Aerial Robotics Kharagpur Task Round

Adyan Rizvi 21MA10006 Section 12

***Abstract*— I have to admit I was a bit intimidated by the tasks when they were first announced on the 25th of May. There were 5 tasks and I thought that it would be difficult for me to even complete two. However after days of learning, coding and effective time management I have managed to complete quite a few tasks and have learned a lot along the way.**

All the problems in the task revolved around computer vision and image processing. As said before there are 5 tasks and i have chosen to do all these tasks using the Python programming language. My editor of choice was Visual Studio code and for image processing I mainly used the opencv python library which made my tasks way easier. 2 of the tasks involved ROS. I have used ROS Noetic on Ubuntu 20.04 for these tasks.

I would like to thank ARK for coming up with this task round along with the seniors of ARK who helped me whenever I had doubts and gave me a lot of guidance throughout the way.

I. TASK 1

A. INTRODUCTION

Robot Operating System (ROS) is a framework which is used extensively in making robots or autonomous machines. It provides many advantages over traditional methods and is language agnostic. ROS works with multiple objects such as publishers, subscribers, topics etc which is necessary to know effectively in order to implement projects. The task deals with ROS and more specifically reading messages from publishers and doing operations on them.

B. PROBLEM STATEMENT

There are two parts to the task .

For the first part, we are provided with a sequence of integers of length n. We can make guesses on what that number can be. Once we make a guess in return we will receive 2 outputs i.e. number of 'Cents' and 'Dollars'.'Cents' denote the number of digits guessed which are present in the given sequence but are out of position, whereas 'Dollars' represent the number of digits guessed which are present in the sequence at the very position we guessed them at. The game ends when you get exactly n Dollars. Your solution must be optimised enough to run for numbers of length 10. Your task is to implement the solution using a template given in ROS.

For the second part of this task, everything stays the same just that instead of 'Dollars' and 'Cents' provided to us after every interaction, we will be provided a value X. The output of the interaction will be as follows +2 for every digit guessed that is present in the string in the correct position +1 for every digit guessed that is present in the string but is out of position. X will basically be the summation of these values. The game ends when we reach the state X=2n where n is the length of the number. Using the above interaction, you have to try to correctly guess the number.

C. INITIAL ATTEMPTS

Initially I faced several problems trying to get the publishers and subscribers receive messages properly. A problem I often faced was that the other node would not return the message in time whereas the current node would already move ahead to the next steps, causing major errors. I primarily fixed this using rospy.Rate(1).sleep() in order to pause the node for a while.

I initially thought of finding dollars in the number by switching two digits among themselves and then noticing the change in dollars and cents. Though this method was used for 10 digit numbers I ended up using a different method for numbers with less than 10 digits which will

be discussed below.

D. FINAL APPROACH

I will be dividing this into four parts

Part 1 for 10 digit numbers:- I used a strategy of switching places of two digits and noticing the change in dollars. If the no of dollars increased by 2 or decreased by 2 then that means the digits were dollars in the old number and the new number respectively. Once some dollars were found i started exchanging digits with already known dollars to find new dollars until the number was found.

```
if(dollar1==dollar2):
    break
# If dollar1-dollar2=2 then digits at pos i and j are dollars in original number
if(dollar1-dollar2==2):
    if(guess[i] not in dollars_array):
        dollars_array.append(guess[i])
        posd.append(i)
    if(guess[j] not in dollars_array):
        dollars_array.append(guess[j])
        posd.append(j)
# If dollar1-dollar2=-2 then digits at pos i and j are dollars in new number
if(dollar1-dollar2==-2):
    if(guess2[i] not in dollars_array):
        dollars_array.append(guess2[i])
        posd.append(i)
    if(guess2[j] not in dollars_array):
        dollars_array.append(guess2[j])
        posd.append(j)
```

Fig. 1. Part 1 for 10 digits

Part 2 for 10 digit numbers:- Since no of dollars will be just x-10, the above strategy was used for this par as well

```
if(dollar1==dollar2):
    break
# If dollar1-dollar2=2 then digits at pos i and j are dollars in original number
if(dollar1-dollar2==2):
    if(guess[i] not in dollars_array):
        dollars_array.append(guess[i])
        posd.append(i)
    if(guess[j] not in dollars_array):
        dollars_array.append(guess[j])
        posd.append(j)
# If dollar1-dollar2=-2 then digits at pos i and j are dollars in new number
if(dollar1-dollar2==-2):
    if(guess2[i] not in dollars_array):
        dollars_array.append(guess2[i])
        posd.append(i)
    if(guess2[j] not in dollars_array):
        dollars_array.append(guess2[j])
        posd.append(j)
```

Fig. 2. Part 2 for 10 digits

Part 1 for less than 10 digit numbers:- I generated a digit that was not in the number. I then replaced the digits in the original number with this new digit and calculated the difference in dollars. If the no of dollars increased or decreased by 1 then that means the digit was a dollar in the old and the new number respectively.

```
if(dollarcent==999):
    return
# If new_dollar-dollar==-1 then that means digit in original number was dollar
if(new_dollar-dollar==-1):
    dollars_array.append(guess[i])
    posd.append(i)
    c+=1
# If new_dollar-dollar==1 then that means digit in new number is dollar
if(new_dollar-dollar==1):
    dollars_array.append(guess2[i])
    posd.append(i)
    c+=1
if(c==dollar-prev_dollar+1): # If we find the dollars we need to find we escape in order to not waste loops
    break
```

Fig. 3. Part 1 for less than 10 digits

Part 2 for less than 10 digit numbers:- I generated a digit which was not in the number. I then replaced digits and noticed difference in x. If x increased or decreased by 2 then that means the digit in the original number and in the new number were dollars respectively. After some dollars were found, I replaced a dollar with the generated digit to check whether it was a cent(diff=1) or not(diff=0). I then replaced each digit in the number with the new digit and they were added to dollars array corresponding to difference in x and whether it was a cent or not.

```
if(new_x==999):
    return
# If diff is 2 then it means digit at pos i in original no is dollar
if(ori_x-new_x==2):
    dollars_array.append(guess[i])
    posd.append(i)
# If diff is -2 then it means digit at pos i in new no is dollar
if(ori_x-new_x==-2):
    dollars_array.append(guess2[i])
    posd.append(i)
# If diff is 1 and digit is cent then digit in original no is dollar
if(ori_x-new_x==1 and iscent==1):
    dollars_array.append(guess[i])
    posd.append(i)
```

Fig. 4. Part 2 for less than 10 digits

E. RESULTS AND OBSERVATIONS

No of tries required for each part considering 3 tries for each

Part 1 for 10 digit nos:- 80 64 93

Part 1 for 6 digit nos:- 48 68 53

Part 2 for 10 digit nos:- 77 87 69

Part 2 for 6 digit nos:- 166 140 122

F. FUTURE WORK

One thing that can be done is keeping track of the digits that will not be in the number. If there is no change in dollars or x then that means the digits in question do not belong in the number. We can keep track of them which will help us optimize our new guess generation.

Also the algorithm for the part 2 for less than 10 digits can be optimized further by keeping track of rejected array along with other optimizations.

CONCLUSION

With this we come to the end of the first task of the task round. Though the logic of the program is important the main purpose of this task was to accustom people to ROS which will come in handy while designing autonomous drones.

II. TASK 2

A. INTRODUCTION

We can't find out the distance between camera and an object in the image taken from said camera. This is because there is no reference or us to calculate the exact distance. This is a problem for autonomous vehicles or drones since knowing the distance between it and the obstacle is necessary for it to work properly.

There is a way around it. If we take images from two cameras and we know the distance between the two cameras, we can use the concept of similar triangles to find out the distance of the object from the cameras or the depth of the object

The first task deals with finding out depth of object using images from two cameras(also known as stereo cameras with the images known as stereo images).

B. PROBLEM STATEMENT

In this task we are provided with a pair of images using two stereo cameras taken at the same time. The images are as follows

Fig. 5. Left Image

Fig. 6. Right Image

We have also been provided with the image of a bike as shown below(Fig 7).

Our task is to first create a depth map using the two stereo images. Next we have to find the bike in both the images and then find out the depth or the distance between the bike and the cameras. To do the above we have also been provided with the camera matrices of both cameras(Fig 8).

C. INITIAL ATTEMPTS

The problem is straightforward so there weren't many initial attempts. The main challenge of this task was to understand the camera matrix and how disparity is calculated and how using disparity we can find the depth of object.

Fig. 7. Bike

```
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]]
```

Fig. 8. Camera Matrices

As far as depth maps go we can use an opencv function to generate it for us.

D. FINAL APPROACH

The first task i tackled was generating the depth map.

First a stereo object was created using StereoBM_create function. Then disparity map was created using stereo object and matplotlib was used to plot the depth map(Fig 9).

```
# Generating depth map
stereo = cv2.StereoBM_create(numDisparities=0, blockSize=5)
depth_map = stereo.compute(gray1,gray2)
plt.imshow(depth_map,'gray')
plt.show()
```

Fig. 9. Generating Depth Map

The next task was to find the bike in both the left and right images. This was relatively simple using the opencv function matchTemplate(). TM_SQ DIFF_NORMED was used to match the template. The top left and the bottom right coordinate of the bike in both images was returned. Using it the midpoint y coordinate of the bike was calculated. This will be used later for depth estimation.

```
res=cv2.matchTemplate(gray1,bike_gray,cv2.TM_SQDIFF_NORMED)
min_val,max_val,min_loc,max_loc=cv2.minMaxLoc(res)
```

Fig. 10. Matching Template

First before we calculate depth, the camera matrices of both cameras were to be found. On using QR Normalization.

rotation matrix left= Identity matrix
rotation matrix right= Identity matrix
translation matrix left=[2 -0.25 1.4]
translation matrix left=[2 0.25 1.4]

fx=640 and fy=480

Thus the distance between the two cameras is 0.25-(-0.25)= 0.5

The depth of an object is calculated by the formula $f * (distancebetweencameras)$/ disparity

Where disparity is the distance between corresponding pixels in both images

This however poses a challenge. How to find corresponding pixels? For this i have applied a method known as sum of absolutes method. To find the corresponding pixel i compare the pixel along with adjacent pixels in a row. For example for pixel with column number 525 and pr(parameter)= 3, I will sum up the absolute difference in intensities of 525 along with 3 pixels to the left and 3 pixels to the right with the second image. Whichever column no in the second image gives me the lowest sum will be the corresponding pixel(We can check in the same row in the second image as row coordinate does not change in stereo images).

Once the corresponding pixel has been found we can calculate disparity as =Column no of original pixel -Column no of corresponding

```
for i in range(tl[0],bottom_right[0]):
    for j in range(tl2[0],bottom_right2[0]):
        img1[tl[1],i]=[0,255,0]
        for r in range(2*pr+1):
            sad+=abs(int(gray1[tl[1],i-pr+r])-int(gray2[tl[1],j-pr+r]))
        if sad<min:
            min=sad
            pos=j
        sad=0
    min=10000
    sum.append(abs(pos-i))
```

Fig. 11. Camera Disparity Using Sum of Absolute Difference

pixel in second image. Doing this over the pixels corresponding to our bike will give us the distance between the bike and the camera. For doing this i took the middle row of our bike. I then scanned column-wise and then found the corresponding pixel in the second image for each pixel in the first image. I calculated the disparity and took the mean of the disparity as my final disparity. Thus my depth = fy*(0.5)/disparity.

```
disparity=statistics.mode(sum)
print("The distance of the bike is "+str(240/disparity)+" in m")
```

Fig. 12. Depth calculation

E. RESULTS AND OBSERVATION

The depth map generated is shown as below(Fig 13).

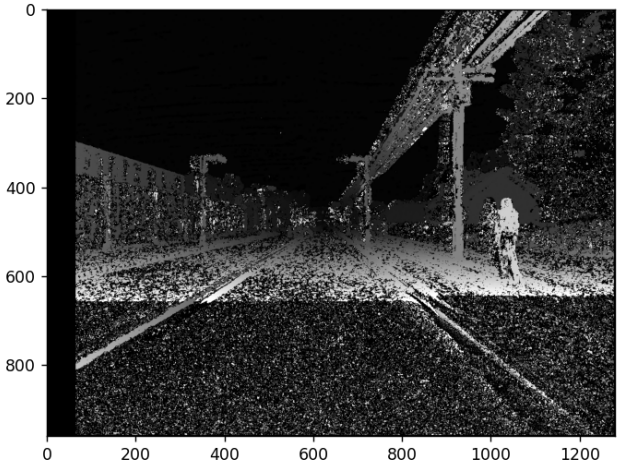

Fig. 13. Depth Map

Distance of bike as pr changes
pr=3 Distance = 23.844 m
pr=5 Distance = 23.2911 m
pr=9 Distance = 23.39 m

If mode of disparities was taken instead of mean then Distance(pr =5) = 24.0 m

F. FUTURE WORK

One major problem with this approach is that in real life we dont have the image of the obstacle with us. Therefore we would have to apply techniques in order to identify obstacles in the real world. One could do this using feature matching or other methods.

CONCLUSION

With this we come to the end of the task. This task has a very important application related to ARK. For autonomous drones it is necessary to know its obstacles and how far away they are from them. Thus this method is of upmost importance for them.

III. TASK 3

A. INTRODUCTION

For image processing it is very important for us to know the pose of our camera with respect to our targets. However this is more difficult than it sounds. To make it easier we can use something known as aruco markers in order to find pose of camera easily.

This task deals with detecting aruco markers in an image and then estimating the pose of a camera using that aruco marker.

B. PROBLEM STATEMENT

We have to detect an aruco marker in an image and print the tag id and pose on the image itself. We also have to do the same for live video capture. Then we have to detect the aruco marker using our own functions(without using library functions) and estimate the pose using our own functions as well.

Fig. 14. Aruco marker to be detected

C. INITIAL ATTEMPTS

The detection and pose estimation of the aruco marker with library functions was fairly straightforward. However detecting the aruco marker using my own functions used a lot of hit and trial and reading.

First i tried to do it using canny edge detection. However that did not work out as planned. Later however i decided to try and detect corners and that did not work as planned either. However later i decided to detect the aruco marker using contour detection. However that in itself posed some problems. After all this I finally got how to detect my aruco marker which I will discuss below.

D. FINAL APPROACH

First let me discuss how i detected the aruco marker using library functions.

opencv has a sublibrary known as aruco using which i detected the marker. I used aruco unctions in order to identify the marker and draw it on the image.

For pose estimation i first had to calibrate my camera and obtain various parameters

```
#Detecting aruco marker
key=getattr(aruco,f'DICT_6X6_250')
arucoDict=aruco.Dictionary_get(key)
arucoParam=aruco.DetectorParameters_create()
bbox,ids,rejected=aruco.detectMarkers(img1,arucoDict,parameters=arucoParam)
```

Fig. 15. Detecting aruco marker using library function

such as camera matrix, distortion vectors etc. For calibrating my camera I used chessboard camera calibration. I took 5 images of with my webcam with a 8 by 6 chessboard in different orientations and then calibrated my camera using calibrateCamera function. Then i saved the different parameters in an .npz file using np.savez function.

```
ret, cameraMatrix, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, frameSize, None, None)

np.savez("Camera parameters",cameraMatrix=cameraMatrix,dist=dist,rvecs=rvecs,tvecs=tvecs)
```

Fig. 16. Calibrating camera

Now to do pose estimation I simply used estimatePoseSingleMarker function of cv2.aruco . It generated the pose of the aruco marker for me. Then i used drawFrameAxes to finally draw the pose on the marker and then showed the image as the final result

```
#Estimating pose
new_rvec,new_tvec,_=aruco.estimatePoseSingleMarkers(bbox, 0.05, mtx,dist)

# Drawing axes
for i in range(len(new_rvec)):
    img1=cv2.drawFrameAxes(img1,mtx,dist,new_rvec[i],new_tvec[i],0.03)
```

Fig. 17. Estimating pose

For live video capture I simply put the above logic in while loop. If code detected aruco marker then the frame was shown simply otherwise I would draw the aruco marker boundary and pose on the marker and display it.

Now let me come to how I detected aruco marker using my own functions.

First I converted image to gray scale and used adaptive thresholding. This allowed me to isolate the boundary of the aruco marker. Then i used detectContours function(set at external)

to detect all contours. Next step would be to check whether contour is square or not. I first ran the contours through approxPolyDp to get approximate polygon. I then checked if the contour was convex and that its area was above a certain threshold value. Doing all the above I was able to isolate the squarish contours.

```
k=list()
for i in range(len(contours)):
    # approxpolyDP approxiamtes the contour to minimumn no of points
    a=cv2.approxPolyDP(contours[i],0.1*cv2.arcLength(contours[i],True),True)

    if(len(a)!=4 or cv2.isContourConvex(a)!=True):
        continue
    if(cv2.contourArea(a)<100):
        continue
    k.append(a)
return k
```

Fig. 18. Detecting square contours

Next step would be to check if detected contour was actually an aruco marker or not. First I would use perspectiveTransform to transform the detected contour onto flat image. Then I would apply thresholding to convert it into binary. I would divide the transformed contour into 8x8 grid and check if the mean of the pixels in the center of each grid is above 240(white) or below 10(black). I also check if there are a certain no of black and white grids(so that a blank white paper doesn't pass the test). If the above conditions are satisfied then it is an aruco marker and we draw it on the image and display it(Fig 19).

```
flag=0
c1=0
c2=0
for i in range(8):
    for j in range(8):
        a=(k2[i]+k2[i+1])//2
        b=(k2[j]+k2[j+1])//2
        temp=result[a-5:a+5,b-5:b+5]
        if(temp.mean()>=240): # If bit is white
            c1+=1
        if(temp.mean()<=10): # If bit is black
            c2+=1
# If sum not = 64 then it is not aruco | c1 and c2 condition to check if it is not simply blank
if((c1+c2)!=64 or c1<=10 or c2<=10):
    flag=1
if(flag==0):
    new_contour.append(contour)
    #Showing detected aruco
    cv2.imshow("Detected aruco",result.astype(np.uint8))
return new_contour
```

Fig. 19. Detecting Aruco Marker

E. RESULTS AND OBSERVATION

As you can see below tag id and marker along with pose of the marker is generated by the code(with library functions). It also works with live video capture tracking aruco and its pose in real time(Fig 20).

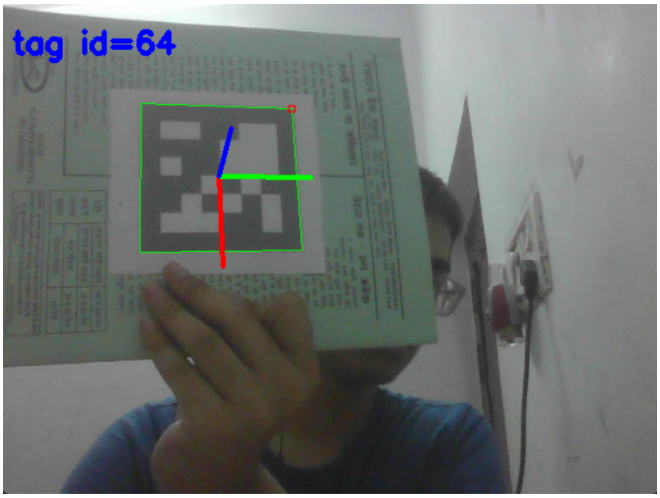

Fig. 20. Detected Aruco with pose

In the code not using library functions it detects the aruco marker(fig 22) as shown below and draws a border around it in the original image(fig 21).

F. FUTURE WORK

One problem with the above code(without library functions) is that it does not print the tag id of the aruco marker on it. This can be changed by calculating its signature and using aruco dict to find out the tag id. Also it relies on custom parameters so it is possible it may not work well in certain circumstances.

CONCLUSION

With this we come to the end of the second task of ARK. This task is very useful when it comes to autonomous robotics. It is necessary for drones to know where they are relative to their environment. Aruco markers can come handy in these circumstances.

IV. TASK 4

A. INTRODUCTION

For an autonomous drone or vehicle to function properly, it must detect its obstacles efficiently.

Fig. 21. Detected Aruco in image

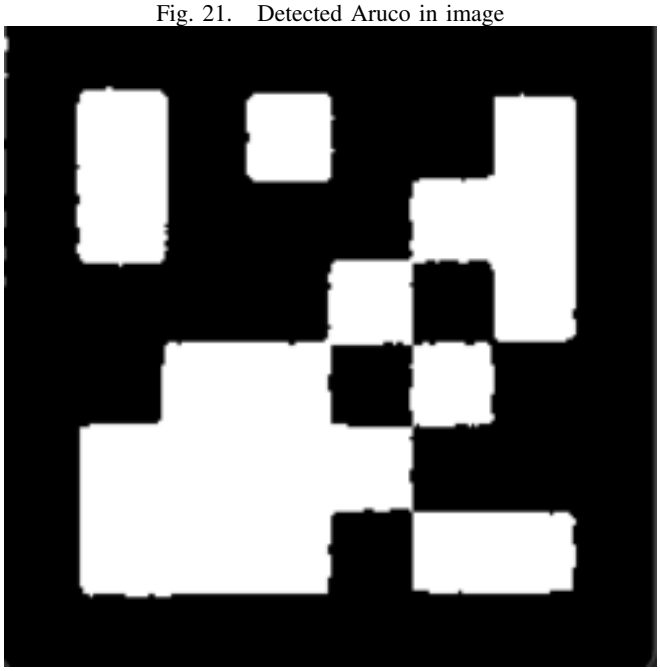

Fig. 22. Aruco extracted

However sometimes the obstacles may not be stationary, ie they may be moving. Thus it also becomes important for the robot to estimate where they may be after a while.

The estimation of motion of objects is known as optical flow estimation. It is a highly valuable topic for researchers and many methods have been derived to estimate optical flow. One of them is known as Hierarchical Lucas Kanade method. This task deals with the same.

B. PROBLEM STATEMENT

We have been given three images which have been taken one after the other.

Fig. 23. First Image

Fig. 24. Second Image

We have also been given two videos, one of the earth rotation and another is of a jellyfish swimming in the ocean

Our task is to estimate the optical flow of the images and the videos using Hierarchical Lucas Kanade Method.

Fig. 25. Third Image

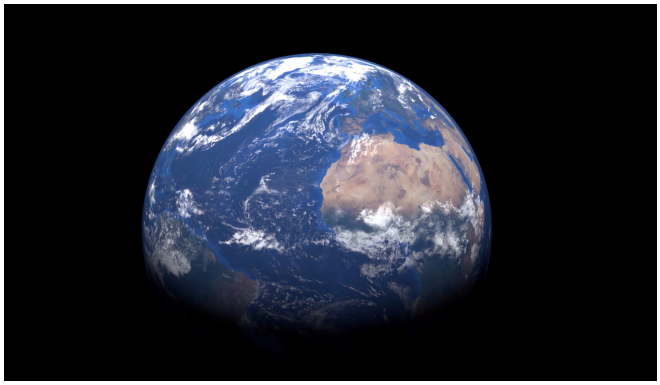

Fig. 26. Frame of earths rotation

Fig. 27. Frame of jellyfish swimming

C. INITIAL ATTEMPTS

A lot of time was spent on understanding how the method works . It is quite mathematics intensive with linear algebra, differential calculus and Gaussian's been used. After understanding how the method worked however, implementing it became much easier.

D. FINAL APPROACH

First lets understand the method itself.

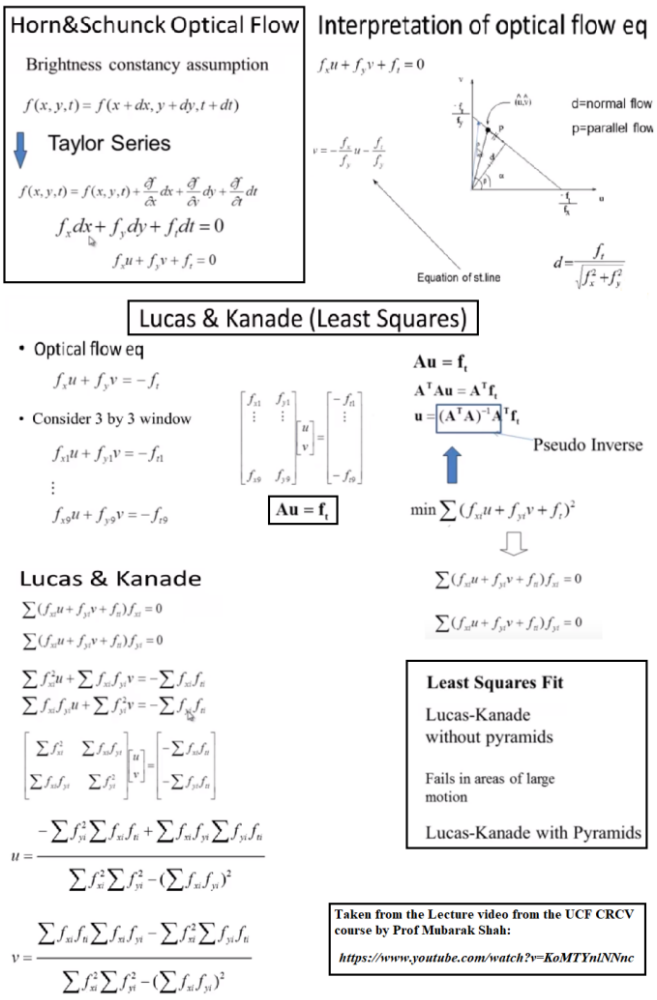

Fig. 28. Lucas Kanade

The code implementation of it is as follows.

Another important part of the assignment is implementation of Gaussian pyramids. The optical flow is estimated of lower level then is interpolated to the higher level to find new optical flow and this process is repeated until the last image. The Gaussian pyramid is generated using the Gaussian filter and then sub sampling the

```
# Looping through image
for h in range(w//2, shape[0] - w//2,w):
    for k in range(w//2, shape[1] - w//2,w):
        # Getting Ix Iy and It values of window surrounding point
        Ix_window = Ix[h - w//2 : h + w//2 + 1, k- w//2 : k + w//2 + 1,].flatten()
        Iy_window = Iy[h - w//2 : h + w//2 + 1,k - w//2 : k + w//2 + 1,].flatten()
        It_window = It[h - w//2 : h + w//2 + 1,k - w//2 : k + w//2 + 1,].flatten()
        # Getting array A and b
        A = np.asarray([Ix_window, Iy_window]).reshape(-1, 2)
        b = np.asarray(It_window)
        # Getting A transpose A
        ATA = np.transpose(A) @ A
        eig, _ = np.linalg.eig(ATA)
        # Noise thresholding
        if eig[0] < t or eig[1]<t:
            continue
        # Getting soln
        ATAI = np.linalg.pinv(ATA)
        sol = ATAI @ np.transpose(A) @ b
        #Storing soln
        u[h, k], v[h, k] = sol[0],sol[1]

return u,v
```

Fig. 29. Code implementation of Lucas Kanade method

image by choosing every second row and column.

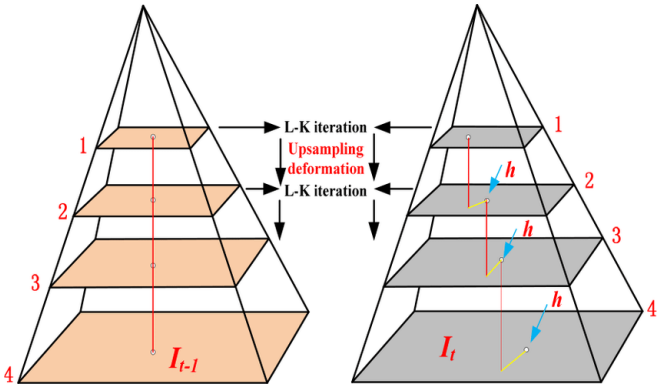

Fig. 30. Gaussian Pyramids

Code implementation of Gaussian pyramids(Fig 31).

```
for i in range(gauss-1,0,-1):
    # Interpolating u and v for new u and v
    shape=(2*u.shape[0],2*u.shape[1])
    u_new=np.full(shape,0).astype(np.float64)
    v_new=np.full(shape,0).astype(np.float64)
    for k in range(u_new.shape[0]):
        for l in range(u_new.shape[1]):
            if(k%2!=0 or l%2!=0):
                continue
            u_new[k,l]=2*u[k//2,l//2]
            v_new[k,l]=2*v[k//2,l//2]
    u=u_new
    v=v_new
    # Applying u and v to image and then finding new optical flow
    warped=first_image[i-1].copy()
    for h in range(warped.shape[0]):
        for k in range(warped.shape[1]):
            if(u[h][k]==0 and v[h][k]==0):
                continue
            if((h+v[h][k]>=warped.shape[0] or h+v[h][k]<0) or (k+u[h][k]>=warped.shape[1] or k+u[h][k]<0)):
                continue
            warped[int(h+v[h][k])][int(k+u[h][k])]=first_image[i-1][h][k]
    u_corr,v_corr=opticalflow(warped,second_image[i-1],w)
    # Adding corrected optical flow to original guesses
    u=u+u_corr
    v=v+v_corr
return u,v
```

Fig. 31. Code implementation of Gaussian Pyramids

After the optical flow was calculated we draw it on the image using arrows to represent the motion. (Fig 32)

```
def drawof(img1,u,v,w): # This function draws the optical flow on the image
    # Looping through the image
    for h in range(int((w-1)/2),img1.shape[0]-int((w-1)/2),w):
        for k in range(int((w-1)/2),img1.shape[1]-int((w-1)/2),w):
            if(abs(u[h,k])+abs(v[h,k])<0.8): # If the optical flow is small we dont draw it to avoid noise
                continue
            # We add the u array to k because we defined u as the change in positive x direction which corresponds to columns
            img1=cv2.arrowedLine(img1,(k,h),(int(k+u[h,k]),int(h+v[h,k])),(0,0,255),1)
    return img1
```

Fig. 32. Drawing optical flow

E. RESULTS AND OBSERVATIONS

The optical flow from the first and second image(Fig 33).

Fig. 33. Optical flow between first and second image

The optical flow from the second and third image.(Fig 34)

The optical flow from the video of the earths rotation(Fig 35).

The optical flow from the video of the jellyfish (Fig 36).

* For window size =5 and Gaussian layers=3 for images

F. FUTURE WORK

The algorithm can definitely be optimized further. Also different window sizes can be used for different results. However Lucas Kanade method does not work in certain situations such

Fig. 34. Optical flow between second and third image

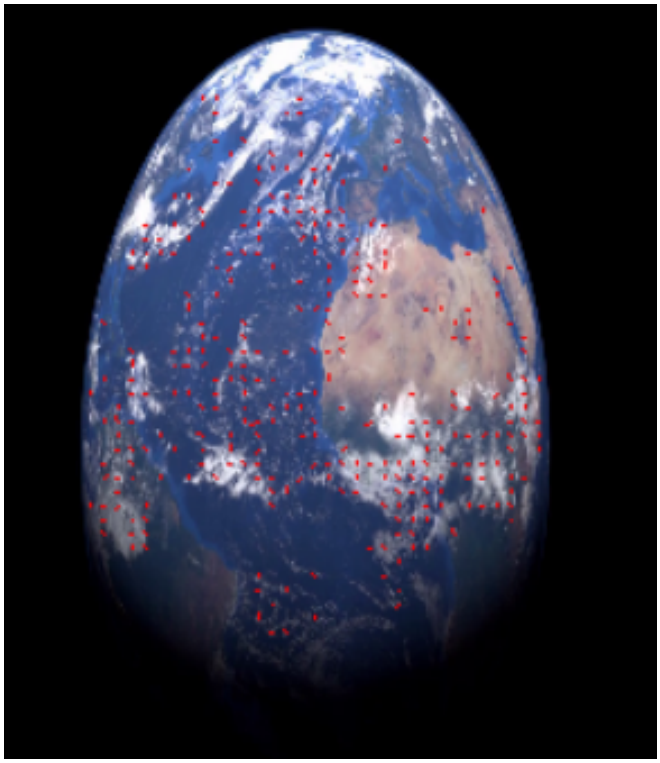

Fig. 35. Optical flow of earths rotation

as when the image is very homogeneous or there are very sharp edges.

There is also a lot of noise. There are a lot of optical flow that we don't need. So we can apply feature extraction to only get optical flow of the required or important parts of the image.

Other more efficient methods such as RAFT can also be used.

Fig. 36. Optical flow of jellyfish

CONCLUSION

With this we come to the end of the 4th task of the task round. This is of much importance to ark as motion estimation of obstacles is essential for a drone to maneuver properly without it crashing.

V. Task 5

A. INTRODUCTION

Path planning is very important when it comes to autonomous drones. There are many algorithms which help in path planning with various obstacles and constraints. However the environment is not always known and we have to piece together the environment with snapshots that the drone manages to capture. Optimizing this process and path finding is a very important task with lots of scope. The task is related to the above.

B. PROBLEM STATEMENT

We are stuck in a dark maze where we cant see anything. However we have matches which we can use to light up the environment around

it. On lighting up a match, a ROS topic returns a snapshot of the maze around us. We can use any no of matches we want and our task is to stitch up the maze using these snapshots and solve the maze using any path finding algorithm we want to use.

C. INITIAL ATTEMPTS

Unfortunately I want able to attempt the stitching part of the task due to time constraints and various other issues. My initial idea was to use the snapshot to incrementally move in some direction and then take a new snapshot and stitch them together using our knowledge of the distance between the old and new centers. However I could never get the ROS publisher and subscriber to work properly and couldn't work on fixing it due to time constraints.

However I did solve the original maze which I will discuss below.

D. FINAL APPROACH

The maze was solved using the Breadth First Search(BFS) algorithm.

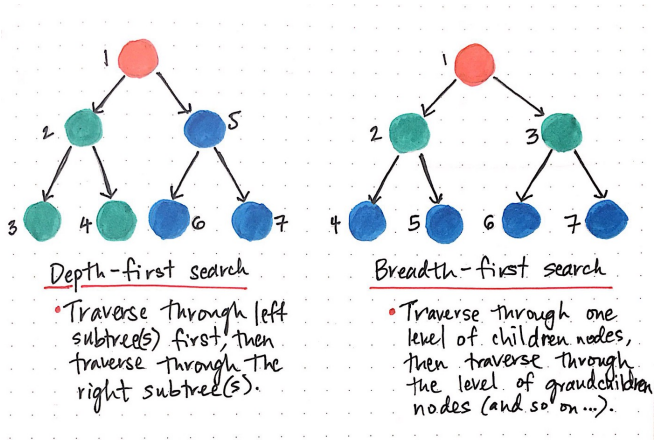

Fig. 37. BFS vs DFS

The breadth first search algorithm is an algorithm that searches for our end point by searching its neighbouring nodes(in this case pixels) before it searches its root pixels. This method guarantees the shortest path from one point to the other irrespective of position.

For implementing bfs, a queue is used. Queue works on FIFO principle(First in First out). We append the starting point of our maze. Then we append its neighbours to the queue if they are not black(part of wall) and have not already been visited. We then pop out the first element of the queue and repeat the above steps till we find the end point(red point). We also store the parent point of each neighbour in a seperate list

```
# If point is white and it is not visited
if(img[point[0]+i][point[1]+j]!=0 and visited[point[0]+i][point[1]+j]!=1):
    queue.append((point[0]+i,point[1]+j))
    visited[point[0]+i][point[1]+j]=1
    parent[point[0]+i][point[1]+j]=(point[0],point[1])
# If point is red we can end
if(img[point[0]+i][point[1]+j]<30 and img[point[0]+i][point[1]+j]>20):
    end=(point[0]+i,point[1]+j)
    flag=1
    break
```

Fig. 38. Solving maze using bfs

Once we find the end point, we find the path. This is done by adding the point to the path list and then its parent point. This is continued till we reach the start point. We then draw the path on the image.

Since there are two end points, we have to remove one. We use bfs to remove the red blob we have already found the path before in a similar algorithm as before. We then apply the same process as above to find the path for second end point.

```
path=maze_solve_bfs(gray) # Calling function first to solve for first end point
# Adding path to img
for a in range(len(path)):
    img[path[a][0]][path[a][1]]=(255,0,0)

new_gray=remove_blob_bfs(gray,path[-1]) # Removing first end blob

path=maze_solve_bfs(new_gray) # Calling function to solve the second end point
# Adding path to img
for a in range(len(path)):
    img[path[a][0]][path[a][1]]=(255,0,0)
```

Fig. 39. Solving the maze

E. RESULTS AND OBSERVATION

The maze in question is given below.

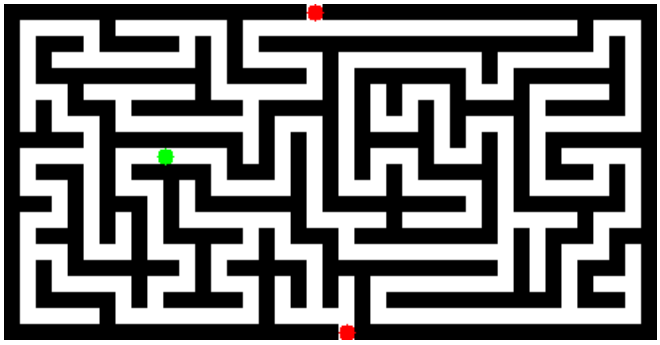

Fig. 40. Maze

This is the maze after we have solved it using bfs algorithm.

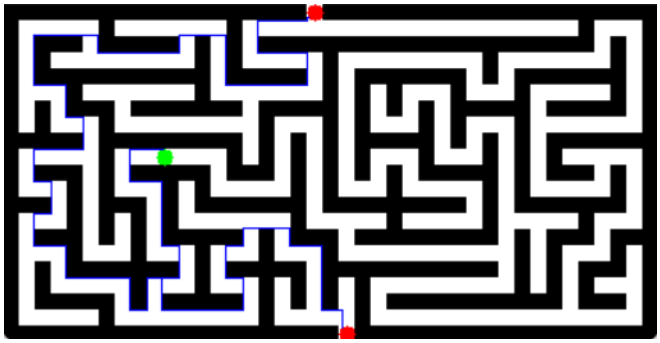

Fig. 41. Solved maze

One thing to note is that the algorithm gives us the absolute shortest path from one end to the other no matter the placement of the points.

F. FUTURE WORK

A major part of what can be done is tackling the image stitching part of the task. It would be interesting to tackle this in ROS and try to optimize the no of matches used to generate our maze.

Different algorithms such as RRT and RRT*, A* etc can also be used in order to solve the maze. One could also use these algorithms while stitching the image in order to optimize our method.

CONCLUSION

With this we come to the end of the final task of the ark round. This has many applications in the field of autonomous drones and autonomous robotics in general. There is also quite a lot of scope for further experimentation and research in this field.

VI. FINAL CONCLUSION

With this we come to end of the task round 2022 of ARK. Though i was not able to solve every single task, i learnt a lot about computer vision, python and various concepts related to image processing. Various sources were used for each task but by far the most important was the documentation of opencv. Other than that various other videos, blogs, lectures and research papers helped me to complete the tasks.

Thank you