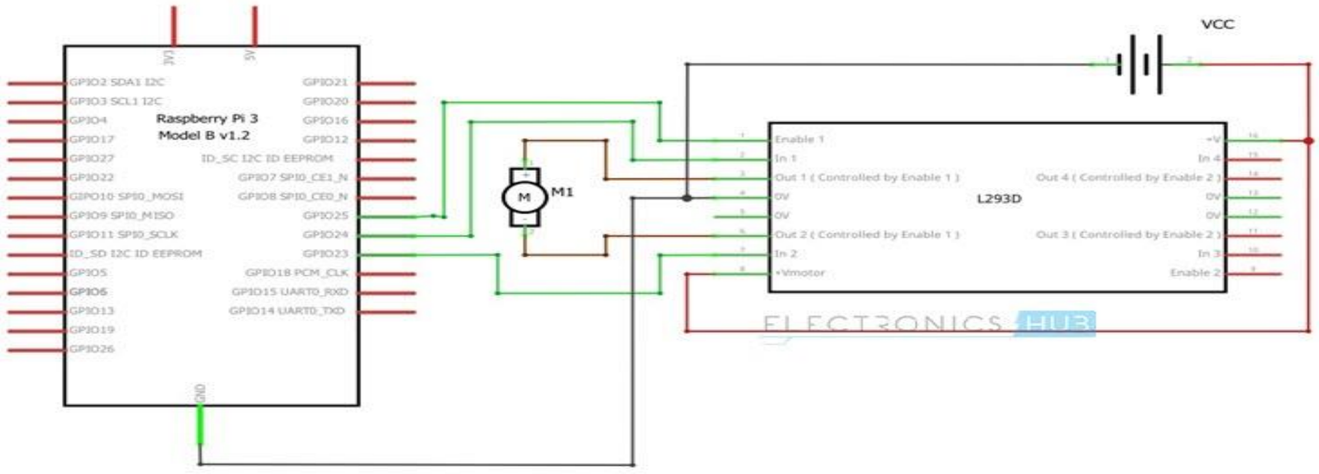
Fritzing Image

As I have said earlier, with L293D Motor Driver IC, we can actually control two motors. For simplicity reasons, I'll demonstrate the circuit, working and program for controlling a single DC Motor with Raspberry Pi. The following image is the Fritzing diagram of the project.



Circuit Diagram

The circuit wiring diagram of the project is shown below. You can easily configure this circuit and the program for controlling two DC Motors with Raspberry Pi and L293D Motor Driver IC.



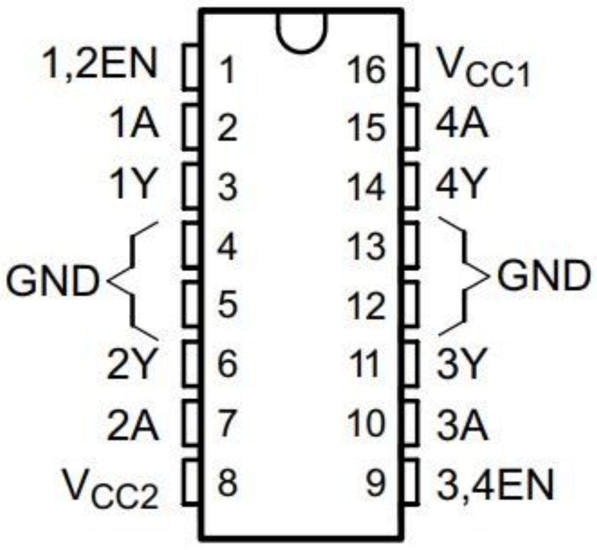
Components Required:

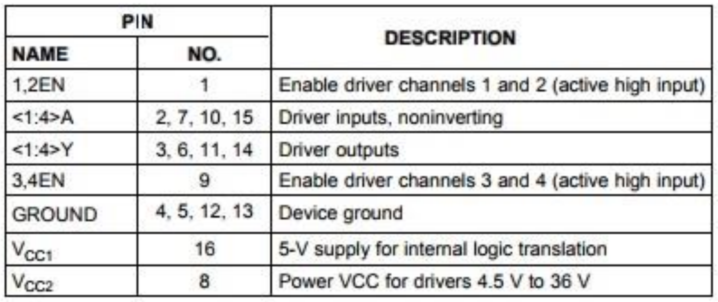
- Raspberry Pi
- Ultrasonic Sensor Module HC-SR04
- Chassis
- DC Motors
- L293D Motor Controller
- Wheels
- PCB
- Resistor (1k)
- Connecting wires
- Power supply or Power bank

Brief Note on L293D Motor Driver IC

I’ve used L293D Motor Driver IC for controlling a DC Motor with Raspberry Pi. It is a very common motor driver IC which is capable of driving two motors with individual currents up to 600mA.

The Pin diagram of the L293D Motor Driver IC, along with the pin description is shown in the following image.





PIN		DESCRIPTION
NAME	NO.	
1,2EN	1	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	Driver outputs
3,4EN	9	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	Device ground
V_{CC1}	16	5-V supply for internal logic translation
V_{CC2}	8	Power VCC for drivers 4.5 V to 36 V

Circuit Design

The design of the circuit for controlling a DC Motor with Raspberry Pi is very simple. First, connect the pins 8 and 16 (VCC2 and VCC1) of L293D to external 5V supply (assuming you are using a 5V Motor).

There are four ground pins on L293D. Connect pin 4 to the GND of supply. Also, connect the ground pin of L293D to GND pin of the Raspberry Pi.

Finally, we have the enable and control input pins. Connect the pin 1 of L293D (1,2EN) to GPIO25 (Physical Pin 22) of Raspberry Pi. Then connect control input pins 2 and 7 (1A and 2A) to GPIO24 (Physical Pin 18) and GPIO23 (Physical Pin 16) respectively.

Ultrasonic Sensor Module:

An **Obstacle Avoider Robot** is an Automated Robot and it doesn’t need to be controlled using any remote. These types of automated robots have some ‘sixth sense’ sensors like obstacle detectors, sound detector, heat detector or metal detectors. Here we have done **Obstacle Detection using Ultrasound Signals**. For this purpose, we have used Ultrasonic Sensor Module.

Ultrasonic Sensors are commonly used to detect objects and determine the distance of the obstacle from the sensor. This is a great tool to measure the distance without any physical contact, like as Water Level Measurement in tank, distance measurement , Obstacle avoider robot etc. So here, we have detected the object and measured the distance by using Ultrasonic Sensor and Raspberry Pi.

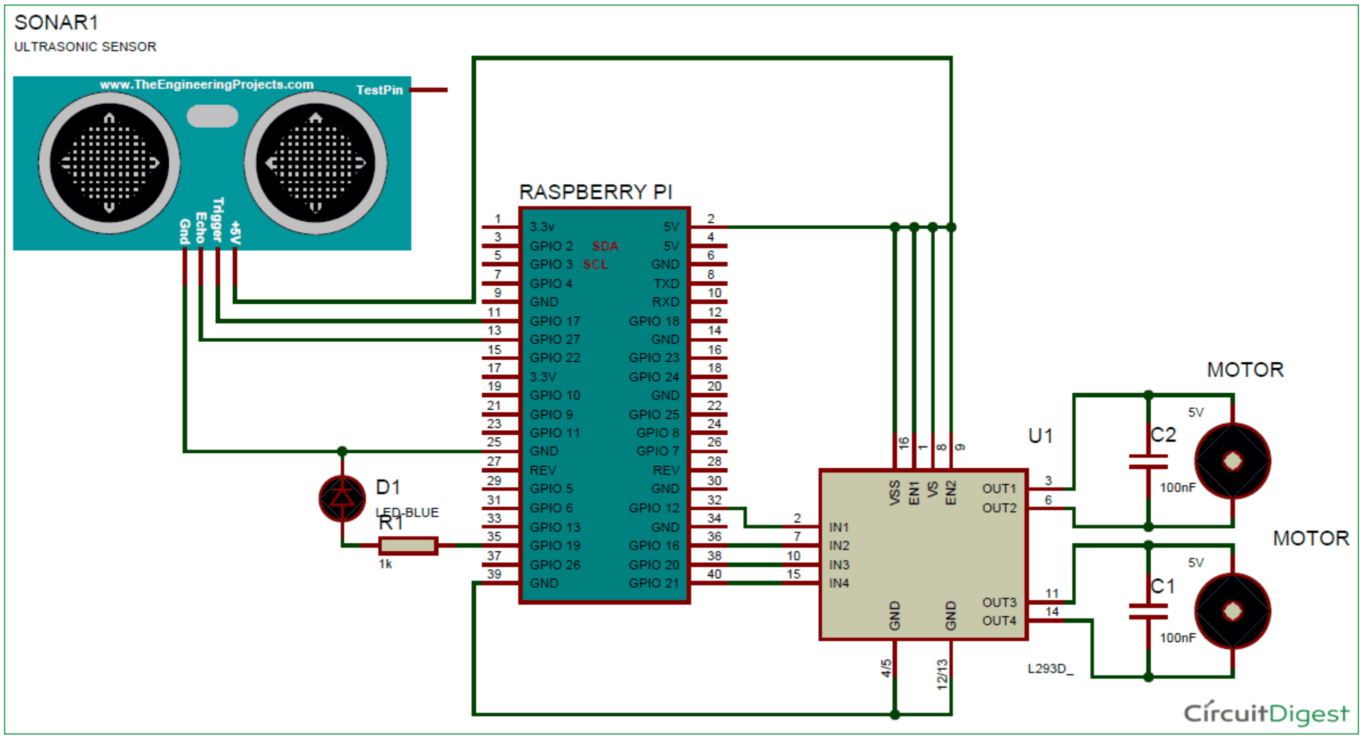
Ultrasonic sensor HC-SR04 is used to measure distance in the range of 2cm-400cm with an accuracy of 3mm. The sensor module consists of an ultrasonic transmitter, receiver, and the control circuit. Ultrasonic Sensor consists of two circular eyes out of which one is used to transmit the ultrasonic wave and the other to receive it.

We can calculate the distance of the object based on the time taken by ultrasonic wave to return back to the sensor. Since the time and speed of sound is known we can calculate the distance by the following formulae.

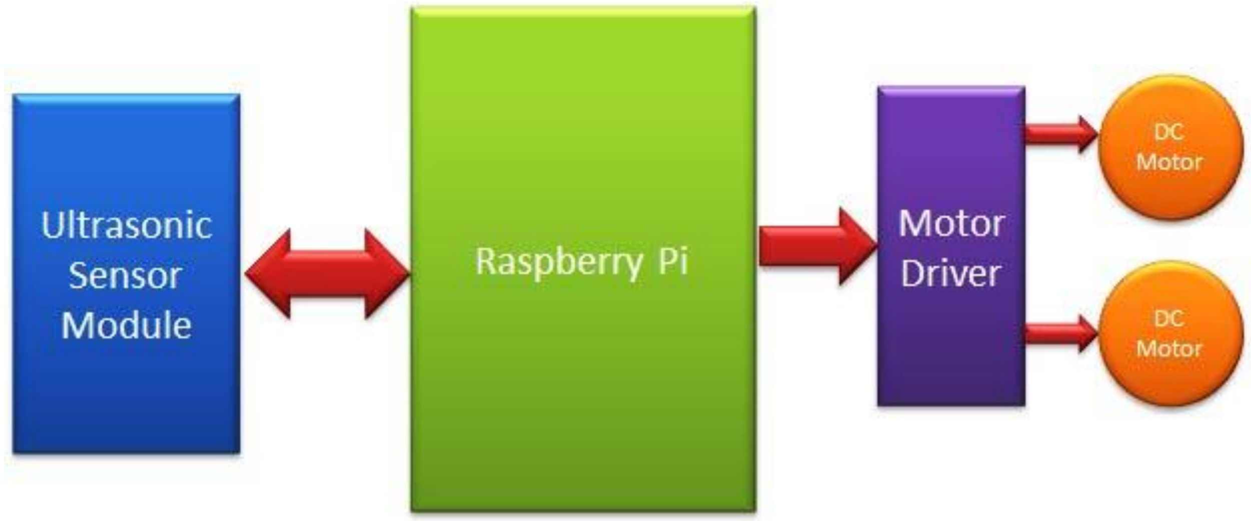
- Distance= (Time x Speed of Sound in Air (343 m/s))/2.

The value is divided by two since the wave travels forward and backward covering the same distance. Thus, the time to reach obstacle is just half the total time taken.

So, we have calculated the distance (in centimetre) from the obstacle like below:



Working-



We have selected 50cm distance for taking any decision by Raspberry Pi. Now whenever Raspberry Pi gets less than the 15cm distance from any object then Raspberry Pi stops the robot and moves it back and then turns it left or right. Now before moving it forward again, Raspberry Pi again checks whether any obstacle is present within the range of 15 cm distance, if yes then again repeats the previous process, else move the robot forward until it will detect any obstacle or object again.

Programming Explanation:

We are using **Python language** here for the Program. Before coding, user needs to configure Raspberry Pi.

The programming part of this project plays a very important role to perform all the operations. First of all, we include required libraries, initialize variables and define pins for ultrasonic sensor, motor and components.

Python Program for Controlling a DC Motor with Raspberry Pi

```
import RPi.GPIO as GPIO

import time



#Import time library

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)



TRIG = 17

ECHO = 27

... .....
```

```
.... .....
```

(After it, we have created some functions *def forward(), def back(), def left(), def right()* to move robot in forward, backward, left or right direction respectively and *def stop()* to stop the robot, check the functions in Code given below.

Then, in the main program, we have initiated Ultrasonic Sensor and read time between transmission and reception of the signal and calculated the distance. Here we have repeated this process for 5 times for better accuracy. We have already explained the process of calculating the distance using Ultrasonic sensor.

)

Complete Code for Obstacle Avoiding Rover-

```
import RPi.GPIO as GPIO                    #Import GPIO library
import time

#Import time library
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)                     # programming the GPIO by BCM pin numbers

TRIG = 17
ECHO = 27
led = 22

m11=16
m12=12
m21=21
m22=20

GPIO.setup(TRIG,GPIO.OUT)                  # initialize GPIO Pin as outputs
GPIO.setup(ECHO,GPIO.IN)                   # initialize GPIO Pin as input
GPIO.setup(led,GPIO.OUT)

GPIO.setup(m11,GPIO.OUT)
GPIO.setup(m12,GPIO.OUT)
GPIO.setup(m21,GPIO.OUT)
GPIO.setup(m22,GPIO.OUT)

GPIO.output(led, 1)

time.sleep(5)

def stop():
   print "stop"
   GPIO.output(m11, 0)
   GPIO.output(m12, 0)
   GPIO.output(m21, 0)
   GPIO.output(m22, 0)

def forward():
   GPIO.output(m11, 1)
   GPIO.output(m12, 0)
   GPIO.output(m21, 1)
   GPIO.output(m22, 0)
   print "Forward"

def back():
   GPIO.output(m11, 0)
```

```
    GPIO.output(m12, 1)
    GPIO.output(m21, 0)
    GPIO.output(m22, 1)
    print "back"

def left():
    GPIO.output(m11, 0)
    GPIO.output(m12, 0)
    GPIO.output(m21, 1)
    GPIO.output(m22, 0)
    print "left"

def right():
    GPIO.output(m11, 1)
    GPIO.output(m12, 0)
    GPIO.output(m21, 0)
    GPIO.output(m22, 0)
    print "right"

stop()
count=0
while True:
 i=0
 avgDistance=0
 for i in range(5):
  GPIO.output(TRIG, False)                 #Set TRIG as LOW
  time.sleep(0.1)                          #Delay

  GPIO.output(TRIG, True)                  #Set TRIG as HIGH
  time.sleep(0.00001)                      #Delay of 0.00001 seconds
  GPIO.output(TRIG, False)                 #Set TRIG as LOW

  while GPIO.input(ECHO)==0:               #Check whether the ECHO is LOW
       GPIO.output(led, False)
  pulse_start = time.time()

  while GPIO.input(ECHO)==1:               #Check whether the ECHO is HIGH
       GPIO.output(led, False)
  pulse_end = time.time()
  pulse_duration = pulse_end - pulse_start #time to get back the pulse to sensor

  distance = pulse_duration * 17150        #Multiply pulse duration by 17150 (34300/2) to get distance
  distance = round(distance,2)             #Round to two decimal points
  avgDistance=avgDistance+distance

 avgDistance=avgDistance/5
 print avgDistance
 flag=0
 if avgDistance < 15:      #Check whether the distance is within 15 cm range
    count=count+1
    stop()
    time.sleep(1)
    back()
    time.sleep(1.5)
    if (count%3 ==1) & (flag==0):
     right()
     flag=1
    else:
     left()
     flag=0
    time.sleep(1.5)
    stop()
    time.sleep(1)
 else:
```

```
forward()
flag=0
```

PYTHON PROGRAM FOR ROVER WITH CONTROLLER

```
import time
import RPi.GPIO as gpio
gpio.setmode(gpio.BCM)

TRIG=4
ECHO=18

def init():
    gpio.setup(17, gpio.OUT)
    gpio.setup(22, gpio.OUT)
    gpio.setup(23, gpio.OUT)
    gpio.setup(24, gpio.OUT)

print("Distanve Measurement...")
gpio.setup(TRIG,gpio.OUT)
gpio.setup(ECHO,gpio.IN)
gpio.output(TRIG, False)

def measure():
    time.sleep(0.333)
    gpio.output(TRIG, True)
    time.sleep(0.00001)
    gpio.output(TRIG, False)
    while gpio.input(ECHO)==0:
      pulse_start = time.time()

    while gpio.input(ECHO)==1:
      pulse_end = time.time()
    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance, 2)
    return distance

def forward():
    init()
    gpio.output(17, False)
    gpio.output(22, True)
    gpio.output(23, False)
    gpio.output(24, True)

def right():
    init()
    gpio.output(17, False)
    gpio.output(22, True)
    gpio.output(23, False)
    gpio.output(24, False)

def left():
    init()
    gpio.output(17, False)
    gpio.output(22, False)
    gpio.output(23, False)
    gpio.output(24, True)

def stop():
    init()
    gpio.output(17, False)
    gpio.output(22, False)
    gpio.output(23, False)
    gpio.output(24, False)


try:
    while True:
        distance=measure()
        print("Distance : %.1f cm" % distance)
        time.sleep(0.3)

        if distance >=15:
            forward()
        else:
            stop()
            t=input("\nEnter Direction:")
            if t=='left':
                left()
            elif t=='right':
                right()
            elif t=='finish':
                break;

            else:
                print("Enter correct keyword")
    gpio.cleanup()

except KeyboardInterrupt:
    gpio.cleanup()
```