Perception Tasks - Aerial Robotics Kharagpur

General Instructions

The following tasks are intended to check your skills in the fields of Computer Vision and Path Planning. Feel free to ask any doubts in the WhatsApp group .

The final solution must be submitted in GitHub with documentation of each task. Please use the provided document Template in overleaf to create documentation/report (template.zip).

We had fun creating these tasks, and we hope that you'll enjoy them too. Do not copy code from friends, as we will use automated plagiarism checkers. Also, if you use any library functions, you might be asked to explain in detail how they work. Ping us for more problems if you don't find this challenging enough.

The final solution is not important; rather, your approach to the problem and your understanding matter.

1 Save Luna !

Luna is a robot. She has wheels to move in the environment. She has two cameras to see the environment. The problem is she doesn't know how to process the environment to maneuver. Your task is to teach Luna to see!

Luna is on the floor, and she sees several objects in front of her, but she is unsure whether to move forward or not. Save Luna from colliding with the objects. Since she has two identical cameras on her body, they are placed parallel at a horizontal distance. The image clicked by the left and right camera is given to you as left.png and right.png. Build a code (in Python preferably) that can generate a heatmap or colourmap of the environment with objects closer as red colour and objects farther as blue colour. The example is given below.

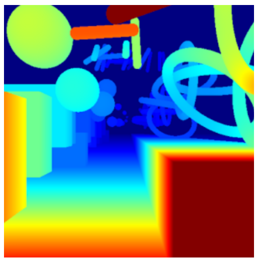
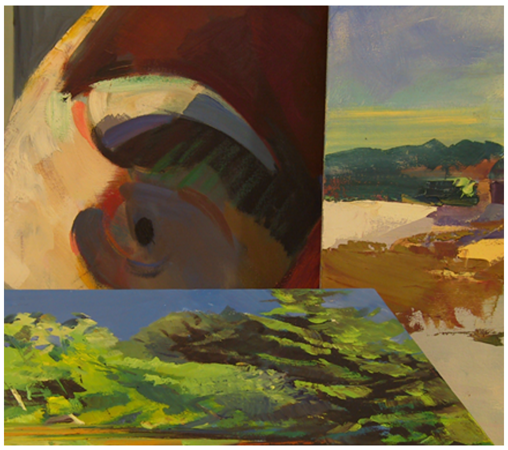
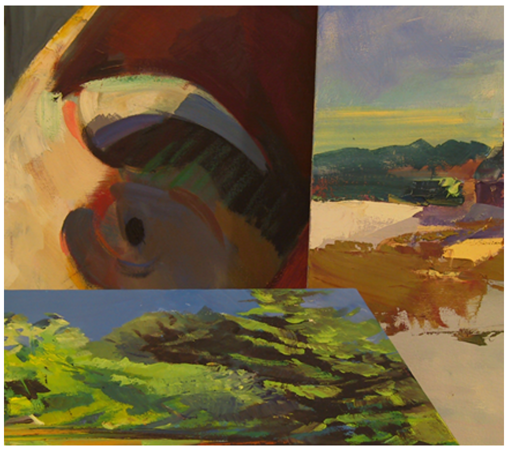


Figure 1: Example of Depth Map



(a) Image clicked by left camera



(b) Image clicked by right camera

Figure 2: Images clicked by two cameras

To create a depth map, you may measure the shift of the same elements in both images. Your main implementation must be from scratch, although you may use **basic** functions from any library you want. Explain the whole process clearly in the report. Save the final depth map depth.png with all the necessary code and checkpoints.

2 Gebe dich nie auf

Yeltsa Kcir was a big fan of π. To show his devotion towards π he hung this picture in his drawing room for his guests to see(look closely at the pixel values of each pixel beginning from top-left corner). However, one day Grant Sanderson, another self proclaimed π lover and jealous of the fame of Mr. Kcir as a π devotee, added some noise to the image(and also corrupted the link to the original image). The resulting pi_image.png is given below. The digits of π that were distorted can be each multiplied by 10*π and converted respectively to the greatest integer less than or equal to them. The resulting numbers can be arranged in descending row major order and used as a 2x2 filter.

Meanwhile in another part of the world Pablo Picasso came up with this artwork. However allegations were made that the art was not original and was a distorted copy of a famous portrait. You were called up to investigate into the matter and decide whether Mr. Picasso was guilty or innocent. From your highly reliable network of spies you have come to know that given Picasso's aptitude in Mathematics(or lack of it thereof!) he only knows to perform 3 operations: bitwise OR, bitwise AND, bitwise XOR. He has a habit of distorting pictures by applying a filter on pre-existing pictures. He does this by putting the filter on top left corner of the picture and performing element by element operation between the filter and picture values and updating the same value in the picture. then he moves the filter to the right by "s" units where s is equal to one of the dimensions of the filter(here 2). You already have the filter(find it!). All you have to do is apply the filter on the distorted image to recover the famous portrait. This portrait will serve as a template to the next part.

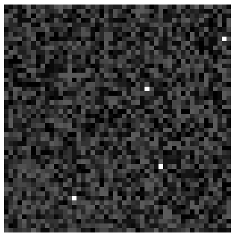


Figure 3: pi_image.png

Given below is an image(collage.png) which contains the template found above. The template you have found needs to be scaled to appropriate dimensions(100x100) to make it useful for template matching. apply template matching from scratch(without using inbuilt functions of opencv) and find the coordinates of the top left corner of the matched template in the image. Add the abscissa and ordinate, multiply it with pi and round it to the greatest integer less than or equal to it to get the password to the zip file.

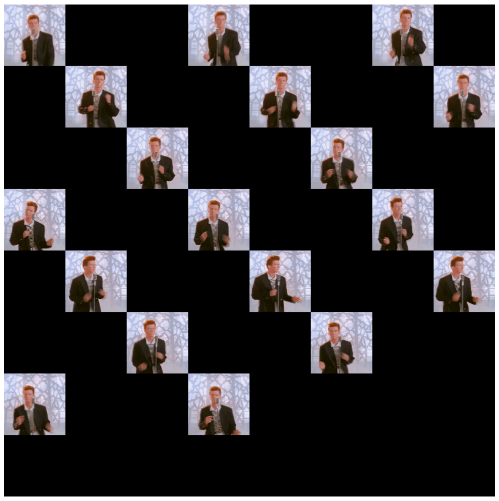


Figure 4: collage.png

In the images provided, perform the RRT-connect algorithm from start to end point and attach the image of the final path formed.

3 Implementing Path Planning with Probabilistic Roadmaps (PRM) (OPTIONAL TASK)

Motion planning is a crucial aspect of robotics, enabling robots to navigate complex environments while avoiding obstacles. This project aims to implement a path planning algorithm called the Probabilistic Roadmaps (PRM) for a two-dimensional (2D) environment.

PRM Algorithm:

The PRM algorithm constructs a roadmap within the workspace, representing feasible paths for the robot. The detailed implementation of the PRM algorithm can be found in the following reference paper:

- Kavraki, L. E., et al. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces.

Project Tasks:

1. **2D Image Map Implementation:**
   - Develop a Python program that implements the PRM algorithm for this 2D image map maze.png representing the environment. You need to run your algorithm for both `Start Easy` and `Start Hard`
   - The program should:
     - Read the image map provided and identify obstacles.
     - Implement the PRM algorithm to find a path between the start point and the end point.
     - Visualize the implementation on the original image map.
   - This initial implementation will serve as a foundation for the more complex environment.
2. **Pygame Environment Integration:**
   - Utilize the autonavsim2D library to create a more dynamic environment.
   - Adapt the PRM implementation to function within the Pygame environment provided by autonavsim2D
   - The program should:
     - Allow the user to define the robot's starting and goal configurations within the Pygame environment.
     - Utilize the PRM algorithm to find a collision-free path between the start and end points.
     - Visually represent the robot navigating the path within the Pygame environment.

Submit whatever you have implemented even if it is not fully working and also document the implementation and your understanding of the algorithm. Multi file and modular implementation and use of Object Oriented Programming is appreciated but not compulsory.

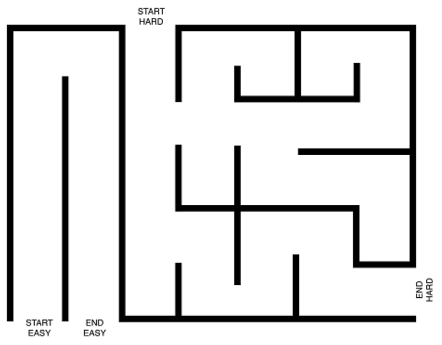


Figure 5: maze.png

End