Inter-IIT Documentation

Team: 2

1 Introduction

For solving the Problem Statement, we divided ourselves into different subsystems- namely UGV Control, Summer Mapping, Winter Following Feedback. The Winter Following Feedback can be sub divided as Decision making and UGV Detection.

UGV Control had to develop a robust control system to maneuver the UGV on the desired central path of the road. In this, a bicycle model based trajectory-velocity planner is used to define velocity setpoints and a Stanley controller is used to perform steering.

To perform summer mapping, the drone was commanded to move across the road, follow the edge of the road and detect the boundaries. The code is intended to store edge points and stitch the map to get an array of the mid-points of the path. Now this array is then used by the UGV Control team to maneuver the car.

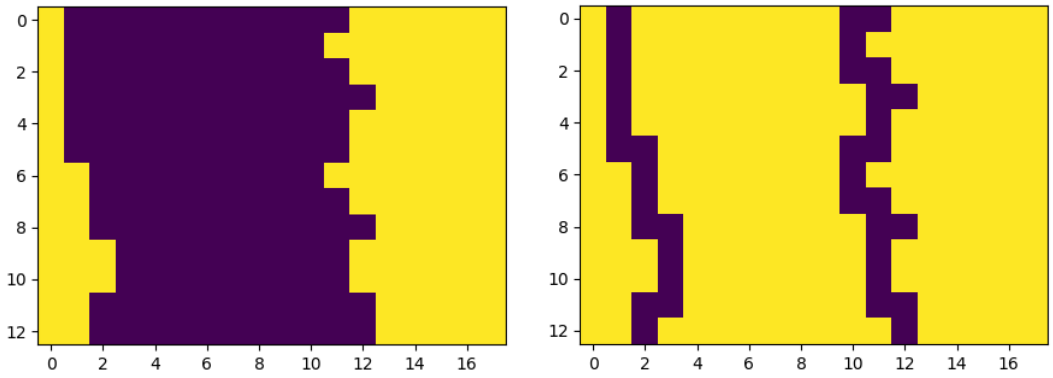
To get feedback of position and other dynamics, we are flying the UAV over the UGV, which gives the dynamics of the car and which in turn is used to accurately complete the task. This is done by the Winter Following feedback team. They are using a PD controller and passing setpoints to maintain drone over the UGV and also maintain the UGV in the field of view of the UAV. To ensure this, the controller is tuned to avoid mountains and carry on with sharp turns.

The UGV Detection team first implemented YOLO algorithm and contour detection to correctly detect the orientation of the car and get its position and velocity.

2 Mapping

Figure 1: RGB image of road from camera

We make use of the Depth data to do the mapping in all three worlds. The RGB data is pretty useless in this task due to the road and background looking super similar and segmentation tasks wont give satisfactory results.



(a) Road segmentation from depth data

(b) Edge detection from segmented data

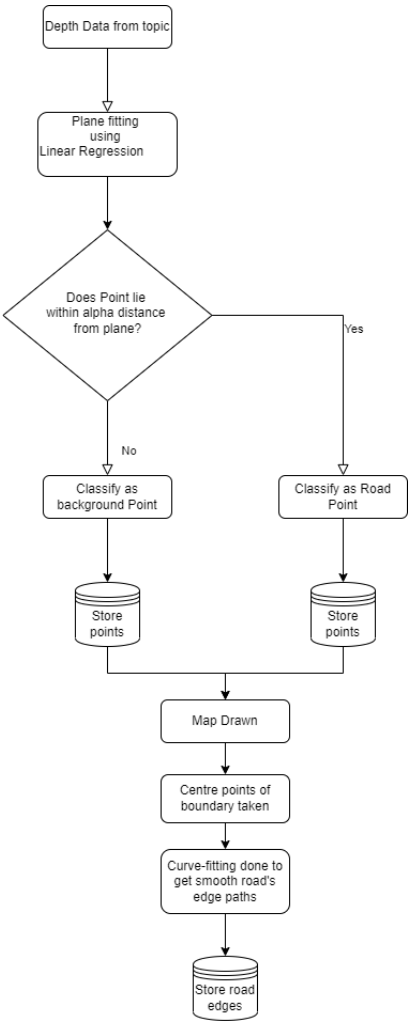


Figure 3: Workflow

Flow of the Algorithm:

- First, we employ linear regression to fit a plane on the data received and find the plane of the road(The drone is positioned such that it is above the road at all time during mapping and thus most points of the depth data lie on one plane, the road's plane)

- Once finding the plane, we try to filter out the background portions of the terrain by filtering them with a threshold and we estimate height of drone from road by analysing the the corresponding depth data.
- Then, a map is drawn to map out the boundaries of the road. The centres of the pixels of the boundaries are taken to be the edges of the road, we estimate next waypoint and yaw for the drone by analysing the edge of road.
- After this the points are stitched/fitted with a smooth curve to get a path.

3 Control of the Car

3.1 The Control Problem

We start by formulating the control problem. Given a desired trajectory S which consists of the points to be followed along the road, we need control inputs $a(t)$ and $\delta(t)$ so that the resulting trajectory minimizes the error between the tracked trajectory of the car and the desired trajectory S.

The car control consists of two components: longitudinal control (to control the acceleration) and the lateral control (to control the steering) of the car. Our implementation employs P control law to control the acceleration of the car and uses the Stanley control law to control the steering of the car.

3.2 Trajectory Generation

After summer mapping we will have a set of points P along the road that we want the UGV to follow. From this set of points we generated a trajectory that specifies the desired values of x and y coordinates and the desired yaw ψ along the trajectory. To generate the trajectory, we have used cubic spline interpolation on the set PP. Each segment in P is approximated as a cubic polynomial and then the coefficients are computed using continuity and differentiability properties of the polynomial.

3.3 Generating the desired velocity to be tracked

Since the desired trajectory S is generated during the summer mapping, we have computed the radius of curvature of the points along the trajectory. Once we have the radius of curvature of the points every turn can be approximated as a local circular motion and hence the following equation holds:

$$a = \frac{v^2}{R} \implies v = \sqrt{aR}$$

Where a is the centripetal acceleration and R is the radius of curvature of the trajectory at the point of analysis. We have tuned this equation to achieve a better performance using the following conditions:

- If Radius of Curvature is less than 15 m: then the desired velocity is taken to be-

$$v_{desired} = \sqrt{aR^{(R-12)}}$$

- Else if the radius of the car is greater than 15 m: the desired velocity is taken to be-

$$v_{desired} = \sqrt{aR^{1.1}}$$

The first condition ensures that for sharp turns ($R < 12m$), velocity decreases so as to ensure than there is no overshoot during the trajectory tracking. The second condition makes sure that for straight roads (R is large), the velocity is as large as we want it to be.

3.4 Velocity Control

Once we know the desired velocity along the trajectory S , we can implement the following algorithm:
For all points p in the trajectory S,

$$a(p) = K_p(v_{desired}(p) - v_{actual}(p)$$

Where $v_{actual}(p)$ is the actual velocity of the car which is calculated by the car detection subsystem, and K_p (Proportional Gain) is a positive constant.

3.5 Steer Control

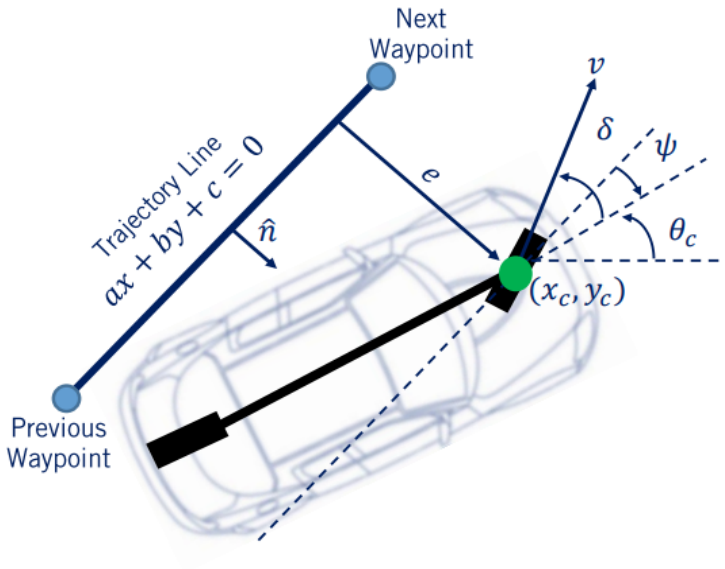


Figure 4: Stanley Controller Model

To control the steer of the car we have implemented Stanley Controller. Since we have the desired yaw at all points along the trajectory and we are also getting the center (x_c, y_c) and the yaw (ψ) of the car from the detection subsystem, we can compute the coordinates of the front axle (x_f, y_f) using the parametric equation:

$$x_f = x_c + l_f \cos(\psi)$$

$$y_f = y_c + l_f \sin(\psi)$$

Here the distance between the center of the UGV and the centre of the front axle (l_f) is taken from the URDF of the Prius model.
Now that we have all the components required to formulate the Stanley Control law, we can compute the required errors, namely the heading error (e_h) and the cross-track error (e_c).

$$e_h = \psi_{desired} - \psi_{actual}$$

The cross track error e_c is the distance between the front axle and the point nearest to the front axle in the trajectory. Using these errors we can compute the Stanley control law at any point p along the trajectory S as follows:

$$\delta(p) = e_h + \tan^{-1}\left(\frac{K_e e_c}{K_s + v}\right)$$

Where v is the velocity of the car which we are getting from the detection subsystem, and K_e (Stanley Gain) and K_p (Softness Constant) are positive constants. The Stanley Control law has been tuned for better performance in our implementation :

- If velocity of the car is less than 13 m/s then-

$$\delta(p) = 6e_h + 8\tan^{-1}\left(\frac{K_e e_c}{K_s + v}\right)$$

- Else if the velocity of the car is greater than 15 m/s then-

$$\delta(p) = 2e_h + 6\tan^{-1}\left(\frac{K_e e_c}{K_s + v}\right)$$

4 Decision Making of the drone

During the winter tracking course, we need to ensure that the drone always remains above the car so so as to always keep receiving the feedback of the car. To do so we have implemented a PD controller.
Consider the position of the UGV in the frame of the camera and define:
$e_{position}$ = position$_{UGV}$ - position$_{UAV}$
e_{vel} =velocity$_{UGV}$ - velocity$_{UAV}$
e_{yaw} =yaw$_{UGV}$ - yaw$_{UAV}$
Then the commanded velocity to the drone is computed as following:

$$\mathbf{v} = K_1(e_{position}) + K_2(e_{vel})$$

And

$$\omega = K_3(e_{yaw})$$

Where $\mathbf{v}$ is the velocity vector of the drone. And ω is the angular velocity of the drone along the z axis.

5 Detection of the UGV

5.1 The Detection Problem

The major problems faced while detection is the detection of the UGV in the noisy and environment and the detection of the orientation of the car from the drone's perspective. The position and orientation of the car is required in the control side of the problem statement by the Stanley Controller.

We use a combination of different algorithms and efficient machine learning models to detect the UGV and its orientation from the drone's point of view.

5.2 Methods Explored

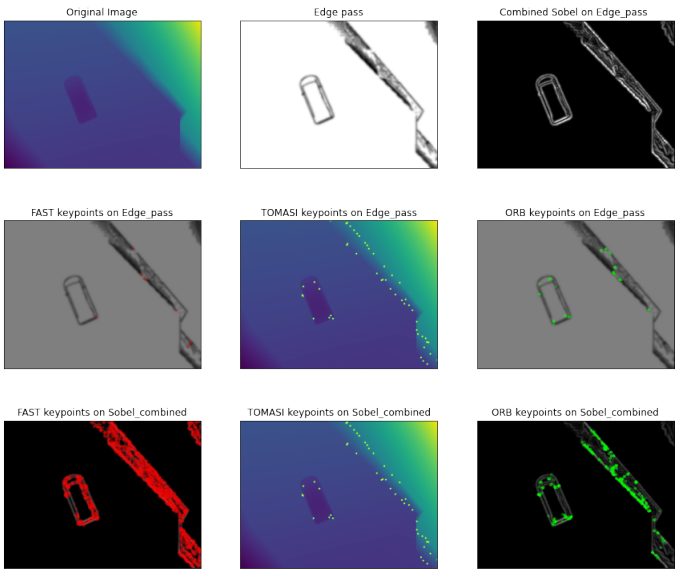
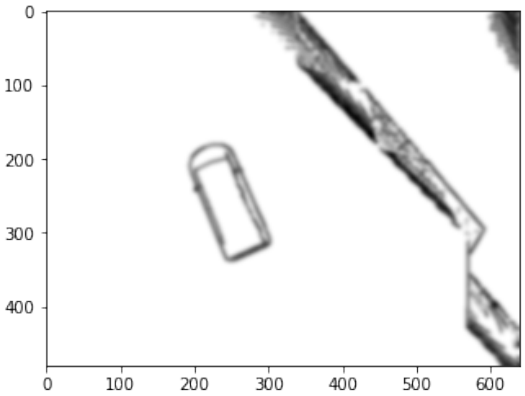


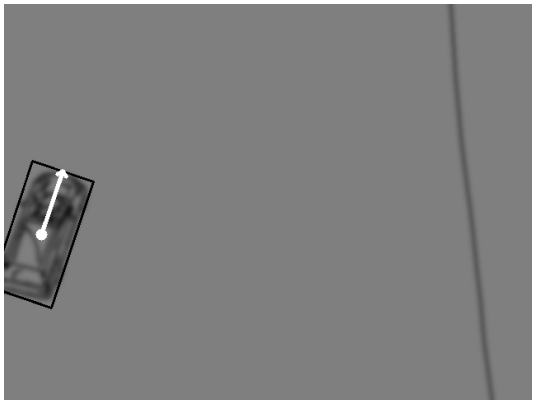
Figure 5: Methods tried for Road Detection

We used Canny filters/Sobel filters to get an outline of Car and road edges from the depth data. Then we employed a keypoint/feature detector like ORB or FAST to find keypoints. We then used] the detection/bbox from YOLO-model to suppress the points inside the car (keypoints of car). The rest of the points are the road's edge points which can be stored and then curve-fitted.
But, the above method turned out to be infeasible due to the uncertainity of identifying edge points on the road edges, and when number of points are crowded, line fitting consequently would be much harder.

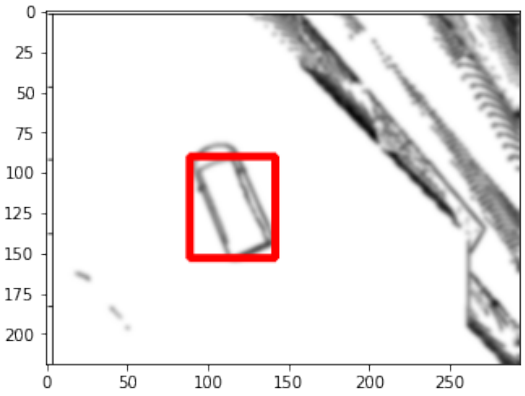
(a) Edge Detection on RGB data



(b) Edge Detection on Depth data



(a) Angle finding using Contour Detection on RGB Edge image



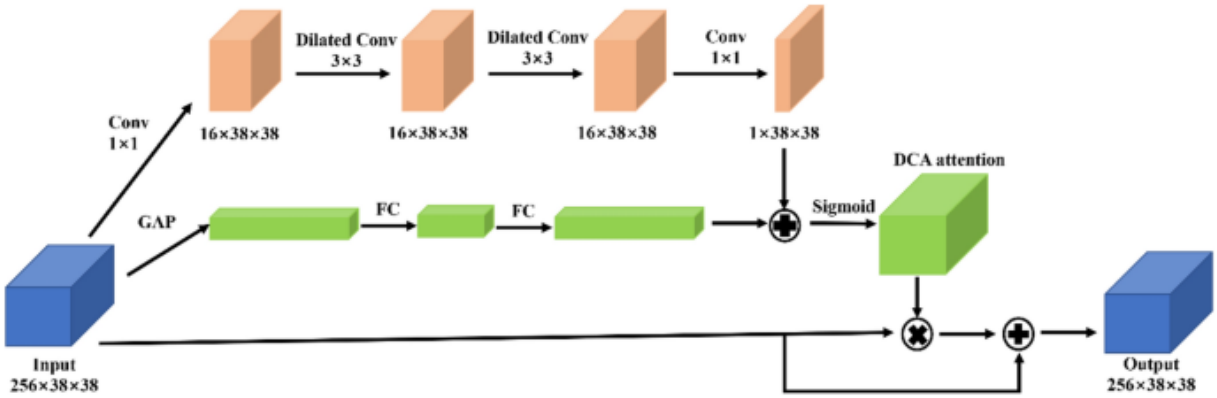
(b) YOLO-detection on Edge-image

5.3 Detection

Detection of the UGV has at first been done by state-of-the-art YOLO-v4-tiny model to reduce computational costs all while maintaining a decent accuracy of the detected bounding boxes.

The following computation has been done on depth data instead of conventionally using RGB data. The algorithm followed is:

- Initializing YOLO-v4-tiny model using trained weights and configuration file
- Converting subscribed data from ROS topic to a numpy array using CvBridge
- Using Canny edge detection to remove noisy data by optimum choice of aperture size
- Reducing noise even further by adding Gaussian Blur with variance = 3
- Detecting car by passing in the final image after pre-processing to get an accurate bounding box
- Applying adaptive threshold to remove noise and obtain accurate contours in the next step
- Applying contour detection using OpenCV to obtain outline of the car.
- Sorting the contours in descending order using area to obtain the outermost contour of the car
- Using OpenCV to find the minimum area box that encompasses the outermost contour and finding the angle at which the box is rotated



YOLO-v4-tiny architecture

5.4 Technical Details

YOLO-v4-tiny is a state-of-the-art model that has been developed to reduce computational power required for smaller form factor systems while maintaining a relatively high accuracy.

Our model has been trained using Darknet repository for 1500 epochs.

Canny Edge Detection has been used to find the edges in the image. This cleans up certain part of the noise and allows for better results on our machine learning model.

Before moving onto detecting the car in the images, we apply Gaussian Blurring on the image to smoothen out edges using the built-in OpenCV function

After applying our model, we crop out that part of the image and add some padding to allow for some errors. We apply adaptive thresholding on this image which is a powerful technique as it takes out manual tuning of threshold values and detects threshold values by itself. We finally apply the contour detection and then get the orientation of the car using OpenCV functions.

5.5 Further Improvements

While optimising our pipeline, we found a method that circumvented the use of YOLO-v4-tiny reducing the computation required at each frame by 60%. This algorithm involved the use of RGB data instead of depth data. The main reason for this is that the depth data obtained in very noisy while during winter the RGB data is extremely clean and at some points, only the car is visible.

The refined algorithm involved using Canny Edge detection and then Gaussian Blurring to clean out the image for contour detection. The contours were extracted and a minAreaRect function from OpenCV used to calculate the orientation. This improved results drastically while reducing our computational power by over 70%.

6 References

1. Improved YOLOv4-tiny network for real-time electronic component detection
2. Stanley Control
3. Stanley Control Image