F 241 MPI - Mid-Semester Report

1. Team Name: Karasuno
2. Team Lead: Abhinava M
3. Members: Abhinava M, Mohit G

S.No:	Name	Email ID	Mobile Number
1	Abhinava Maddha	f20180844@hyderabad.bits-pilani.ac.in	9618903255
2	Mohit Guddanti	f20180321@hyderabad.bits-pilani.ac.in	6303627234
3			
4			
5			
6			

4. **Project Title:** Navigation app for the visually impaired

   Navigation tool in backpack for the blind(original)

5. **Description of Work Carried Out Till Date : (10)**

For the original project we had completed learning about python and raspberry pi and decided to use a webcam attached to a bag strap with all the hardware interfacing inside the bag and navigation relayed to the user through earphones. We started interfacing the rasp-pi board with the camera. We did not buy all the components hence we decided to do a software project on the same idea.

For the new project we have started learning android app development online as we are not cs students with prior knowledge on this. So far, we have learnt a bit of Java and Android studio.

6. Tools and Techniques used (if any) : (5)

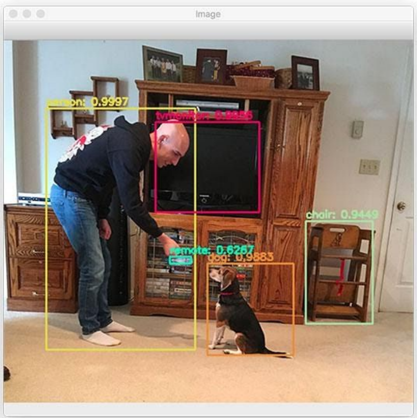
We are planning on using the camera app in-built in the phone, YOLO algorithm with Opencv Python and google-maps to help the user detect an obstacle from a safe distance and walk accordingly.

7. Details of Algorithms / Designs / Simulations (if any): (15)

The app would receive information from the maps for general directions and the visual information from the camera app would be transferred to a prewritten deep-learning algorithm like YOLO(real time object detection) to detect objects in front and the app will be programmed in such a way that it receives all this information and intimates the user when an obstacle is ahead of him 8 ft and 2 ft away and what kind of an obstacle, which he will receive through a set of earphones connected to the phone. The user will have to attach the phone to his bag strap or shirt pocket in order to use the camera.

8. Important Results Obtained: (10)

We have tried using YOLO with Opencv Python to detect a few objects like chairs, humans and cars. We still have to train it to detect more objects.



-object detection in an image

Code for Object detection in videos (real time detection).

YOLO Object Detection with OpenCV

```python
# import the necessary packages
import numpy as np
import argparse
import imutils
import time
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", required=True,
   help="path to input video")
ap.add_argument("-o", "--output", required=True,
   help="path to output video")
ap.add_argument("-y", "--yolo", required=True,
   help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
   help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
   help="threshold when applyong non-maxima suppression")
args = vars(ap.parse_args())
```

```python
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
   dtype="uint8")

# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])

# load our YOLO object detector trained on COCO dataset (80 classes)
# and determine only the *output* layer names that we need from YOLO
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
```

```python
# initialize the video stream, pointer to output video file, and
# frame dimensions
vs = cv2.VideoCapture(args["input"])
writer = None
(W, H) = (None, None)

# try to determine the total number of frames in the video file
try:
    prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() \
        else cv2.CAP_PROP_FRAME_COUNT
    total = int(vs.get(prop))
    print("[INFO] {} total frames in video".format(total))

# an error occurred while trying to determine the total
# number of frames in the video file
except:
    print("[INFO] could not determine # of frames in video")
    print("[INFO] no approx. completion time can be provided")
    total = -1
```

```python
# loop over frames from the video file stream
while True:
    # read the next frame from the file
    (grabbed, frame) = vs.read()

    # if the frame was not grabbed, then we have reached the end
    # of the stream
    if not grabbed:
        break

    # if the frame dimensions are empty, grab them
    if W is None or H is None:
        (H, W) = frame.shape[:2]
```

```python
    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes
    # and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
    net.setInput(blob)
    start = time.time()
    layerOutputs = net.forward(ln)
    end = time.time()

    # initialize our lists of detected bounding boxes, confidences,
    # and class IDs, respectively
    boxes = []
    confidences = []
    classIDs = []
```

```
    # loop over each of the layer outputs
    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
            # extract the class ID and confidence (i.e., probability)
            # of the current object detection
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            # filter out weak predictions by ensuring the detected
            # probability is greater than the minimum probability
            if confidence > args["confidence"]:
                # scale the bounding box coordinates back relative to
                # the size of the image, keeping in mind that YOLO
                # actually returns the center (x, y)-coordinates of
                # the bounding box followed by the boxes' width and
                # height
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                # use the center (x, y)-coordinates to derive the top
                # and and left corner of the bounding box
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))


                # update our list of bounding box coordinates,
                # confidences, and class IDs
                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)
```

```python
        # apply non-maxima suppression to suppress weak, overlapping
        # bounding boxes
        idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
            args["threshold"])

        # ensure at least one detection exists
        if len(idxs) > 0:
            # loop over the indexes we are keeping
            for i in idxs.flatten():
                # extract the bounding box coordinates
                (x, y) = (boxes[i][0], boxes[i][1])
                (w, h) = (boxes[i][2], boxes[i][3])

                # draw a bounding box rectangle and label on the frame
                color = [int(c) for c in COLORS[classIDs[i]]]
                cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
                text = "{}: {:.4f}".format(LABELS[classIDs[i]],
                    confidences[i])
                cv2.putText(frame, text, (x, y - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

```python
        # check if the video writer is None
        if writer is None:
            # initialize our video writer
            fourcc = cv2.VideoWriter_fourcc(*"MJPG")
            writer = cv2.VideoWriter(args["output"], fourcc, 30,
                (frame.shape[1], frame.shape[0]), True)

            # some information on processing single frame
            if total > 0:
                elap = (end - start)
                print("[INFO] single frame took {:.4f} seconds".format(elap))
                print("[INFO] estimated total time to finish: {:.4f}".format(
                    elap * total))

        # write the output frame to disk
        writer.write(frame)

    # release the file pointers
    print("[INFO] cleaning up...")
    writer.release()
    vs.release()
```

Command exectution:

YOLO Object Detection with OpenCV

```
$ python yolo_video.py --input videos/overpass.mp4 \
    --output output/overpass.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 812 total frames in video
[INFO] single frame took 0.3534 seconds
[INFO] estimated total time to finish: 286.9583
[INFO] cleaning up...
```

YOLO video
detection.mp4

(**Signature)**