Ashutosh Varshney
Roll No: 18CS30009

ARK Task(Task 1 and 2)

4th March 2019

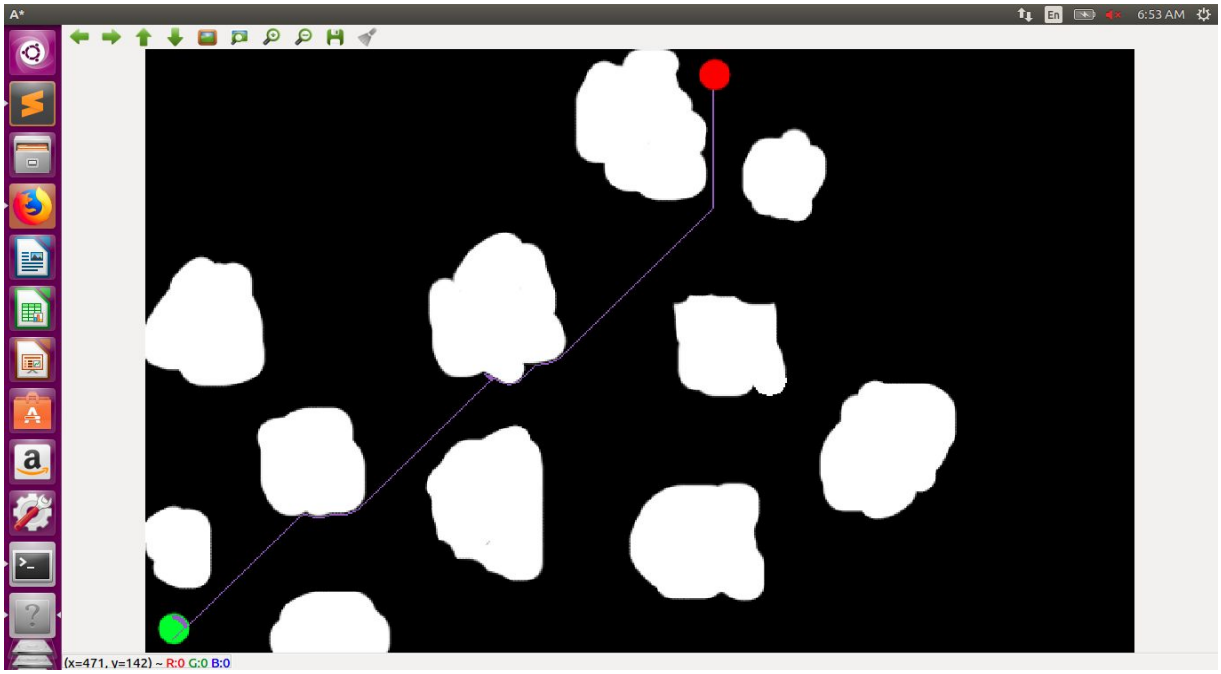
Dynamic Path Planning

OVERVIEW

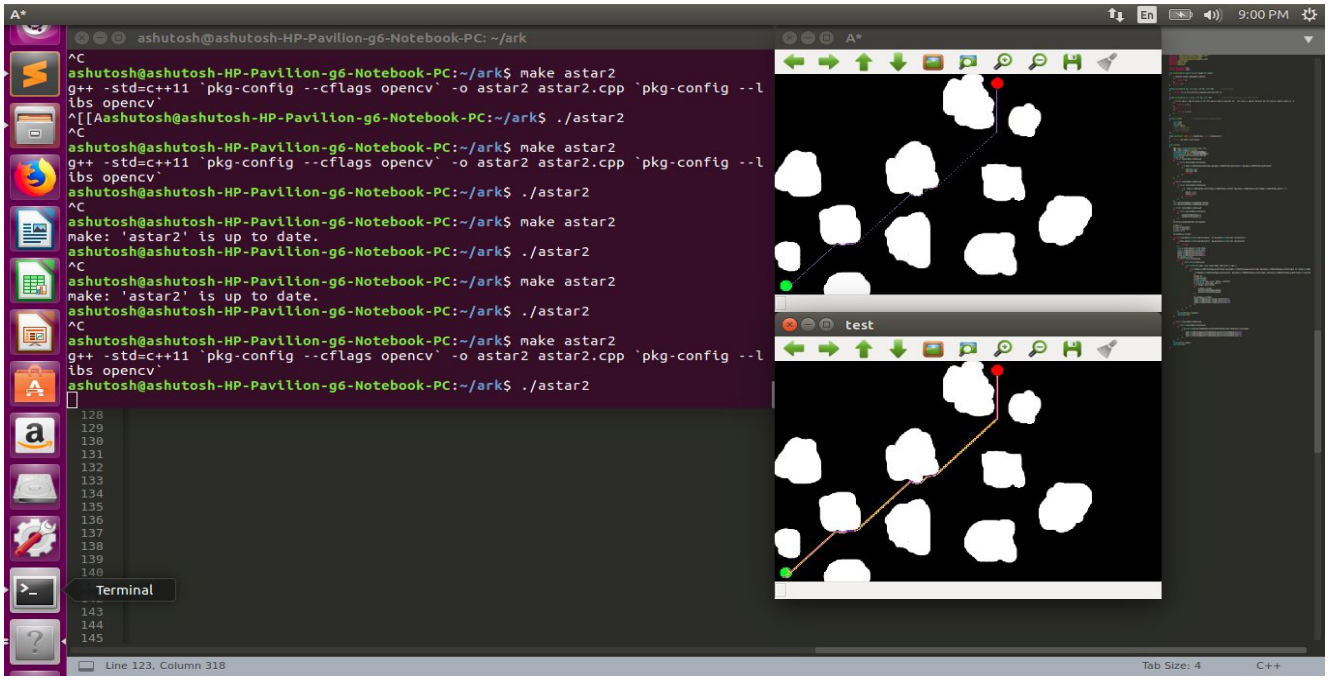
The task was to design a program which would create a path which will dynamically avoid obstacles and reach the destination. It includes reading a video frame by frame which has some static black objects and some dynamic blue circles and marking the path that the bot would take dynamically at each frame.

ATTEMPTS WITH THEIR OBSERVATIONS,PROBLEMS AND RESULTS

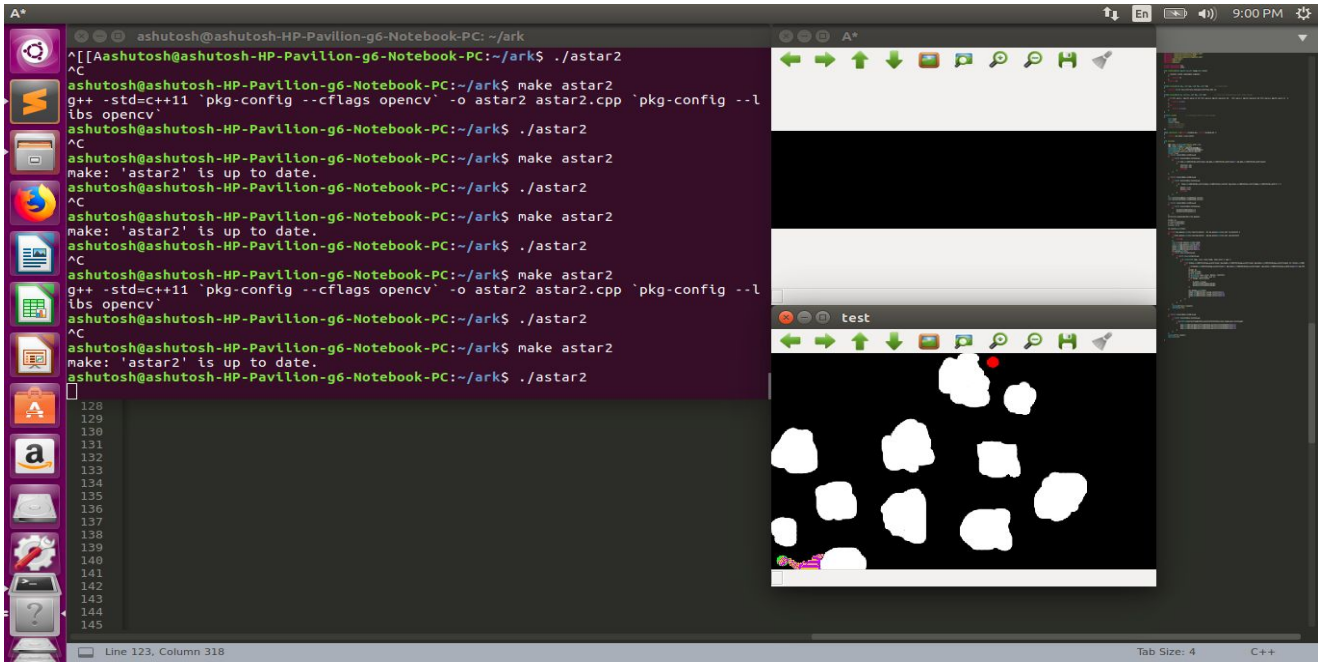
1. I first thought about A* and read about it from the net. Basically it includes 2 functions: the cost function and the heuristic. My idea was that while iterating pixel by pixel, if I only use heuristic instead of using both the functions, it would work. The code would be much faster than the A* algorithm and would serve the purpose. Then I searched about this approach on net and found Best First Search algorithm. I wrote the code and ran it on the image provided in the workshop. **OBSERVATION:** The code worked perfectly on a sample image provided in the workshop. The bot takes sharp turns. **RESULT:** Though the algorithm worked perfectly but it is greedy in nature and won't be efficient in all cases.



2. Due to the limitations of previous approach I tried to implement A* algorithm in a static environment. I faced some coding issues which includes using priority queue. I searched it on net and learned about constructor overloader. **OBSERVATION:** The code is slow compared to 1st approach. Presently it is not giving correct output at every run but sometimes it gives the correct output. **PROBLEM :** The same code on running, weirdly gives 2 different outputs; one of which is correct and the other is wrong. The region which is being explored weirdly changes each time I run it.



Correct output.(The yellow region in the image is the explored portion and the path is violet.)



(The same code now gives different explored region.)

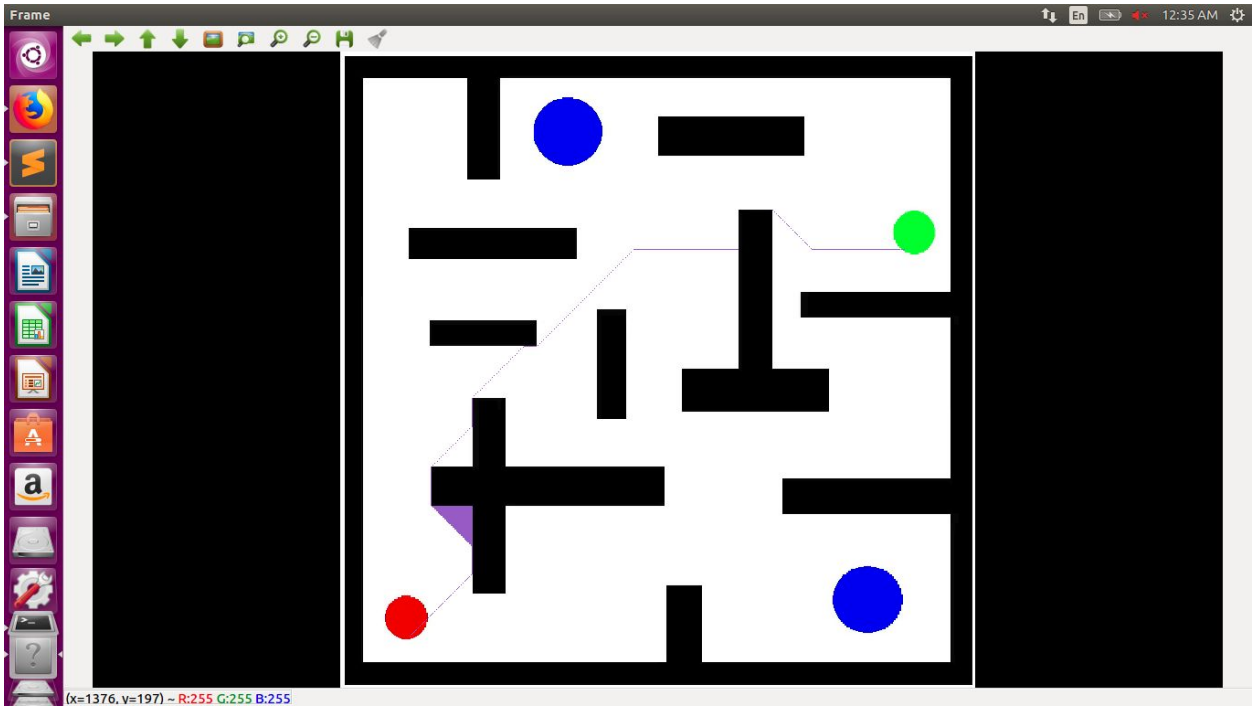
3. I switched to dynamic environment now and thought of implementing RRT algorithm as it would be very helpful in dynamic obstacle avoidance and would just divert the path instead of finding a new one. The algorithm will randomly iterate in many new possible directions at each iteration till it reaches the destination. So, if it finds any dynamic obstacle, it will simply take the other possible random iteration that was generated. I learnt about how to read a video frame by frame from net. **PROBLEM:** I have problems in implementation. While capturing the video frame by frame we use a while loop but while iterating I will require saving the next pixel for each random iteration from that point and I don't know how to do that.

LATEST ATTEMPT

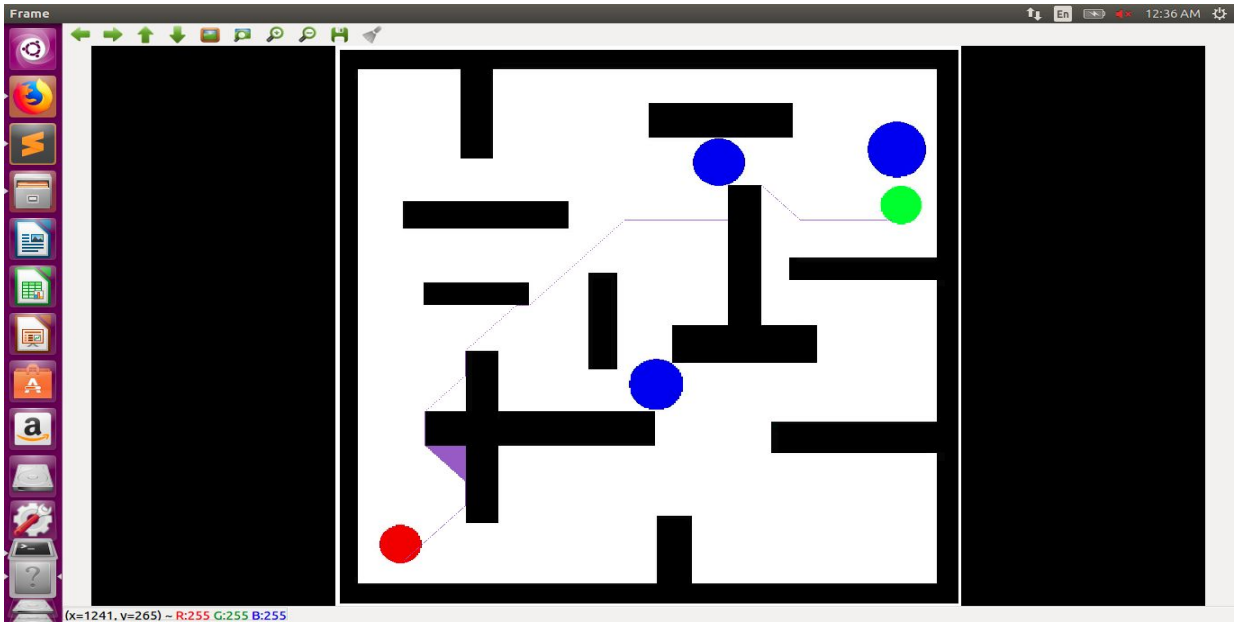
Based on the first attempt I used Best first search algorithm. The process includes reading a video frame by frame, treating each frame as a static environment and then printing the path frame by frame.

PROBLEMS:

1. The major drawback is that if there is a concave obstacle with the concave surface pointing the source in the path then the bot gets stuck there. This is because the algorithm used is greedy and is not efficient.
2. Since the code is written for a point sized bot, when the dynamic obstacle comes in its way it doesn't changes its path since there is still a thin gap between the dynamic and static obstacle and it doesn't completely block the path.



Problem 1: The concave shaped obstacle hinders the path.



Problem 2: The path doesn't change due to the thin gap.

RESULT

The present code is not efficient and requires improvement.

FUTURE WORK

1. My plan is to implement RRT perfectly as it will solve problem no. 1.
2. Secondly, I will write the code for a rectangular sized bot of some specific dimensions.

Disparity Map

OVERVIEW

In case of stereo images, there is a relative shift of objects in the left and right image known as disparity. Objects that are nearer have larger disparity while farther objects have smaller disparity. The task is to write a code that generates a depth map or the disparity map of the image using the 2 stereo images.

ATTEMPTS WITH THEIR OBSERVATIONS,PROBLEMS AND RESULTS

1. I wrote a code which uses kernel to divide the left image into smaller parts. Then it uses matchTemplate() to find its matching value in the right image and then minMaxLoc() to find the minimum or maximum of all the values according to the method used in the function and find the correct match. I read about the functions from OpenCV documentation. The function can use 6 different methods. Then after finding the correct match, the disparity can be found approximately by finding the difference between the x coordinates. Then the map can be generated by marking the intensity value to the pixel corresponding to the centre of the kernel by the formula : *I = (DISPARITY/TOTAL COLUMNS)*255.*
   PROBLEM: The template matching is extremely slow so in spite of successful compilation, the code doesn't give the output.
2. My next attempt was to do the matching normally by creating a 3x3 temporary kernel from left image and sliding over the right image. Since the complexity will be O(n^4) I took small portions of the test images(i.e. The chair in those images). And the rest procedure remains

the same. **PROBLEM:** The code compiles successfully but is not giving correct output and I am unable to detect the problem.

3. Then I searched about the techniques used presently to do the job and thus I found StereoBM algorithm which is a Block matching algorithm. Block matching is almost same as the approach I used previously.
4. I studied **UDACITY COMPUTER VISION COURSE GEORGIA TECH. UNIVERSITY from LESSON 16 TO LESSON 18** afterwards. I learnt about *stereo geometry, epipolar geometry and stereo correspondence.* **RESULT:** The previous attempts can be optimised greatly as the cameras used for generating the test images have parallel planes as was told in the meeting to explain the task (I suppose). So the epipolar lines of the images will be horizontal lines and while searching for the best match I need not traverse the whole image, I can just move it in the horizontal direction. This will significantly save computational time.
5. So based on the previous results I just started moving the temporary kernel over the present row and not the whole right image. The complexity now reduces to O(n^3). The code now finds the absolute difference of each pixel of the kernel. It sums it up, finds the minima and for that minima it stores the perfect match. Then it prints the disparity map using the formula mentioned earlier. The kernel used here is 3x3 in size.
   OBSERVATIONS:

   i) (The generated disparity map is shown below.)

ii) For a perfect match the sum should come out to 0, but it didn't come to 0. This means that my earlier assumption that the epipolar lines will be horizontal is false.

iii) The computational time is large and took 10 to 15 seconds to generate the map after running the code. The reason is that the complexity is still large.

RESULTS: The disparity map generated is not good enough. I think that it depends on the kernel size. So a possible way to rectify it is to use a trackbar for it. The second problem is that the computational time is large. I presently don't have a solution to it.

LATEST ATTEMPT

I wrote the code for the 5th approach using trackbar now. The code compiles and runs successfully but due to large computational time the trackbar does not work and the code gets hanged. So I removed the trackbar function and started inputting the threshold values manually.

OBSERVATIONS:

(Disparity map using 5x5 kernel)

(Disparity map using 7x7 kernel)

So basically there is negligible difference between the outputs. The only difference I observed is that the computational time goes on increasing as I increase the kernel size.

RESULT: The present code gives good results but is computationally slow. Therefore some new optimised method needs to be found.