Software Task Round

Selections - Round - Final

Congratulations on making it this far into the selections of the AGV Software Team!

For this final phase of selection, **five (5)** tasks are given. You must complete at least one. For each task, you first need to understand the problem statement, read concepts from the internet (some resources are provided in this doc itself), and then implement (code) it.

The task explanations are given at the end of this document in a separate section.

1. Make your programs Object Oriented if possible, although this is not a requirement.
2. The resources given in this document are not sufficient. You are expected to find more on your own.
3. You may discuss among yourselves and help each other, but sharing your code is a strict no. The use of plagiarized code will result in complete and immediate disqualification for both candidates, from whom the code/idea has been copied and who have copied it.
4. The test data and their format for all tasks will be shared with you separately.
5. For the tasks that require it, you can visualize the path or data using Matplotlib or OpenCV.
6. Don't hesitate to contact any of us in case of any obstacles.
7. We have allotted the time with sufficient consideration for various factors and hence cannot give any extension to anyone as it would be extremely unfair to the rest of the candidates

A word of advice- You are expected to do at least one task, but you are encouraged to do more - this will increase your chances of getting selected and will also help you explore your interests in different fields we work on and get a general idea about them.

1. Bringing it all together

Introduction

While the task that follows below does not directly relate to autonomous vehicles, a lot of the ideas and concepts that you will use to solve it will directly carry over to autonomous vehicles and automation in general. This includes but is not limited to planning in continuous spaces, designing controllers to carry out your planned trajectory, and using simulators and integration so that you can test out your algorithms before deploying them in the real world.

For this task, you will be using ViZDoom which is an AI research platform/simulator based on the game which arguably birthed the FPS genre - Doom. You can visit their github repo here. Follow the instructions there to set it up on your machine (which is just a simple pip install for the most part) and go through the documentation and examples on how to use it. The overall goal of this task is to navigate a maze and reach a checkpoint.

Level 1

Load this custom .wad file into the simulator. A WAD file is a game data file used by Doom and Doom II, as well as other first-person shooter games that use the original Doom engine. On correctly loading the .wad file you will see something like this.

Global Planning

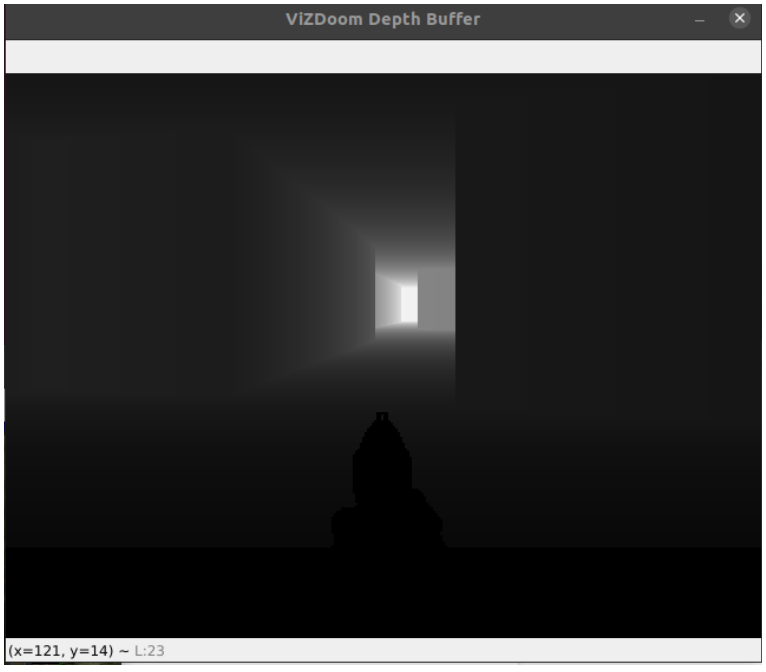
You can either use the automap from the simulator or use the map directly from here. The white pixel is the initial position of your ego and blue is where you will find the blue skull needed to finish the level. You can use any planning algorithm you like to plan a trajectory but we prefer RRT*. You will need to keep an eye out for dynamic and kinetic constraints of your ego since any path connecting the two points might not be feasible for your controller to follow in the next level.

Trajectory Following

You will need to now translate your global trajectory to actions in space and direct your ego to the blue skull. You might need a closed loop controller that corrects the errors that can build up in following the trajectory.

Level 2

You will no longer have access to the automap or a predefined map. You are only allowed to use the data from the buffers. The objective remains the same. You will most likely need to implement some sort of DFS search in the space while keeping approximate track of where you are, controls for backtracking and behaviors to search for open passages; however you are free to take any approach you like. An example depth buffer is given below.



Submission

To make your submission for this task, use this modified version of the ViZDoom repository. The examples folder contains two python files, for Level 1 and Level 2 of this task. You can add your code in these files and submit them. We have already set up the .wad file of the custom map in this repository and made changes in the configuration files accordingly.
Feel free to run and change the existing scripts in the examples folder to get a better understanding of the environment.

In addition to the mentioned codes, submit an image of the final global path that your designed algorithm suggested for the given map image. Submit a screen recording of the next parts of the task.

2. Image Stitching

Introduction

Image stitching is a very important problem in robotic vision. You may have seen 360° camera feeds deployed in autonomous vehicles like this one:

Suppose we have a robot fitted with multiple cameras. We would like to avoid

1. Reusing redundant information that is shared across, and
2. Discarding important information that is split across multiple views.

To tackle this, we generate a composite image, and process that instead. This way, all components of our system coherently perceive our surroundings. We therefore get a holistic view of the world using all camera angles.

Random sample consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers when outliers are to be accorded no influence on the values of the estimates.

In **Image Stitching**, RANSAC is used to estimate the **Homography matrix**. It turns out that the Homography is very sensitive to the quality of data we pass to it. Hence, it is important to have an algorithm (RANSAC) that can filter points that clearly belong to the data distribution from the ones which do not.

Learning Task (10%)

Here, we present to you this video which you can go through to learn about RANSAC and its applications. You may read or find other materials online but this is an excellent place to start if you do not know where to refer.

Once you are done with the learning you are all set to grab the “real’ tasks and put your knowledge to the test. Here is the list of tasks you need to perform after going through the above resources.

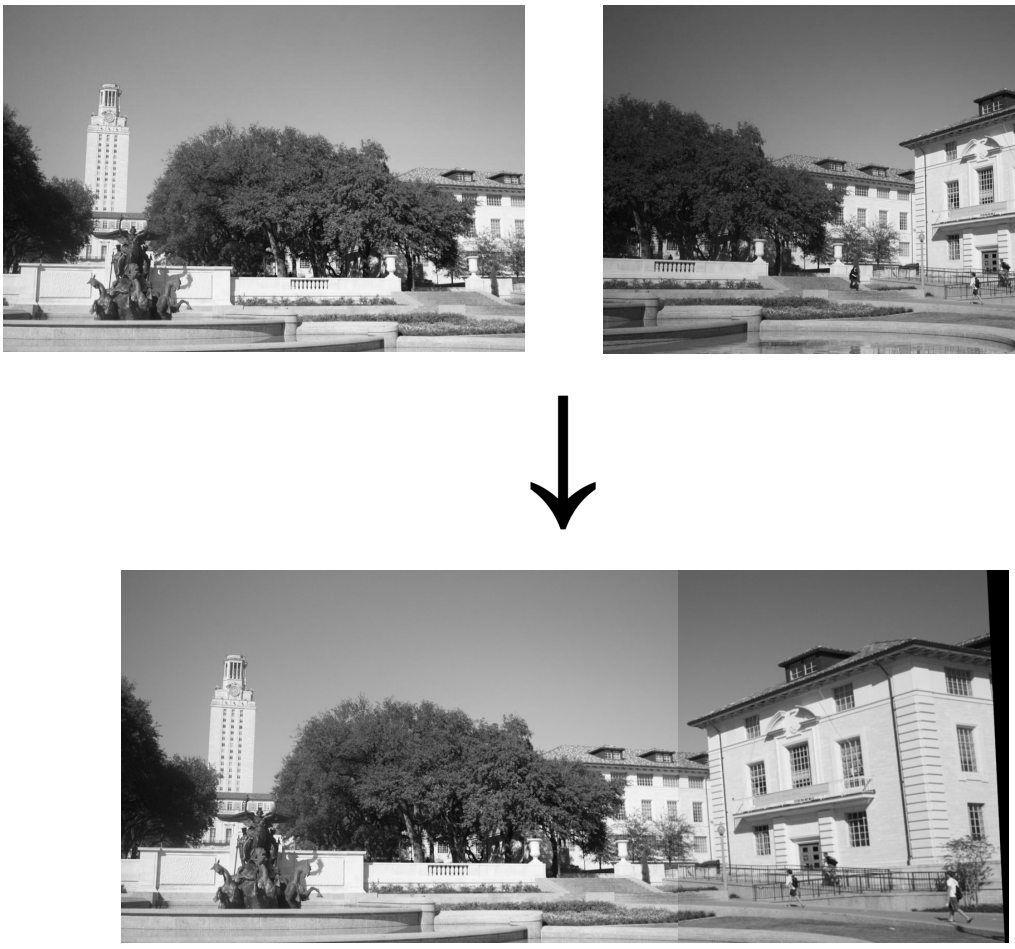
2D Line Fitting Task (30%)

You will be given an image, where you have to detect the input points (all the black circles are your input points with the center of the circle being considered as the coordinates of the point) and your task is to find the best fit line passing through the set of (of course you need to take care of outliers

using RANSAC ;p). Also, showing some effort in visualizing and explaining why the idea works well with this dataset would certainly fetch you some extra points.

Image Stitching Task (60%)

Now, you will be seeing some real applications of RANSAC. Here, we go. Your task is to combine two photographic images with overlapping fields of view to produce a **segmented panorama**, the way it is shown here.



a. Learning Subpart (20%)

For this task, you will certainly need some additional Computer Vision concepts that you have to learn yourself. Here, is Introduction to Computer Vision course which is provided for your reference. [**NOTE** you do not need to go through the entire course but find concepts relevant to your work. ;)]

b. Coding Subpart (40%)

After completion of the above course lectures, implement **feature matching** using the **built-in** functions of OpenCV. For testing your code, you can use these images, or you can even use your own images. Finally, stitch two images together by finding the **homography** between them using **feature matching** and **RANSAC**.

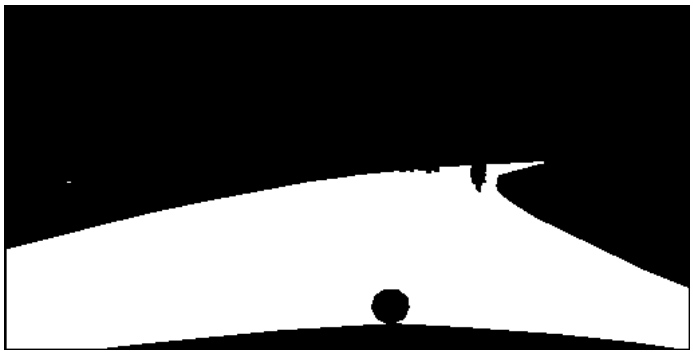
3. Lane detection

Introduction

One of the most fundamental tasks in autonomous navigation is detecting driveable area. However, when you only have 2D camera information, the problem of lane detection becomes much more challenging. Making accurate predictions in the shortest time possible is an intriguing problem with a rapidly improving state-of-the-art even today. This is a problem of semantic segmentation.

Semantic Segmentation Task (60%)

In this task, you are provided 23 images. You must classify each pixel in these images as belonging to the road or not, and therefore generate a segmentation map. A sample image and its corresponding segmentation map is shown below:

It's not necessary you get ideal results on these particular images. Instead, we want you to think of innovative methods that would work across various images in different conditions. You may even design a new algorithm that no one has thought of before!
(P.S. **Do NOT use the provided segmented maps to generate yours! They are merely for reference.**)

The images are provided along with sample outputs at this Link.

Additional tasks

1. **Intersection over Union - Self Evaluation (10%)**

   To judge your accuracy, use Intersection over Union (IoU). Calculate the mean IoU over your predictions with the segmented maps provided here.

Bonus - Read about precision and recall. Calculate mean precision and recall values over all the predicted maps obtained.

2. Perspective Transform - A Birds Eye View (20%)

Perspective transformation is used to align images properly. Read about birds eye view and apply perspective transform to convert these images into bird's eye view using the segmentation maps and the original images.

After obtaining the warped image, think of ways you could improve your lane detection algorithm using it. It is much easier to visualize the edges of the road from a birds-eye view compared to a camera's perspective. After detecting lanes, try to find the position of the ego vehicle from the center of the lane. Also, calculate the radius of curvature of the path the vehicle is following at the instance. Hint: Warped images might help.

3. Gamma Correction - Undoing Some Good Work (10%)

Gamma correction is a technique used for image enhancement (an important step for segmenting images in adverse conditions).

For each image provided, the image obtained after implementing gamma correction has been provided at the link and link. Calculate the gamma values and convert the transformed images into the original ones.

Bonus - Read about adaptive gamma correction and histogram equalization.

Deliverables

Your deliverables for this task are the following:

1. **Semantic Segmentation**: 23 segmentation maps, which are binary images in `.png` format. A pixel being bright indicates it corresponds to a road pixel in the original image.
2. **Self Evaluation**: A `.txt` text file. The last three lines of this file must mean IoU, mean precision, and mean recall. If you do not attempt the bonus task, the last two lines must be `inf`.
3. **A Birds Eye View**: An output image in `.png` format for each of the 23 input images and a `.txt` text file for position of the vehicle and radius of curvature of the road for each of the input images.
4. **Undoing Some Good Work**: A `.txt` text file containing the values of gamma obtained. 46 segmentation maps for each of the 46 (23 gamma corrected + 23 transformed) images.

4. Image Dehazing

Introduction

Autonomous vehicles deployed in the real-world often encounter adverse conditions. For example, small particles (dust, smoke, water drops) suspended in the air scatter light approaching the camera, leading to a degradation of image quality. This phenomenon is called haze, and the corruption of visual information leads to poor image processing results, making perception difficult. In this task, you are to develop a method to bring clarity to hazy images.

Real-Time Dehazing (40%)

You are provided a set of 16 hazy images at this link. You must efficiently process them to remove the haze. There are two things to keep in mind:

PSNR and SSIM - Clarity (30%)

The images you produce must be clear, dehazed versions of the original. Read up on the metrics Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM). To determine your method's performance, calculate the PSNR and SSIM between your output and the ground truth given here.

Video Dehazing - Speed (30%)

If your method is computationally expensive, it is not feasible to be used on a real-time system. The need for speed is especially important here as this is merely an image enhancement step. More important and heavier tasks like segmentation, feature extraction, etc. are to be performed further down the pipeline, so we should avoid wasting too much of our limited on-board resources on dehazing. There is a hazy video provided at this link. You must dehaze each frame of the video and write the framerate on the output frame.

Deliverables

1. **Clarity**: You must submit 16 dehazed images in `.png` format. If the filename of an input image is `k_outdoor_hazy.png`, your output filename should include `k` (e.g. `k_outdoor_dh.png`) so it is easier to correlate. You must also submit a `.csv` formatted file where each line contains three comma separated values in the order `k,psnr,ssim` where `k` is the input image index, `psnr` and `ssim` are its corresponding PSNR and SSIM values.

2. **Speed**: You must submit the dehazed video in `.mp4` format with the following parameters:

   `filename:` `dehazed.mp4`

   `fourcc:` `mp4v`

   `fps:` `30`

   `framesize:` `(1920,1080)`

   The frame processing rate in fps must be printed on each frame.

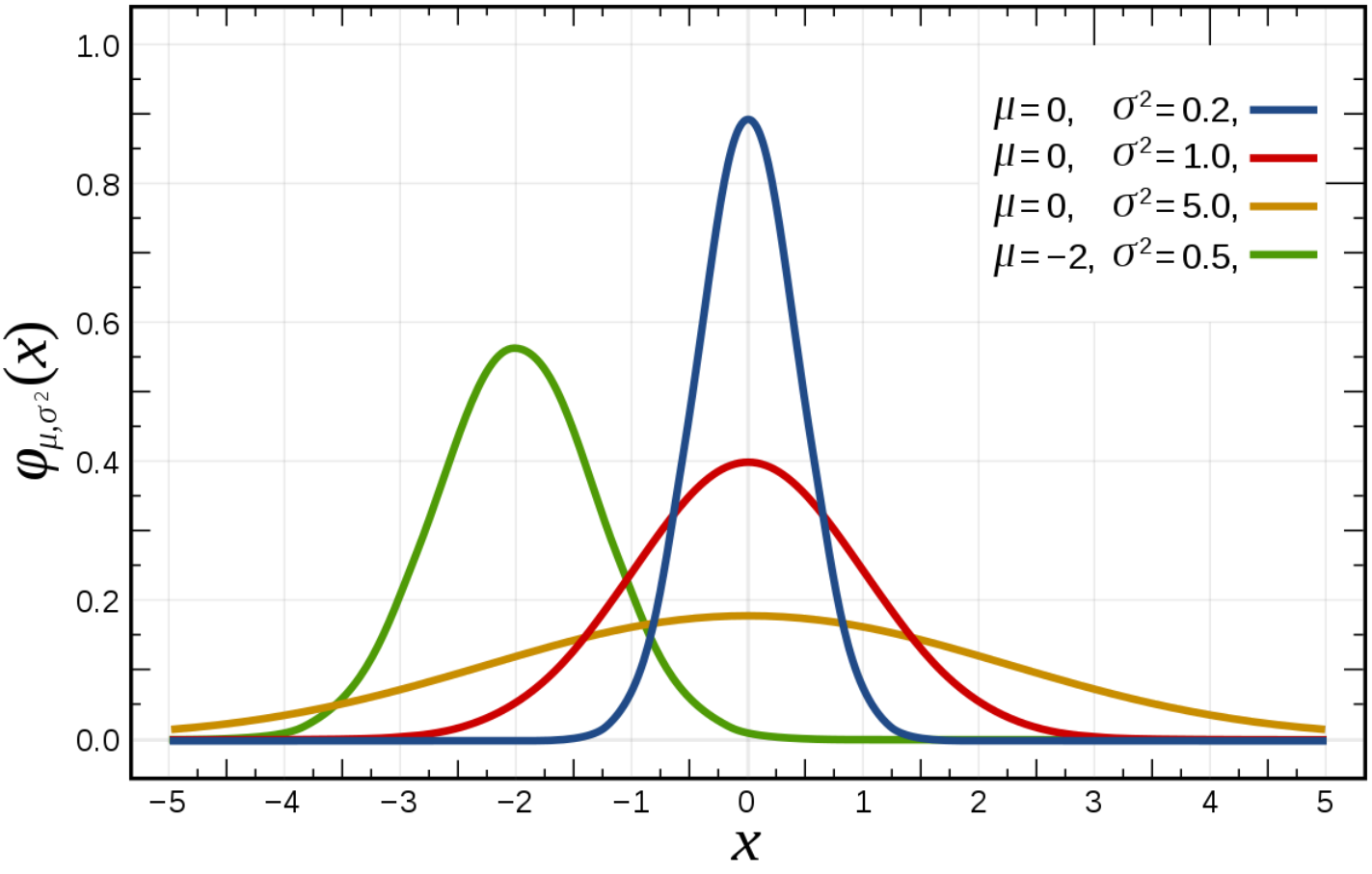
3. **Real-Time Dehazing**: Code, brief documentation thereof, and an explanation of your approach.

5. Kalman Filter

Introduction

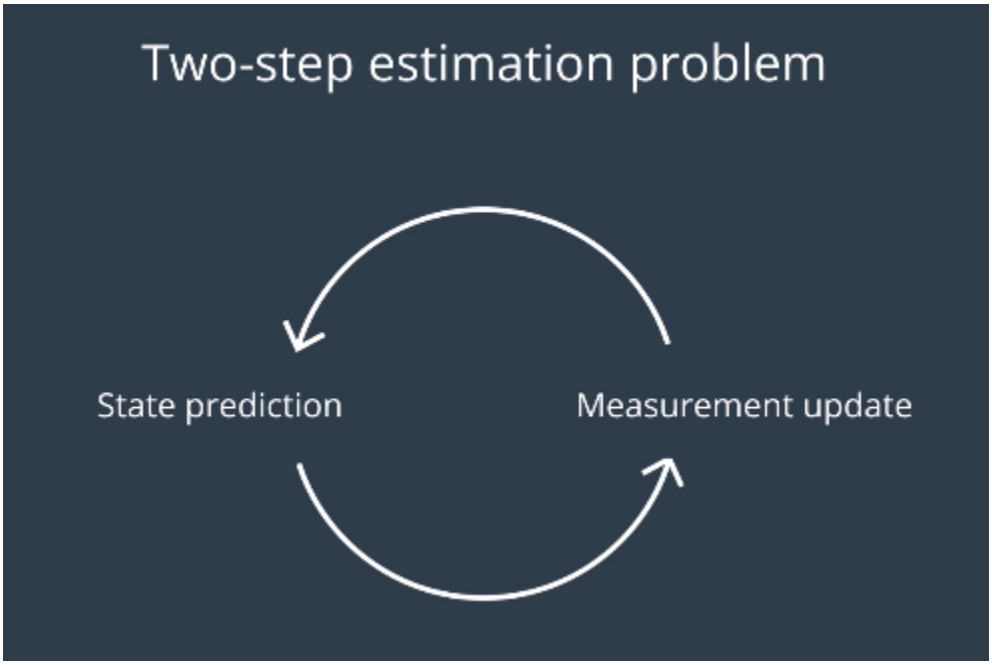
For a self-driving car, it is very important to know where you are and how fast you are moving at all times with high precision. So what we need for this is just your average motion equations and also measurement data. The motion and measurement steps occur iteratively. First, we get a state estimate with the help of our motion model and then we tally that to the measured data.

But wait which one to believe? Which one has more error - Our motion equations or measurement data? Hence to tackle this exact problem we use a very fundamental mathematical function and its properties - **The Gaussian**.



We consider each of our steps as an operation on Gaussians.

To do this task more effectively we use a mathematical technique known as the Kalman Filter. What What Kalman Filter does is that it takes into account both the state estimate and the measurement estimate and gives out a very accurate estimate combining the above two estimates.



Read about the Kalman filter and its various Matrices. Apply the motion update step and the measurement update step. And then print out the position, velocity, and uncertainty matrix at each step.

Problem Statement:

1. You have to find the accurate position of your bot. You have data from two sensors - the GPS coordinates, and the bot's average velocity, given in a file. You have to read each state successively, but only after you have processed the previous one.
2. You need to implement a Kalman filter for a self-driving car traveling in a 2D world.
3. All the required parameters are to be read from a .txt file: link
4. At each step process the data by applying the Kalman Filter, and print the updated positions and their uncertainty.
5. Visualize the processed points using OpenCV or matplotlib.
6. Read about the Extended Kalman Filter (EKF) and Unscented Kalman Filter. (Optional)

Resources:- PROBABILISTIC ROBOTICS

Hint:- You can use the Eigen library for matrix multiplication in C++ and numpy for Python.

Explanation of Input Data: (link)

The format goes something like this:

Line 1: Initial pos x, Initial posy
From Line 2: Pos x, Pos y, vel x, vel y

You can open the file and have a look too. The velx, vely is the average velocity between the previous and current state.

As for the covariances, we have decided to leave it up to you. Well, it goes without saying that you can't choose some random values. The values should go hand in hand with the data such that you end up as close to where you started (Looks like life to me xD).

Feel free to contact any one of us regarding any problem you face. Here is the contact information of all the 2nd years in the Software Team.

Contact Information

1. Pranav Kulkarni	9420256993	pranavkulkarni610@gmail.com
2. Himadri Bhakta	6290612390	himadribhakta@gmail.com
3. Roopsa Sen	9433075482	roopsa.sen@gmail.com
4. Rohit Ranjan	9934946872	ranjanmail.rohit@gmail.com
5. Kaushal Jadhav	9284735270	kaushaltjadhav@gmail.com
6. Shashwat Naidu	7207820759	shashwatnaidu07@gmail.com
7. Mradul Agrawal	9461128404	mradulag20@gmail.com
8. Monish Natarajan	9632743801	monish.natarajan@gmail.com
9. Aayush Jain	9820621210	aayushjain1098@gmail.com
10. Avi Amalanshu	9953769542	avi_amalanshu@engineer.com
11. Shreya Bhatt	9886789976	emailtoshreya@gmail.com
12. Pradipto Mondal	9635762296	pravsman111@gmail.com
13. Mohhit Kumar Jha	8513870876	mohhitkumarjha@gmail.com
14. Saumyadip Nandy	8700062801	bonny.tffs@gmail.com
15. Sidharth Sinha	9518953231	sidharths147@gmail.com