

Abstract

This project is a software that takes frames from stereo camera as an input and tells what object is in the frame and at what distance. This can basically be thought of as a tool that will help visually impaired people when made into a wearable device, moreover this can also be used to make and automate some alarming systems that needs to know the distance of particular objects from a point.

This project requires simultaneous frames from a stereo camera with known parameters. The frames are then processed to find out a central object in them, and further it uses SIFT feature matching to know the offset of similar points in both the frames and using that it calculates how far that object is from the camera. And finally, it announces the object and it's distance for a given pair of stereo frames.

Keywords: Stereo Camera, Object Detection, Feature Matching, Depth Measurement, Distance Measurement, SIFT, YOLO.

Contents

Acknowledgements and Work Disribution	ii
Introduction	iii
1 Object Detection	1
2 Feature Detection and Matching	4
3 Distance Estimation	6
4 Conclusion, Future Work and References	9

Acknowledgments

I acknowledge the support of Dr. Yamuna Prasad sir in making the project. It was due to his quality guidance and firm belief in me that I could complete this project. I also acknowledge Dr. Vinit Jakhethiya whose imparted knowledge on computer vision made my life easier while completing this project. I also acknowledge all the professors, because it is their contribution that made me capable enough, that today I make complex projects like this. I also acknowledge Stackoverflow and my friend Vikas Gola, as they helped me recover from some dead-ends where nothing was working.

Work Distribution

The entire work was done by me. I do not have any team-mate for this project.

Introduction

This project is a simple surrounding familiarisation system that detects objects in front of it and speaks out the type and the distance of that object from the user/device. The motivation behind making this was a compassion to give back to the society as a computer science engineer. The idea came to my mind by a need: helping visually impaired - blind or semi-blind people - to familiarise with their surroundings. Also I liked the field of computer vision and AI so I thought it would be a great idea if I am able to make such a useful system out of it.

So, briefly this whole task of detecting object and finding their distance, consists of various sub-modules working together. The first one would obviously be object detection from image frames. For that, I have used YOLOv3 model, as it is quite fast as compared to others and gives accurate results. I used ImageAI to load the model and do the work of detection. The next thing is computing distance of the detected object. This job requires the knowledge of an image from a different angle to see how much the pixel value of the same point in the object changes. Thus a need arises for a stereo camera. A stereo camera is a pair of camera which are calibrated and are fixed in every degree of freedom with respect to each other. They are just horizontally (X-axis) separated among them, but have the same vertical (Y-axis) height and lies on the same Z-plane. Also they are at a same angle with each other, as in roll, pitch and yaw. The distance between them and their focus, both are known. Together they create camera parameter for the stereo camera pair.

To find the common points in both the images, I have used the SIFT (Scalar Invariant Feature Transform) algorithm, as it matches features most accurately as compared to ORB and Harris Corner Detector. As the camera are at a same vertical height, the matched features only vary in horizontal pixels, and that variation is used to calculate the distance of that object using the camera parameters. After that the verdict is announced using a text-to-speech engine. So in this way, the person will know that if there is any object in front of him and if so, at what distance it is.

I was also planning to produce the whole system into a wearable device which would work using Raspberry Pi module and a well engineered set of cameras to replicate a stereo camera that a person could wear like sunglasses and earphones to communicate the verdict. But due to covid19 situations, I wasn't able to order any hardware for that. Now it is only in software format in my laptop.

Chapter 1

Object Detection

The first part of the whole system is the Object Detection module. The job of this module is to look for any object in the left frame. The input to the system is a pair of stereo frames clicked on the same time. I am doing object detection in the left frame, because in almost all the cases, the same objects will be there in both the frames, as the horizontal distance between the cameras will not be much. So detecting objects in one frame will do the job.

```
def objdtn_setup(): # Setup the models for object detection
    global execution_path
    global detector
    execution_path = os.getcwd()
    detector = ObjectDetection()
    '''
    detector.setModelTypeAsYOLOv3()
    detector.setModelPath( os.path.join(execution_path ,
                                         "yolo.h5"))
    '''
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath( os.path.join(execution_path ,
                                         "resnet50_coco_best_v2.0.1.h5"))
    '''
    detector.setModelTypeAsTinyYOLOv3()
    detector.setModelPath( os.path.join(execution_path ,
                                         "yolo-tiny.h5"))
    '''
    detector.loadModel()
```

For object detection, I am using ImageAI, a library that has the support of various models like: YOLOv3, TinyYOLOv3 and Resnet, and these are the most popular, fast and efficient models for this job. Actually YOLOv3 is one of the fastest models for object detection in the accuracy range that it provides. TinyYOLOv3 is even faster than YOLOv3 but it lacks accuracy and Resnet has better accuracy than YOLOv3 but is far more slower than that. So the natural choice for this project is YOLOv3, however the other two can also be used as you can see in the

code. ImageAI works on Keras which in turn works on TensorFlow. So ImageAI loads the model and does the detection and returns an array of detected objects with the class, the probability and the coordinates of the bounding box.



(a) Bus detected

Now here I faced an issue, there can be more than one objects in the same frame. So to deal with that, I decided to choose the object with biggest area in the image because obviously it would be the closest or the largest object of all. So this software will tell the distance of the largest object in the frame. Another issue can be that there are no detectable objects in the given frame, so to handle that case I am returning a boolean value that whether any object has been detected or not. And finally it also returns the dictionary containing the name, probability and the coordinates of the bounding box of the detected object.

Code on next page ...

```
def objdtn(frame_name): # detect objects in the left frame
    global execution_path
    global detector
    detections = detector.detectObjectsFromImage(input_image=
        os.path.join(execution_path, frame_name),
        minimum_percentage_probability=70)
    if(detections!=[]):
        prime_object = {}
        max_area = 0
        for eachObject in detections:
            arr_cd = eachObject["box_points"]
            area = (arr_cd[2]-arr_cd[0]) *
                (arr_cd[3]-arr_cd[1])
            if(area>=max_area):
                max_area = area
                prime_object = eachObject
        return True, prime_object
    else:
        return False, {}
```

Here we are done with detecting the object and passed on the name and the coordinates of its bounding box to further processing which we will see in the next chapter.

Chapter 2

Feature Detection and Matching

After Object Detection comes the second part: Feature Matching. Actually this is the part which came to my mind the latest. I was not thinking of this way of approaching the problem. The way I was thinking of finding the depth from the stereo images was by making a grayscale depth-map of the scene. And using the pixel value of the of the depth-map (that ranges from 0 to 255) to find the actual depth or distance. I was planning to map the range 255-0 to zero to infinity, but that posed a whole lot of problems. First of all I need to make a depth map and for that I searched for a lot of things, but none of them worked. There is a functionality in *opencv* called disparity, it was the only thing that produced the closest thing to a depth-map. But it only worked in a few images. And produced noisy values in rest all. And even if I would somehow have managed to make depth-maps, mapping actual distance to the brightness of the map would pose a whole another difficulty. I then had to do regression and reach on a function that maps the brightness to actual distance.

But one day while thinking an idea came to my mind. All I had to do for distance calculation, was to find the coordinates of corresponding points in both the images. So I remembered that I have studied something like that before, and hence came to me the idea of using SIFT (Scalar Invariant Feature Transform) to match the features. But while using it I came to know that as it is a patented algorithm, the newer version of *opencv* does not allow one to use it. Then after a lot of searching I had to downgrade the version of my *opencv*. And finally it worked. But after that I came to know that SIFT's patent has expired.

SIFT is a feature detection and matching algorithm. It detects and describes local features of an image in vector form. For doing SIFT feature matching, first the features of both the images are extracted and stored, and then each feature of the first image is matched with each feature of the second image based on their euclidean distance of their feature vectors. We can specify how close the match should be to be called a match.

SIFT as the name specifies, is scalar invariant, i.e., even if the sizes of the images are different the features will still be matched. Also it is somewhat rotation invariant for minor rotations. Moreover, SIFT is best fit to use in my application, because the images I am giving SIFT as an input, are taken from a pair of Stereo Cameras, and the cameras are perfectly alligned, so there won't be much of a rotation of one image with respect to the other. The good thing is that there won't be much of a scaling difference either, only there will be translation of objects in both the frames. And for that SIFT is the best.

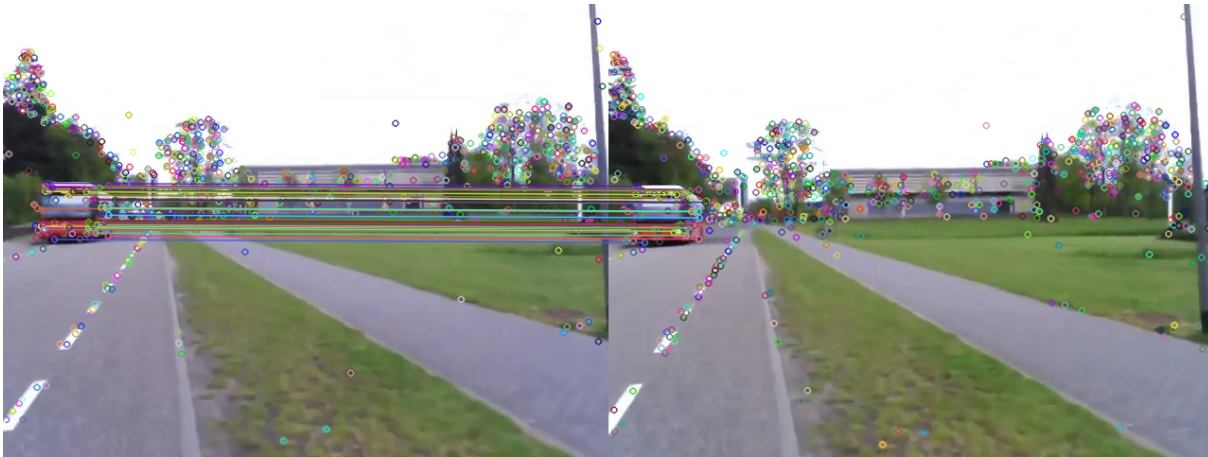
```

def feature_detector(frame1, frame2, box_pts): # Match features
    left_im = cv2.imread(frame1)
    right_im = cv2.imread(frame2)
    sift = cv2.xfeatures2d.SIFT_create()
    kp_1, desc_1 = sift.detectAndCompute(left_im, None)
    kp_2, desc_2 = sift.detectAndCompute(right_im, None)
    index_params = dict(algorithm=0, trees=5)
    search_params = dict()
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(desc_1, desc_2, k=2)
    good_points = []
    for m, n in matches:
        if m.distance < 0.7*n.distance:
            if(is_it_in(box_pts, kp_1[m.queryIdx].pt[0],
                        kp_1[m.queryIdx].pt[1])):
                good_points.append(m)
            ret = [(kp_1[m.queryIdx].pt[0] -
                    kp_2[m.trainIdx].pt[0])
                  for m in good_points]

    return ret

```

Now for the purpose of finding distance, we only need the coordinates of matched features of the detected object. We don't need the rest other features. So to limit that I am taking only those points that lie in the bounding box of our detected object.

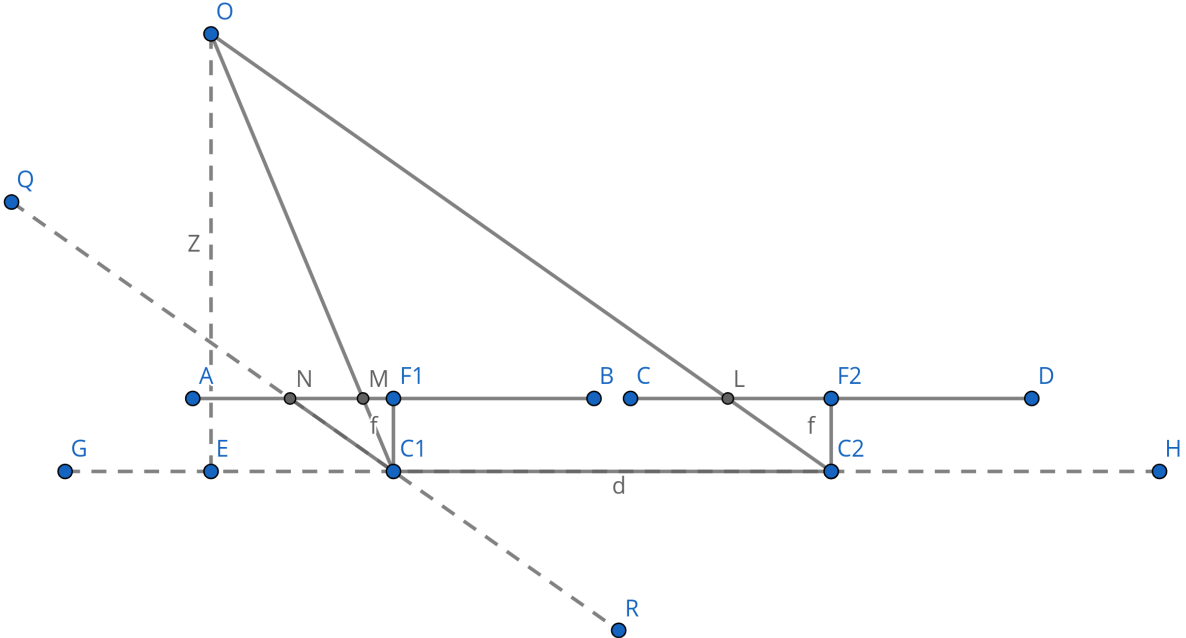


(a) Features Matched only from the bus

Chapter 3

Distance Estimation

This is the final module of the software. In this module, the input is the coordinates of the matched features of the object in bounding box. Here, one thing should be noted that, the y-coordinate of the features are almost same in both the images as the vertical height of both the cameras are the same. So only thing that differs is the x-coordinates of the matched feature. Also, it is obvious that, of the same feature, the x-coordinate in the left frame will be greater (or equal) than the x-coordinate in the right frame. So now we have to calculate the distance of the object based on the difference of the x-coordinates of the same feature point in both the images.



(a) The geometry behind this

To illustrate that I have included a geometrical diagram. First things first: $C1$ and $C2$ are the cameras. AB and CD are the image planes of the respective cameras. Also, AB and CD are the part of the same line AD . $F1$ and $F2$ are perpendiculars from $C1$ and $C2$ to their image planes. And $C1F1$ and $C2F2$ are the focus of the camera which is equal to f (equal in both). Both $C1$ and $C2$ lie on the line GH , which is parallel to AD . And the distance between the cameras or

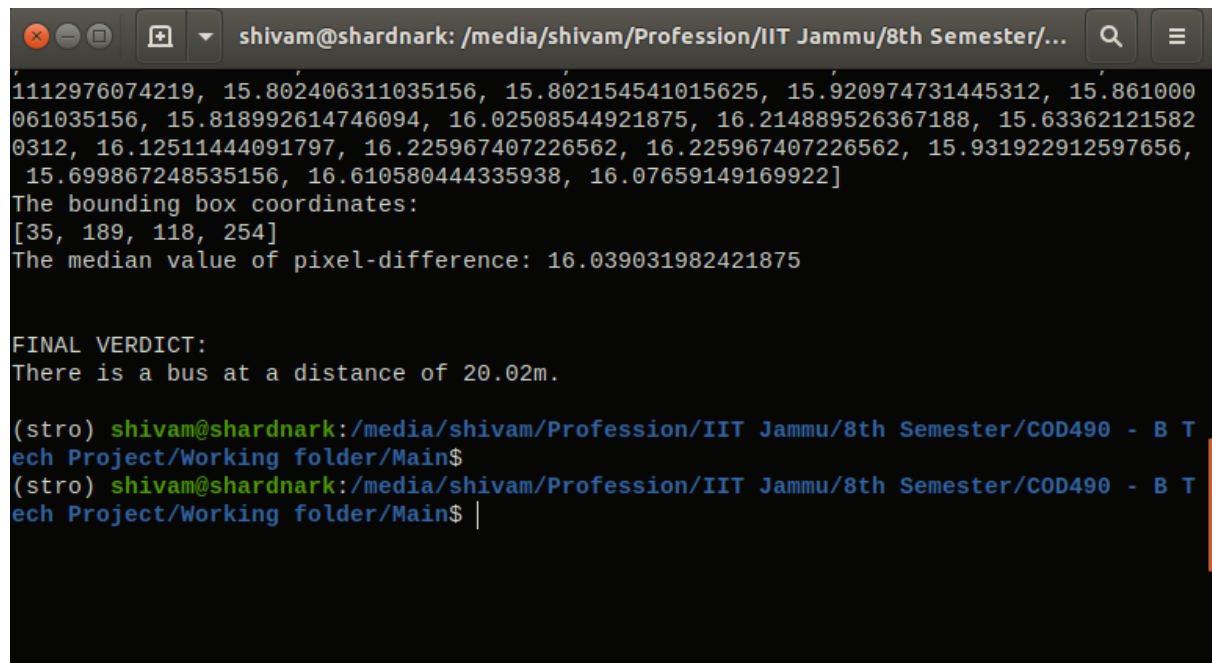
C1C2 is d . O is the object (or the feature point, in this case) in real life. Its image on left and right cameras are M and L respectively. OE is perpendicular to GH, and is the distance of the object that we need to find, let's say its Z . QR is parallel to C2O and passes through C1 and cuts AB at N.

It should be clear that triangle C1MN and triangle OC1C2 are similar triangles, because $\angle OC1C2$ is equal to $\angle C1MN$ (transverse angles) and $\angle OC2C1$ and $\angle C1NM$ are equal (opposite angle of a parallelogram). Now, in left image, the image of object O is at M, so the value of its x-coordinate is AM, let's denote it as x_1 similarly let's denote CL, the value of x-coordinate of image of O in the right image as x_2 . Hence, due to similarity of the triangles, $f/Z = (x_1 - x_2)/d$, which gives us the distance, $Z = fd/(x_1 - x_2)$. And in this, fd is the parameter of the stereo camera pair which can be measured or can be calculated by a known Z (like image of a scale, or an object at a known distance).

```
def main_code(left_frame, right_frame, fd, enj): #Combining all
    truth, prime_object = objdtn(left_frame)
    if(truth==False):
        print("Nothing in frame")
    else:
        diff_arr = feature_detector(left_frame, right_frame,
                                    prime_object["box_points"])
        if(diff_arr==[]):
            print("No significant features detected")
        else:
            print('The array of pixel-difference of
                  x coordinates: ')
            print(diff_arr)
            print('The bounding box coordinates: ')
            print(prime_object["box_points"])
            med = statistics.median(diff_arr)
            print('The median value of pixel
                  difference: ' + str(med))
            distance = fd/med
            print("\n\nFINAL VERDICT:")
            print("There is a " + str(prime_object["name"]))
                + " at a distance of " +
                "{:.2f}".format(round(distance, 2)) +
                "m. \n")
            enj.setProperty('rate', 150)
            enj.say("\nThere is a " + str(prime_object["name"]))
                + " at a distance of " +
                "{:.2f}".format(round(distance, 2)) +
                " meters. \n")
            enj.runAndWait()
            enj.stop()
```

All these theory has been incorporated into the code shown. Now there is an interesting issue, that there are more than one feature points for the same object, so which one should be used to find the distance. Can the mean of their difference of x-coordinates be taken? Well taking mean may incorporate some errors in some frames, because mean doesn't tend to give central value when the values include both small and large numbers. Like, the mean of 1, 2, 3, 4 and 1000 is far away from the core values 1, 2, 3 and 4. So here outliers can distort the result vastly. Hence median would be a better measure for an application like this. So I have taken the median of the differences of the x-coordinates of the features, and then calculated the distance by, $Z = fd/(x_1 - x_2)$. Here, for the specific case, I have estimated the fd by looking into the image, as again I did not have access to any stereo camera, all these images that I got, I got from the internet. However, this estimation works well.

So here is the final output, also shown in the screenshot: **There is a bus at a distance of 20.01m**. One more feature the software has is, it speaks-up this sentence using a text-to-speech engine, hence would be able to communicate to the visually impaired people.



```

shivam@shardnark: /media/shivam/Profession/IIT Jammu/8th Semester/...
1112976074219, 15.802406311035156, 15.802154541015625, 15.920974731445312, 15.861000
061035156, 15.818992614746094, 16.02508544921875, 16.214889526367188, 15.63362121582
0312, 16.12511444091797, 16.225967407226562, 16.225967407226562, 15.931922912597656,
15.699867248535156, 16.610580444335938, 16.07659149169922]
The bounding box coordinates:
[35, 189, 118, 254]
The median value of pixel-difference: 16.039031982421875

FINAL VERDICT:
There is a bus at a distance of 20.02m.

(stro) shivam@shardnark:/media/shivam/Profession/IIT Jammu/8th Semester/COD490 - B T
ech Project/Working folder/Main$
(stro) shivam@shardnark:/media/shivam/Profession/IIT Jammu/8th Semester/COD490 - B T
ech Project/Working folder/Main$ |

```

(a) The final output

Chapter 4

Conclusion, Future Work and References

Conclusion

So in conclusion, the project is a nice combination of ML and Computer Vision. It is great to see that how effective feature matching has proved to be to find the distance. If produced into a wearable device, this software would be a real game-changer and will help the wearer a lot in being familiarised with the environment. And what I learnt by doing this project is how two diverse fields can be combined to create a beautiful product. Also I got to know more about SIFT and Object Detection and most importantly the theory of calculating distance from stereo images.

Future Work

There is a lot of scope of work in this project in future. The most primary one is making the hardware-wearable device having this feature. It would be like a sunglass with a camera in place of each glass-frame, and an earphone will be there to communicate. All of these can be controlled by a Raspberry Pi. And in the software itself, there can be some optimisations to run this on real-time. GPU support can be included, as the newer Raspberry Pis are coming with GPUs.

References

- [1] <https://www.youtube.com/watch?v=75YtldrfxBU>
- [2] <https://www.youtube.com/watch?v=KOvHsn66sZA>
- [3] <https://github.com/OlafenwaMoses/ImageAI/blob/master/imageai/Detection/README.md>
- [4] <https://github.com/OlafenwaMoses/ImageAI>
- [5] <https://medium.com/@pwc.emtech.eu/object-detection-with-imageai-106d584984e9>
- [6] <https://lmb.informatik.uni-freiburg.de/resources/datasets/StereoEgomotion/>
- [7] <https://www.youtube.com/watch?v=sW4CVI51jDY>