Aerial Robotics Kharagpur Documentation

Pradyumn Mahajan 21AE30017

***Abstract*— This document briefly summarises my efforts to complete the tasks given to me by the ARK team as a part of the Task Round of their selection procedure for the software team. Although I have only been able to do tasks 2.2 and 3.1 part 1 in the given time limit, I have learnt a lot during the course of this task like learning python, OpenCV and ROS from scratch. During the course of this Task I learnt about the application of python and its libraries in various fields such as stereo images, ArUco marker detection and pose estimation and various path finding algorithms such as Breadth First Search, Depth First Search, Djikstra and A Star algorithms. I am very grateful to the ARK team to have been given the opportunity to perform this task.**

I have learnt about numerous things such as reading 2 stereo images and then using them to generate a depth map. This is used by a drone to detect obstacle distances which it needs to avoid while moving. I also learnt about ID detection and pose estimation using ArUco markers which are synthetic square markers with a binary matrix inside which determines its ID. Pose Estimation is based on finding correspondences between points in the real environment and their 2d image projection and has a variety of applications in robot navigation, augmented reality etc. I also learnt about various path planning algorithms which are used by drones to obtain the best and most efficient path that it would travel.

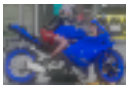
I. INTRODUCTION - TASK 2.2

In this task, we were provided with a pair of stereo images, an image of the obstacle (bike) and the projection matrices of the cameras and were asked to generate a depth map, find the obstacle and calculate the distance of the obstacle from the the camera.

II. PROBLEM STATEMENT

The stereo images provided to us were :

The image of the bike is:



And the projection matrices of the cameras are
p-left = [[640.0, 0.0, 640.0, 2176.0], [0.0, 480.0, 480.0, 552.0], [0.0, 0.0, 1.0, 1.4]]
p-right = [[640.0, 0.0, 640.0, 2176.0], [0.0, 480.0, 480.0, 792.0], [0.0, 0.0, 1.0, 1.4]]

We were supposed to use this information and generate the depth map, find the obstacle and calculate the distance of the obstacle from the camera.

III. FINAL APPROACH

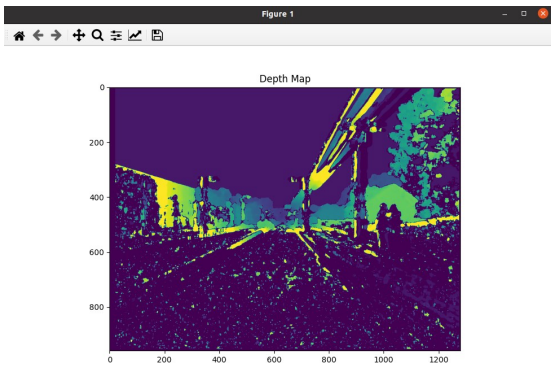
I first read all 3 images in python using the OpenCV function cv2.imread() in gray scale.

```
5  limg=cv2.imread("left.png",cv2.IMREAD_GRAYSCALE)
6  rimg=cv2.imread("right.png",cv2.IMREAD_GRAYSCALE)
7  obs=cv2.imread("bike.png",cv2.IMREAD_GRAYSCALE)
```

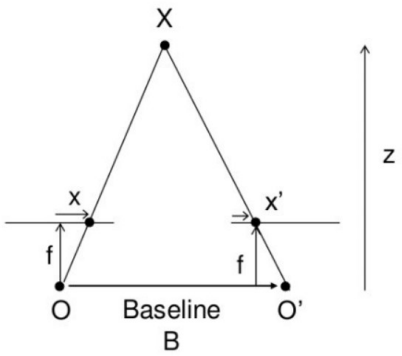
Then I created a depth map using the OpenCV function cv2.StereoBM_create() .I passed the number of disparities and the block size as parameters to the function. The number of disparities is the number of pixels that the window will slide over. It needs to be a multiple of 16.

```
22  stereo = cv2.StereoBM_create(numDisparities=16, blockSize=13)
23  depthmap = stereo.compute(limg,rimg)
```

The Generated Depth Map looks like:



The disparity is the apparent pixel difference between 2 stereo images.It is calculated by using similarity of triangles.



$$disparity = x - x' = \frac{Bf}{Z}$$

Here, x and x' are the distance between points in the image plane corresponding to the scene point 3D and their camera center, B is the distance between two cameras and f is the focal length of the camera.

In order to find the disparity, I used template matching to find the coordinates of the center of the bike in the left and right camera images and found the value of the disparity in the depth map at this point.

```
threshold = 0.999
res = cv2.matchTemplate(limg,obs,cv2.TM_CCORR_NORMED)
loc = np.where(res >= threshold)

if(len(loc[0]) != 0 and len(loc[1]) != 0):
    for pt in zip(*loc[::-1]):
        cv2.rectangle(limg, pt, (pt[0] + obs.shape[1], pt[1] + obs.shape[0]), (0,255,255), 3)
        cv2.imshow("temp",limg)
        xl=pt[1] + obs.shape[0]//2
        yl=pt[0] + obs.shape[1]//2

res = cv2.matchTemplate(rimg,obsr,cv2.TM_CCORR_NORMED)
loc = np.where(res >= threshold)

if(len(loc[0]) != 0 and len(loc[1]) != 0):
    for pt in zip(*loc[::-1]):
        cv2.rectangle(rimg, pt, (pt[0] + obsr.shape[1], pt[1] + obsr.shape[0]), (0,255,255), 3)
        cv2.imshow("temp",limg)
        xr=pt[1] + obsr.shape[0]//2
        yr=pt[0] + obsr.shape[1]//2
```

The matched template looks like :

Now, I found the values of B and f from the given projection matrices which were then used to calculate the depth of the obstacle.

The projection matrix of a camera can be expressed as the product of an intrinsic and extrinsic matrix. The intrinsic matrix is a 3*3 upper triangular matrix and the extrinsic matrix is a combination of a 3*3 rotation matrix and a 3*1 translation matrix.

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

This is the code for the distance calculation of the obstacle.

```
fy_multi_d=(p_right[1][3]-p_left[1][3])
dist=fy_multi_d/(xr-xl)
print("Distance = ",dist)
```

IV. Results and Observation

The depth map was generated and the distance of the obstacle was found to be 21.818m.

V. Future Work

This algorithm can be applied to a stereo camera attached to a drone which can be used for obstacle avoidance.

VI. Conclusion

In this task, we were provided with the stereo images obtained from 2 cameras along with the projection matrices of the 2 cameras. We were supposed to find the distance of a bike which was the obstacle from the camera by generating a depth map from the 2 images. I generated the depth map to find the depth at the center of the bike, the coordinates of which were obtained by template matching and used this value along with the focal length and the distance between the cameras obtained from their projection matrices to obtain the distance of the obstacle from the camera. This task is of much use to ARK as it can be used by drones for avoiding obstacles.

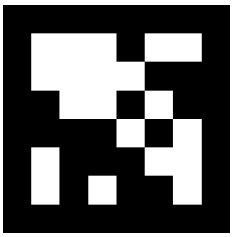
VII. Introduction - Task 3.1 Part 1

In this task, we were provided with an image of an ArUco marker and were asked to determine its ID and relative pose. We were also supposed to display these on the image. In the given time frame, I was only able to complete the ID part and could not do the pose estimation.

ArUco markers are binary square fiducial markers that consist of a wide black border and an inner binary matrix which determines its ID. The main benefit of these markers is that a single marker provides enough correspondences (its four corners) to obtain the camera pose.

VIII. Problem Statement

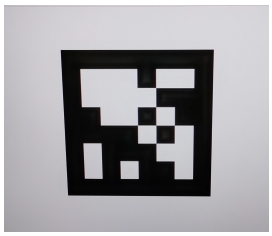
We were provided with the following ArUco marker and were asked to determine its ID and pose.



We were supposed to estimate these both from an image and from live video capture from our webcams and print them on the image.

IX. Final Approach

For the detection of ArUco marker ID from image, I first took the image of the ArUco marker and pasted it on word. I then took a photograph of the image on word and used it to detect the marker. This is what the image looks like.



I read this image in python using the cv2.imread() function.

```
img=cv2.imread("detect aruco.jpg")
```

For the video capture part, I first took the webcam video as input using the cv2.VideoCapture(0,cv2.CAP_V4L2) function. The webcam is closed whenever ” ” (Space) is pressed

```
vid=cv2.VideoCapture(0,cv2.CAP_V4L2)
while True:
    isTrue,img=vid.read()

    if cv2.waitKey(10) & 0xFF==ord(' '):
        vid.release()
        break
```

I then found the ID of the ArUco marker that my webcam was reading by using the cv2.aruco.detectMarkers() function of OpenCV.

The marker detection processes consists of 2 major steps.
First, adaptive thresholding is done on the image to segment the markers and then the contours that are either not convex or do not estimate a square are discarded. Some filtering is also done based on the contour size.
Second, A perspective transformation is applied to obtain the marker in its canonical form which is then thresholded to separate the white and black bits.The image is then divided into different cells according to the marker size and the border size. The number of black or white pixels in each cell is counted to determine if it is a white or a black bit. Finally, the bits are analyzed to determine if the marker belongs to the specific dictionary.

The cv2.aruco.detectMarkers() function takes in three parameters.
1) inputImage
2) dictionary: It refers to a set of markers with a specific application. For the given ArUco marker, the dictionary required was DICT_6X6_250 which was obtained by using the cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250) function.
3) parameters: This includes all the parameters that can be customized during the detection process. This is obtained by the cv2.aruco.DetectorParameters_create() function.

The cv2.aruco.detectMarkers() function also returns 3 values:
1) markerCorners: A list containing the (x,y) coordinates of the detected ArUco markers.
2) markerIds: A list containing the detected IDs of the markers.
3) rejectedCandidates: A list of potential markers that were found but rejected.

```
12  dict=cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
13  parameters=cv2.aruco.DetectorParameters_create()
14
15  (corners,ids,rejected)=cv2.aruco.detectMarkers(img,dict,parameters=parameters)
16
```

The value of the ID was then printed on the output image/video. To do this I first joined the corners obtained on the image and printed the value on the image.

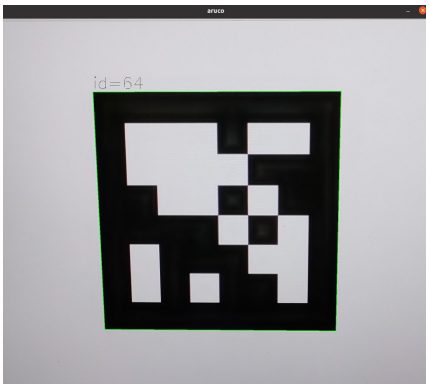
```
18  if len(corners)>0:
19      ids=ids.flatten()
20
21      for (corner,id) in zip(corners,ids):
22          corners=corner.reshape(4,2)
23          (tl,tr,br,bl)=corners
24
25          tr=(int(tr[0]),int(tr[1]))
26          br=(int(br[0]),int(br[1]))
27          tl=(int(tl[0]),int(tl[1]))
28          bl=(int(bl[0]),int(bl[1]))
29
30          cv2.line(img,tl,tr,(0,255,0),2)
31          cv2.line(img,tr,br,(0,255,0),2)
32          cv2.line(img,br,bl,(0,255,0),2)
33          cv2.line(img,bl,tl,(0,255,0),2)
34
35          cv2.putText(img, "ID = " + str(id),(tl[0], tl[1] - 15),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 255,
36
37  cv2.imshow("aruco",img)
```

For the image part, the image needed to be downsized to be displayed propery on the screen. The code for downsizing the image is as follows:

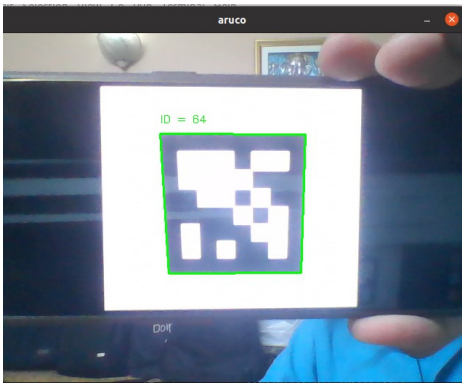
```
resized = np.full(((img.shape[0])//2,(img.shape[1])//2,3),255,dtype=np.uint8)

for i in range((img.shape[0])//2):
    for j in range((img.shape[1])//2):
        sum1,sum2,sum3=0,0,0
        for k in range(2):
            sum1+=img[i*2+k][j*2+k][0]
            sum2+=img[i*2+k][j*2+k][1]
            sum3+=img[i*2+k][j*2+k][2]
        resized[i][j][0]=sum1/2
        resized[i][j][1]=sum2/2
        resized[i][j][2]=sum3/2
```

The output image looks like: (This is a screenshot of my webcam)



The output video looks like: (This is a screenshot of my webcam)



X. Results and Observation

The ID of the ArUco marker was found out to be 64 and was printed on the image and on the video from the webcam.

XI. Future Work

The pose of the ArUco marker can also be added to the program. ArUco markers are of great importance in computer vision applications because their detection is simple and quick. Pose estimation is also of great importance in robot navigation and augmented reality.

XII. CONCLUSION

This task was about the detection and pose estimation of ArUco markers. The detection of ArUco markers was performed by using the cv2.aruco.detectMarkers() function of OpenCV module. The ID that was found out was printed on the video obtained from the webcam. Detection and Pose Estimation of ArUco markers is of great importance to ARK as aerial robotics deals a lot with computer vision. ArUco marker detection and pose estimation has a huge application in perception and control of UAVs.

References

[1] https://docs.opencv.org/4.x/dd/d53/tutorial-py-depthmap.html
[2] https://learnopencv.com/geometry-of-image-formation/
[3] https://www.youtube.com/watch?v=jhOTm3MZDaY
[4] https://www.cse.psu.edu/ rtc12/CSE486/lecture12.pdf
[5] https://www.cs.cmu.edu/ 16385/s17/Slides/11.1-Camera-matrix.pdf
[6] https://docs.opencv.org/4.x/d5/dae/tutorial-aruco-detection.html
[7] https://programming.vip/docs/3d-pose-estimation-using-aruco-tag-in-python.html
[8] https://developer.apple.com/documentation/avfoundation/avcameracalibrationdata/2881 intrinsicmatrix