UNMESH MASHRUWALA
Innovation Cell
IIT BOMBAY

STPC 2021-22 Phase 3 Proposal

A report of the technical updates from earlier phases and budget breakdown for Phase 3

STPC 2021-22 Phase 3 Proposal **1**

1. Subsystem - Localization: **3**
Task 1 - EKF Testing: 3
Task 2 - Lego-Loam Bor 4
Task 3 - Lio-SLAM 5
Task 4 - Google Cartographer 5
Task 5 - ORB-SLAM 2: 6

2. Subsystem - Motion Planning **7**
Task 1 - Reeds Shepp Planner 7
Task 2 - Road Segmentation 8
Task 3 - Frenet Frame Approach 9
Task 4 - Hybrid A* 10
Task 5 - MPC Planner 11

3. Subsystem - Controls **12**
Task 1 - Control_node.py: 12
Task 2 - Pure Pursuit Model 13
Task 3 - Path and Velocity Discretization model: 13
Task 4 - C++ Implementation of MPC: 14
Task 5 - Explored C++ with Casadi to implement model predictive controller: 15
Task 6 - Discrete_Controller.py 16

4. Subsystem - Decision Making **18**

5. Subsystem - Computer Vision **19**
Object Detection 19
Task 1 - Mannequin Detection 19
Task 2 - Stop Sign Detection 20
Task 3 - Lidar Camera Calibration 20
Task 4 - Lane Detection 21

6. Subsystem - Mechatronics **22**
Task 1 - Velodyne Lidar mount 22
Task 2 - CAN module PCB design and printing 23
Task 3 - Brake by wire 24
Task 5 - Steer By Wire: 26

Phase 2 Work Update

Technical Progress

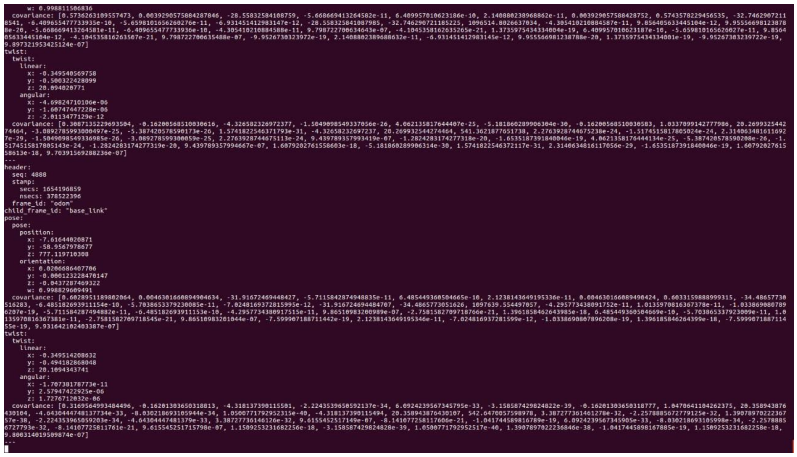
1. Subsystem - Localization:

Task 1 - EKF Testing:

- We needed an algorithm to determine the state vector of our vehicle, much required for the Controls and motion planning subsystem through multiple sensor fusion; we found EKF as one of these implementable algorithms.

- EKF was earlier tested on software using gazebo world, in which we simulated noise data of Odom, GPS, and IMU type and successfully implemented it using the robot_localization package.
- After that, we started hardware testing using GPS and IMU in our inventory. First, we configured them to run individually. We fused them using EKF and tested them on SeDriCa. A bag file data was created for around 700s with different orientations and variations.
- After the testing, we observed that a lot of covariance still hinders our location's accuracy. This imu_utilis package was explored to determine the imu noise so we can calibrate our EKF parameters accordingly.

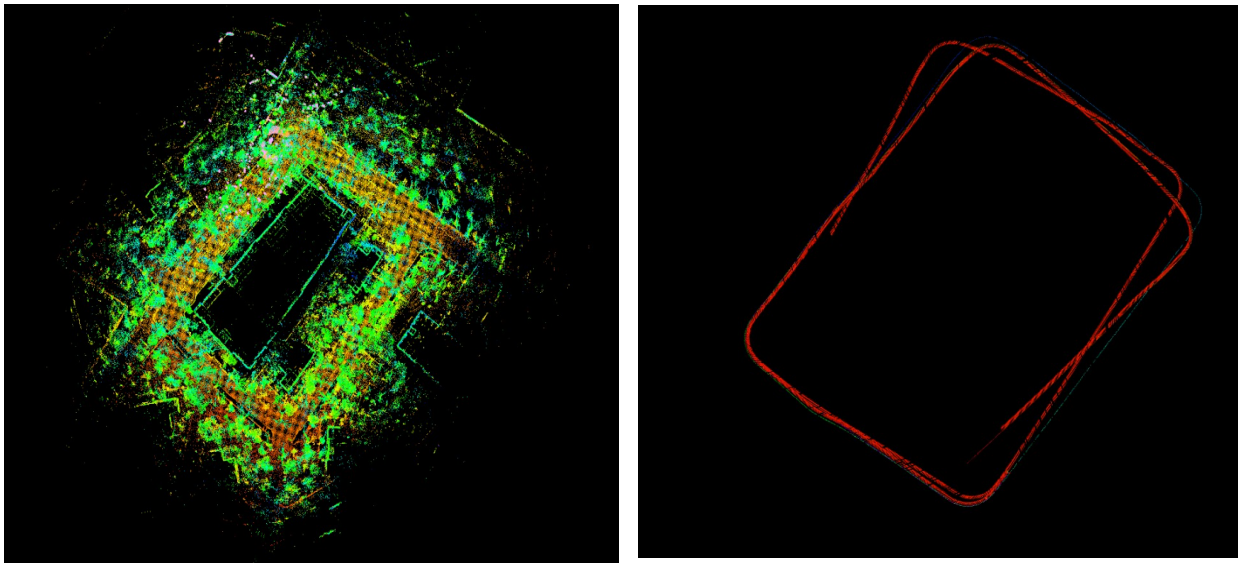
Results Observed:



- The covariance values are low. Hence we conclude that the results are decent. Still, we need to improve the results by fine-tuning the parameters in the .yaml file and using an alternative source of odometry data through wheel encoders.

Task 2 - Lego-Loam Bor

- We tried and tested several sophisticated SLAM algorithms for relocalization and mapping purpose
- LeGO-LOAM-BOR is a lightweight, ground-optimized lidar odometry and mapping (LeGO-LOAM) system for ROS-compatible unmanned ground vehicles (UGVs). This package is beneficial for testing bagfiles as it can process bagfiles at an improved speed of order 5x to 10x.
- We tested this package on bagfiles with lidar data from VLP-16 and OS1-64 lidars. We used point cloud segmentation and feature extraction to extract separate planar and edge features to filter out noise.
- We then used the features in a two-step Levenberg-Marquardt optimization approach to solve distinct components of the six-degree-of-freedom transformation across successive scans.
- The algorithm extracted the features quite well. But the loop closure needed to be more satisfactory. And for that, we experimented with the other SLAM algorithms.

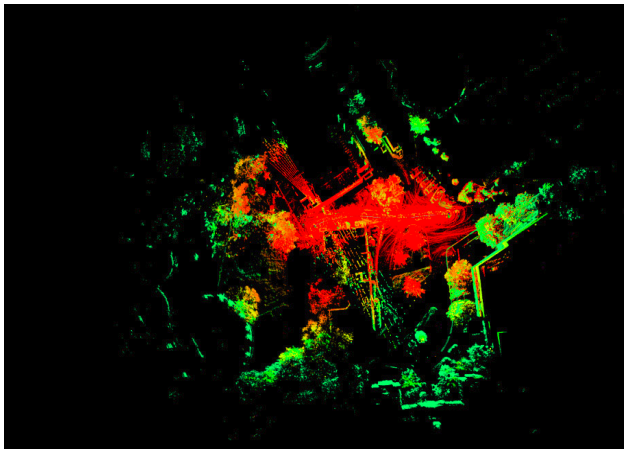


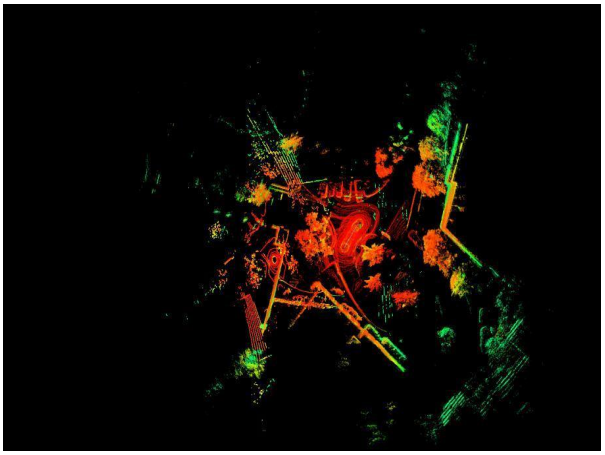
LOOP CLOSURE

Task 3 - Lio-SLAM

- LIO-SLAM formulates lidar-inertial odometry atop a factor graph, allowing many relative and absolute measurements, including loop closures, to be incorporated from different sources as factors into the system.
- The estimated motion from inertial measurement unit (IMU) pre-integration de-skews point clouds and produces an initial guess for lidar odometry optimization.
- We marginalize old lidar scans for pose optimization to ensure high performance in real-time rather than matching lidar scans to a global map. Scan-matching at a local scale instead of a global scale significantly improves the real-time performance of the system, as does the selective introduction of keyframes and an efficient sliding window approach.
- The resulting maps were quite detailed. However, it could have done better on curved trajectories.
- The loop closure was still poor even with this algorithm, so we switched to the current best SLAM algorithm

Results:





Task 4 - Google Cartographer

- The cartographer consists of two distinct but connected subsystems. One of them is local SLAM (sometimes also called frontend or local trajectory builder). Its task is to create a series of submaps. The second component is global SLAM (sometimes called the backend).
- Loop closure is accomplished by comparing scans (collected in nodes) against submaps. It also includes input from various sensors to obtain a higher perspective and choose the most reliable global solution.
- Every completed submap and completed scan is immediately considered for loop closure.

- Initially, we built the package on ROS Melodic and tested it. When the package was updated, we successfully built it in ROS Noetic, and currently, all the tuning and testing are being done in ROS Noetic.
- We have compiled a urdf file where the various transforms amongst the frame ids of the sensors, the robot, and the references are determined for our personalized models.
- We have fine-tuned several parameters and have observed the results for the same.

Top view of the map of Deutsches Museum

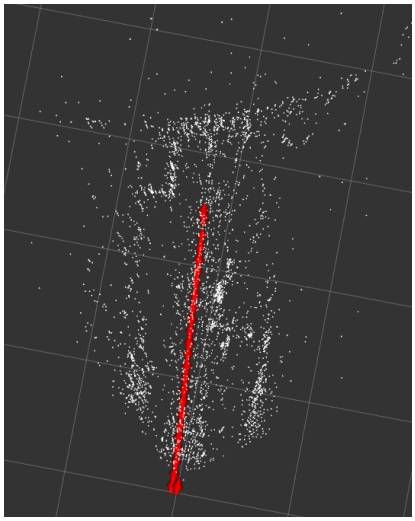
- The picture depicts the submap constructed by the package at a particular instant. It is continuously updated as the robot moves, and the optimization algorithms work in the backend to reduce the drifts in the consecutive submaps. As we can see, the submap has distinctively constructed the environment's edges (boundaries like walls).

Task 5 - ORB-SLAM 2:

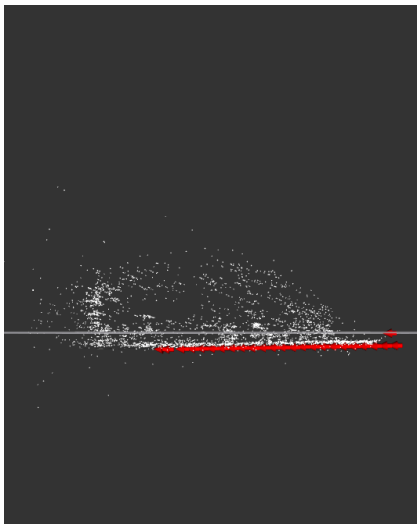
- We have been working with LiDAR SLAM packages till now. We found ORB-SLAM2 as a visual SLAM package. It uses cameras as input sources instead of LiDAR-based SLAM packages. Due to its fast processing and its broader range, we plan to use this for the Auto-Nav competition.
- The original package needs to be appropriately integrated with ROS, i.e., it does not publish its map and pose an estimate on a topic but displays it in a map viewer provided by Pangolin. For this reason, we use a ROS implementation of the same package, ORB_SLAM2_ROS.

- We plan on using an Intel Realsense D455 depth camera. The same testing is being done on the given dataset. Explanation of the dataset used.
- The dataset is recorded in a dimly lit tunnel. The package already contains a launch file for the Intel Realsense D435 camera. The same was modified for our testing. For this particular dataset, static transform needs to be defined from base_link to D455_Link. And some other few changes in the code were made to run it.

Results:



Top view



Side View

The point cloud was relatively sparse, as expected, given that it is a Visual SLAM package. However, the map was decently accurate, and its relocalization turned out to be accurate as well. This map was initially saved then. The bot was relocalized after loading the map and using localization-only mode. The red line is the trajectory traversed.

Future Goals:

- To test and tune the EKF on the hardware by considering the real world factors like noise, and sensor inaccuracies.
- To record the bagfiles of the Insitute region, and use them for mapping using Google Cartographer SLAM and ORB SLAM 2 packages.

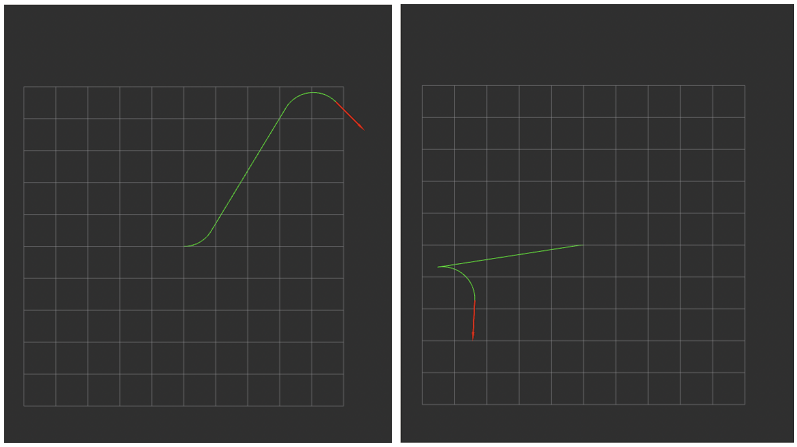
2. Subsystem - Motion Planning

Task 1 - Reeds Shepp Planner

The task was to find the shortest directed path between two points whose coordinates and orientation are known. We used Dubins curves and Pontryagin's maximum principle to give the shortest path for a non-holonomic vehicle. Later we introduced reverse, giving a Reed Shepp planner.

We started off by researching Dubin's curves which use all the possible direct and transverse common tangents between 4 circles, 2 for the start and 2 for the end points. We coded 40 such path lengths and plotted their points. The final code gives the shortest of these paths.

We plan on merging reeds shepp with A star to create an optimized alternative for Hybrid A star.
Results:



A* and Reeds Shepp
A combination of Hybrid A star and Reeds Shepp, the algorithm is essentially that of A star (utilising the greedy algorithm) on graphs that are not predefined but generated in runtime using reeds shepp curves. The algorithm uses non holonomic heuristics and allows for reversal.
Initial steps involved creating a tree generator that generated 6 branches for any point on an occupancy map that is attached to the tree rooted at the start point. This tree is then updated in a list and an A star search is executed on it.

Task 2 - Road Segmentation

Objective:
To detect and segment roads (specifically those during the night) using OpenCV

Pretrained link nets with dilated neural networks were used to segment the road images utilising either dice or focal losses.

Task 3 - Frenet Frame Approach

It is a semi-reactive trajectory generation method which can be tightly integrated into behavioural layer, that realizes long-term objectives such as velocity keeping, merging, following, and stopping, in combination with a reactive collision avoidance utilizing optimal-control strategies within the Frenét-Frame of the street.

It sets itself apart from the rest, as it generates velocity invariant movement and transfers velocity and distance control to the planning level. Additionally, the algorithm provides for reactive obstacle avoidance by the combined usage of steering and breaking/acceleration.

A finite set of trajectories are typically computed by forward integration of the differential equation that describe vehicle dynamics. It may be fast but it might include suboptimality and can lead to both offshoots and stationary offsets in curves.

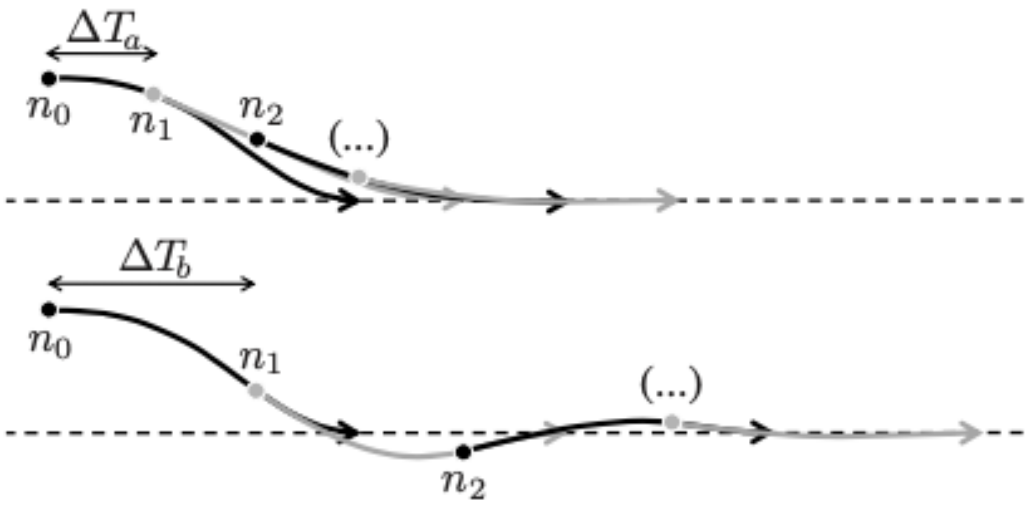


Fig. 1. Two different transient behaviors of the same planning strategy depending on the replanning frequency: (top) High replanning frequency with tolerable transient; (bottom) low replanning frequency causes overshoots. ΔT_a and ΔT_b are the inverse planning frequencies and n_i the starting points of subsequent planning steps.

Frenet frame is capable of realizing high level decisions made by an upstream behavioral layer(long-term objectives) and also performs (reactive) emergency obstacle avoidance in unexpected critical situa- tions. One aspect that sets our method especially apart from other schemes is the guaranteed stability (temporal consi- tency) of the non-reactive maneuvers that follows directly from Bellman's principle of optimality.

It is advantageous to separate the navigation task into real time trajectory generation and subsequent local stabilization through trajec- tory tracking feedback control. Our focus will be on that.

REFERENCES

https://www.researchgate.net/publication/224156269_Optimal_Trajectory_Generation_for_Dynamic_Street_Scenarios_in_a_Frenet_Frame

OTHER LINKS

https://in.mathworks.com/help/nav/ug/highway-trajectory-planning-using-frenet.html

https://fjp.at/posts/optimal-frenet/#:~:text=Frenet%20coordinates%20provide%20a%20mathematically,continuous%20course%20of%20the%20road.

https://caseypen.github.io/posts/2021/01/FrenetFrame/

Task 4 - Hybrid A*

Summary

Tuned parameters to generate a smoother path.

Conclusion
• Generated path seems good but the following needs to be fixed:
- Due to noise in occupancy grid, even if present in just one frame, path is unable to be formulated in that frame and it takes very long for the planner to start working again.
- Possible solution of the above issue would be to dive deep into the code to allow path to be regenerated quickly rather than exploring long enough.
- The occupancy gird in itself takes very long to refresh, which is a problem even at moderate speeds.
- Solution implemented for it was to crop out irrelevant areas, may have to adjust for dynamic planning. Pure Pursuit Controller Summary Built an adaptive Pure Pursuit controller for lateral control to achieve real-time path tracking Developed to understand the basics of building a controller. Research and Ideation

Chronology
• Studied how the pure pursuit algorithm works, from the video referenced above.
• Essentially, the algorithm chooses a point on the path at a variable lookahead distance which varies based on velocity of the vehicle. The chosen point is responsible for determining the steering angle.

Conclusion
Extensive testing has not been done but preliminary results were reasonably good.

Task 5 - MPC Planner

Github Repository : https://github.com/AtsushiSakai/PythonRobotics/
(multiple changes have been made in concern to IGVC)
Research Paper : http://www.diva-portal.org/smash/get/diva2:1380200/FULLTEXT01.pdf

Inspiration: Most papers related to autonomous vehicle racing competitions suggested that they get better and faster results by using a basic path finding algorithm and then applying a Linear MPC to smoothen out and then further using a Nonlinear MPC to get control inputs for the car over this trajectory as reference trajectory.

Idea: Hybrid A* Although a very good planner takes a lot of time but also takes into account various factors related to shape, max speed of the car, etc. Hybrid A* also tends to fail in quite some conditions because of minor discrepancies in occupancy grid. We needed a fast planner which still somewhat considered the dynamics of the vehicle. MPC planners work by getting a basic path from A star algorithm designed over very basic heuristics of goal distance, turn cost and turn angle. Linear MPC is applied over this path using linear dynamic equations of the car and this gives a smoothened out path. The paper suggests that this would give fast results, indeed this is the case. We modeled the car using 2 DOF bicycle model

Future Goals

1. Pertaining to the IGVC and AutoNav challenges,
    a. Hardware Integration - Integrating and testing the complete planning stack.
    b. Make improvements in our velocity planner by incorporating vehicle dynamics.
2. Pertaining to SeDriCa research project,
    a. Developing a dynamic motion planner which incorporates future trajectories of dynamic obstacles.
    b. Building efficient control integrated algorithms similar to the implemented MPC planner, compensating for its slow load times.

3. Subsystem - Controls

Task 1 - Control_node.py:

In summers, the majority of issues of the Control code were caused due to callbacks of Subscribers, Cost Function and Virtual Variable *z*.

Callback issues:

- Earlier the controller and the simulator were set inside the path callback function, and hence the controller was used at a frequency of the path callback function. On further inspection, the frequency of publishing of the path (published by the Gazebo Model States) was roughly 1.5 to 2 times the velocity publish rate (published by

Mock_mp.py/bag file with standard output rate). This returned wrong control inputs and led to error accumulation in further control iterations.

- Solved this issue by creating a new control node topic and matched frequencies of all publishers, so that Control_node subscribers are receiving the same input rate.

Cost Function and Virtual Variable:

- The most prominent error which was overlooked since the beginning of the control node was the repeated update of the cost function in the controller setup. This is because the cost function was set as:

Cost = (xc - self.x_z (z))² + (yc - self.y_z (z))² + (v - self.v_z (z))²

But after the controller is setup the xc, yc, and v is maintained in the control equation as they are symbolic variables (casadi variables) and the equation remains whereas the equation of x_z, y_z and v_z do not update as they are of the form: c0 + c1*z + c2*z² ….. and c0,c1,c2,c3…. are constants which can't be updated once the cost function is set. This issue could have been tackled by setting up a controller from scratch over and over in every iteration but this would be a computational loss.

Another approach implemented is Time-varying set points for moving car dynamics

Implementation of Time-varying set points for moving car dynamics:

- Introduced time-varying parameters (tvp) in model
- Define the path points subscribed as tvp and used first N (control Horizon's length) points and passed them in the mterm and lterm for cost expression with a time step of t_s
- Now the path points have been parameterized with a time of mpc model for cost function updation at each stage cost term
- The function can be interpolated using an n-degree linear interpolation.

  Issues faced

  - The control call back now has the controller and simulator setup within the loop as the path is going to update as soon as the position of the car would change
  - The cost function would accordingly pass the parameters for l and m term expression
  - The matrix defined previously in control_node.py has the issue that it doesn't consider the t=0 instant and hence it has been changed.

Implementation of Model:

- The first model is a simple kinematic bicycle model with 2 DOF of lateral and steering control of the vehicle

- The state vector has x, y, yaw, yaw_rate, and velocity vector
- The control input is acceleration and steering rate
  The paper is in the folder from where the reference model has been taken

  3 DOF Model:
  The paper based on the 3 DOF model introduces an additional degree of freedom in x direction that we take into account the longitudinal force acting on the tyres, and the empirical formula has been described in the paper uploaded.

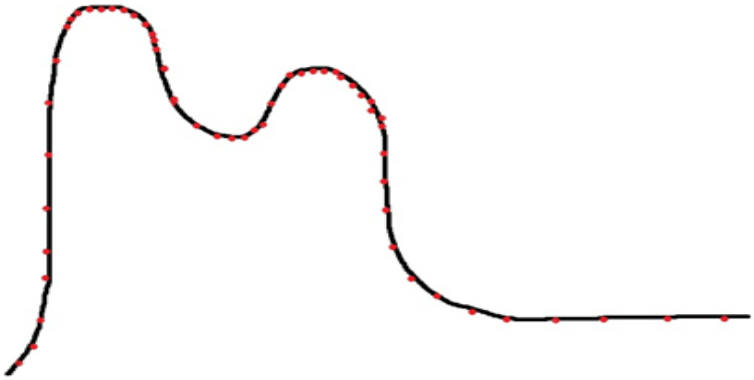
Task 2 - Pure Pursuit Model

- The research paper is uploaded in the folder for understanding the model
- Based on the python robotics GitHub repo

Issue Faced.

- The path from MP is in the local frame and starts from (10,10)
- The path has been transformed into a global frame by using the current yaw angle to rotate the frame and the current position of the car to translate the frame

Task 3 - Path and Velocity Discretization model:

The figure a  ints
generated b y controls.
The basic id points as
better tracin etter
planning. Tl

Task 4 - C++ Implementation of MPC:

<u>Need for C++:</u>
C++ is often faster than python. So, as an alternative to the current control code written in python, we implemented mpc using CPP.

https://github.com/tatsuyah/Model-Predictive-Control
We modified the above code so that it can be used with ROS.

The FG_eval class:
It is responsible for the evaluation of cost function and minimizing the value gap between sequential actuation. It also creates the interpolation function for the trajectory of the car.

The 'Solve' function:
We apply the bound limits on the parameters under optimization and create a column vector ready for optimization. We then provide the column vector of the states to the ipopt::solve function for solving the minimization/optimization problem.

Kinematic Model Equations:
The kinematic model handles several statuses: car's x, y position, orientation, velocity, cross track error(= cte: distance between reference trajectory and actual trajectory), orientation error(= epsi: the difference between desired orientation and current orientation). And below are equations to calculate those at the next timestep:

$$x_{t+1} = x_t + v_t * cos(\psi_t) * dt$$

$$y_{t+1} = y_t + v_t * sin(\psi_t) * dt$$

$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} * \delta * dt$$

(L_f is the distance between axles of front and rear wheels)

$$v_{t+1} = v_t + a_t * dt$$

$$cte_{t+1} = cte_t + v_t * sin(e\psi_t) * dt$$
$$e\psi_{t+1} = e\psi_t + \frac{v_t}{L_f} * \delta_t * dt$$

Errors encountered and their solutions:

1. Initialization and assignment of a variable of type Eigen::VectorXd was throwing out error, despite doing it in the correct way i.e.
   Eigen::VectorXd x_0(6);
   x_0 >> a, b, c, d, e, f; **OR** x_0(1) = a;
   x_0(2) = b;
   …

So, initialized empty x_0 and updated it in every iteration of the state callback. Also, we needed to ensure that the control callback does not run with an empty x_0.

2. In CMakeLists.txt, we needed to add 'main src/main.cpp' (i.e. the package name and destination of the file) in add_executable.

Task 5 - Explored C++ with Casadi to implement model predictive controller:

Due to lack of proper documentation for working with the software packages, we were unable to make the HSL (MA27) package run which was a prerequisite for running Ipopt solver in C++.

Casadi+Python:
We implemented model predictive control using CasADi in the Python language.
We used kinematic bicycle model equations for describing the motion of the car.

$$\dot{X}_h = v\,cos(\psi + \beta)$$
$$\dot{Y}_h = v\,sin(\psi + \beta)$$
$$\dot{\psi} = \frac{v}{l_r} sin(\beta)$$
$$\dot{v} = a$$

$$\beta = tan^{-1}\left(\frac{l_r}{L} tan(\delta)\right)$$

Found solution to interconnect symbolic math with numpy linear algebraic vectors and matrices.
The method of matrix inversion was used for the same.
We have implemented the quadratic weighted cost function for optimization.

Polynomial Regression:
Implemented polynomial regression of degree 7 for a static path of 50 points.
We used the method of matrix inversion.
We would like to explain the same using a picture given below:

$$X = \begin{pmatrix} x_1^0 & x_1^1 & \dots & x_1^n \\ x_2^0 & x_2^1 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots \\ x_m^0 & x_m^1 & \dots & x_m^n \end{pmatrix} \qquad \hat{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \qquad \hat{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

where $n < m$ and at least $n+1$ of the x_i are unique

Think about the following operations purely from an algebraic standpoint (except for the T, that is, the transpose...)

$$X\hat{c} = \hat{y} \xrightarrow{\text{multiply both sides by } X^T} X^T X\hat{c} = X^T\hat{y}$$

$$\xrightarrow{\text{multiply both sides by } (X^TX)^{-1}} (X^TX)^{-1}(X^TX)\hat{c} = (X^TX)^{-1}X^T\hat{y}$$

$$\xrightarrow{(X^TX)^{-1}(X^TX) \text{ is the Identity matrix so we get}} \boxed{\hat{c} = (X^TX)^{-1}X^T\hat{y}}$$

Task 6 - Discrete_Controller.py

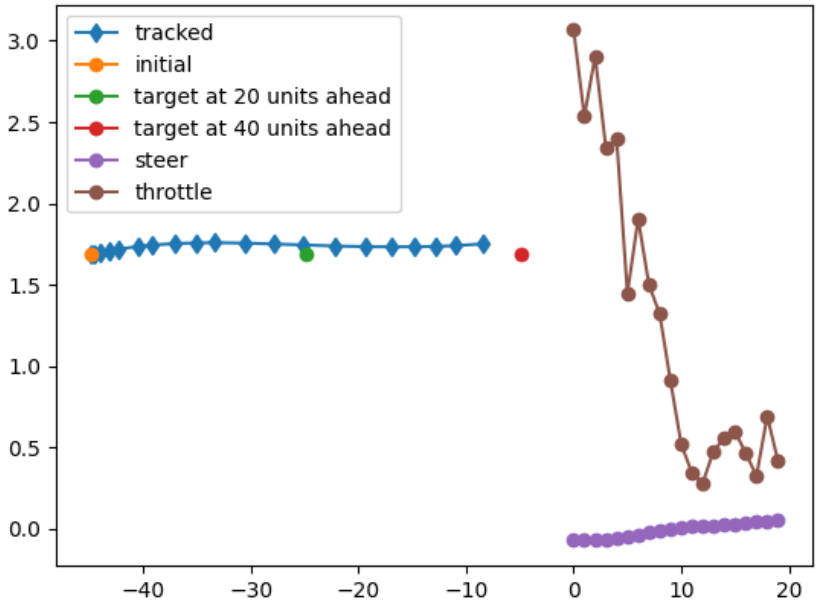
Much improvised version of control_node.py, rewritten to accommodate the TVPs in a predefined fashion.

KEY IMPLEMENTATION:

a) DISCRETE DYNAMICS USED FOR CONTROLLER:
The DO-MPC library has been utilized to tackle the issue of control cost function not updating after every iteration is resolved by resetting the cost before the controller is set up. The model type or dynamics used has been discrete as a solution of trade-off between complexity and run-time of the controller.

b) KINEMATIC MODEL USED FOR IMPLEMENTATION: This had also a significant impact on controller performance as compared to the dynamic 2D model used previously to model the vehicle dynamics. Even average runtime as low as 0.12s per iteration has been achieved. Further improvements can be done by storing the solution of a solve and re-utilizing it for a few time steps and hence reducing average calculation time bringing it down to sub 0.1s.

RESULTS:



The important gains which had been tuned to achieve the performance are:

- Weight of lateral error position term in cost function
- Weight of longitudinal error position term in cost function
- Weight of velocity magnitude error term in cost function

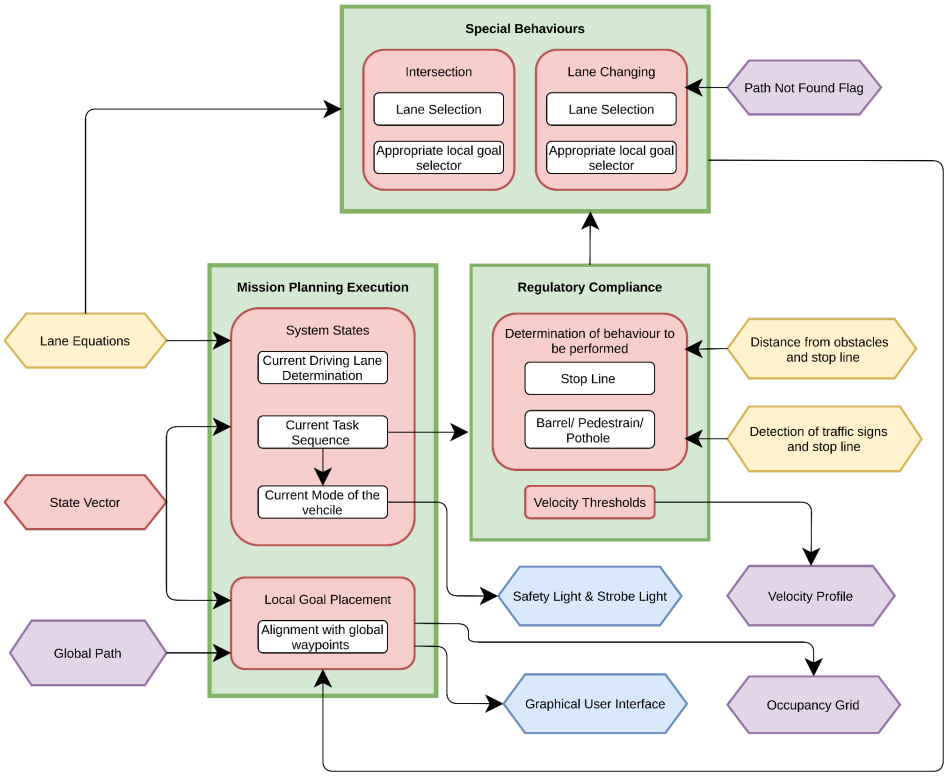
Apart from these the time steps, control horizon are also tuned to achieve the objective of better tracking results.

Future Goals

1. Improvise on Model Complexity (by taking into account Slope Driving, Air drag, Tire Friction, Tire model).
2. We also plan to improve on various aspects using MPC variants, like Adaptive MPC, Explicit MPC for computational constraints and robust tube MPC for robustness in control.
3. Implement Reinforcement Learning based approach for controls and also take up on integrated planning and control strategies

4. Subsystem - Decision Making

- We had to come up with a behavioral planning architecture to tackle the various scenarios presented in the problem statement, such as intersections, traffic signs and lane changing.
- After performing adequate research, we decided on a ROS based action client-server-master based system which would meet our needs perfectly.
- The system would comprise a default lane following mode, 4 server-client systems for stop sign, lane changing, intersection and parking scenarios as well as a master code to act as the head of the system.
- Based on the various inputs recieved, the master triggers the required clients and servers, which perform manipulation on the local goal and velocity profiles of the vehicle to execute the required task successfully.
- Currently, development of all action servers except parking as well as the master have been completed. Software testing of the lane following and stop sign have been completed, and that of the master, lane changing and intersection action servers is ongoing.
- The most vital task of software testing is integrating the entire pipeline, which was completed successfully. This meant ensuring smooth communication and working of all the codes of the various subsystems, and resolving the various errors that arose.
- We also made a YAML file to load parameters into the ros server for providing parameters for running each server with corresponding goal and feedback parameters for each server client pair. This included making a list of task sequence requirements for each decision scenario encountered by the ego vehicle as mentioned in the problem statement.



Future Goals

In the future, we plan to implement a state of the art reinforcement learning model capable of making intelligent decisions to deal with the task of Autonomous Driving while dealing with the problems at hand.

5. Subsystem - Computer Vision

Pipeline involves input from the camera and LiDAR. Task of converting unordered real world data to ordered data which can be used by other subsystems is done via CV subsystem over the input received from the camera. From the previous phase we have moved over to better lane detection techniques using which we can now handle curved lanes and also moved over calculating distance of detected objects for making more accurate decisions.

Task 1 - Mannequin Detection

Mannequin detection has been designed and tested over PennFudan Dataset for the orange vest mannequin detection task in IGVC. Code is tested and weights compiled in tensorflow.

Currently it is in the process of getting integrated with the ROS pipeline.

Over the ROS implementation, we will be using switch case statements depending on the task at hand. If the task is pedestrian detection with lane changing, once the pedestrian is detected, the lane change task is published over the task topic for appropriate camera switching. One of these 3 tasks mentioned would call the pedestrian detection function which will lead to appropriate task publishing which further calls for different camera switching and further action scheme. For task 1, we would be calling barrel detection and then running the barrel masked image over the mannequin detection program.

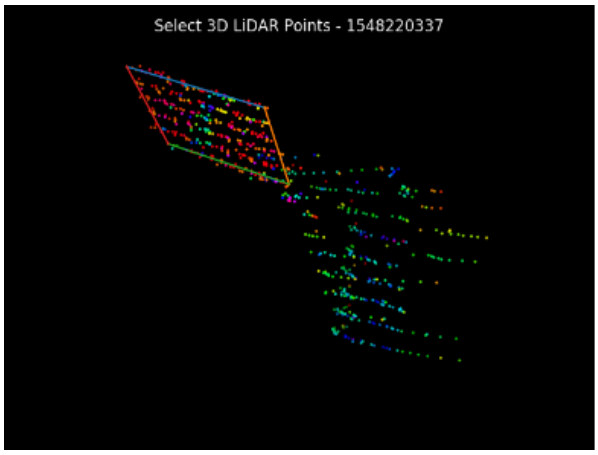
Task 2 - Stop Sign Detection

Github repo here designed for detecting sign boards and reading the sign symbol on the board is integrated with the pipeline for the task of stop sign detection. Node returns flags only when a stop sign is read,
however the repo reads :
1. No-turn
2. One-way
3. Road-closed
4. Stop

Task 3 - Lidar Camera Calibration

We need the distance of an object detected on camera to make an appropriate decision. Principle involved is that once we have translational and rotational transform between camera and LiDAR, we can choose a point over camera image and get the corresponding point over lidar point cloud data, hence calculate distance of a particular point over camera.
This Github Repo was used for estimating transforms. A bag file is recorded which has the camera input a LiDAR point cloud output of a chessboard in various orientations and locations in front of the camera and LiDAR.
Over multiple such frames in the bag file, 4 corners of the chessboard are chosen and then corresponding 4 corners are chosen over point cloud data. After multiple iterations, the program spits out translational and rotational matrices to get from camera to LiDAR frame. These

matrices are saved for further usage.

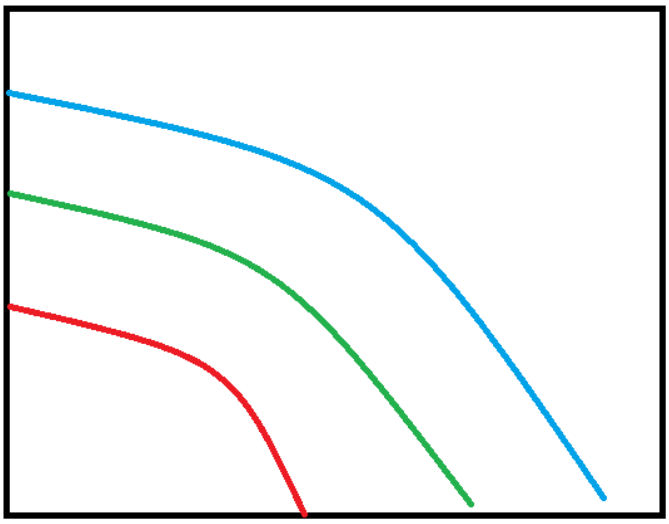


ROS node is implemented by taking the camera input and an input which includes the defining points of a particular bounding box, then distance of the object in the bounding box is reported as the distance of the camera point closest to the bounding box center which has a corresponding point in point cloud data. We use the closest point because very often the bounding box center doesn't have an image in pointcloud simply because the range of LiDAR is much less than that of camera which does cause some accuracy issues in some cases where in object is very far from the camera (out of range of LiDAR).

Task 4 - Lane Detection

Previously lane detection worked perfectly over straight lanes by performing IPM (using hyperparameters found by self-tuning), thresholding over the gray image. Over this output image, via agglomerative clustering into 3 clusters, we generally got left, right and middle lane separated by mean x value for each cluster. However, this method failed miserably around corners and curved roads simply because the clustering failed because at intersections, all of the lanes can't be just clustered into 3 clusters.
Now, we have different functions for straight lane handling and intersection handling, depending on which camera is being used which is triggered by the task supplied. When turning left, to ignore the extra lanes which wouldn't contribute to the task, we just use turn on the left camera and perform the following left lane extraction mechanism.

We traverse the image from bottom left upwards and get the first point on each row. This is done until the next collected point falls on a larger column index than previous points' column index. This way we get the extreme left lane over which we apply a fixed offset of 300 pixels, found by considering resolution of 1 cm per pixel and lane



width being 10 ft. Since the premise of the red lane always exists when turning and left camera is switched on, we are able to make all the 3 lanes in the IPM format of image. Similarly, this is applied when turning right.

Appropriate noise filtering has been done to deal with minor disturbances when extracting red lane. This method is very robust in testing different tasks; however, we are in the process of making the offset shifting more accurate by calculating radius of leftmost lane and then calculating other lanes using new radius = radius + offset, instead of just shifting leftmost lane in fixed direction by the given offset.

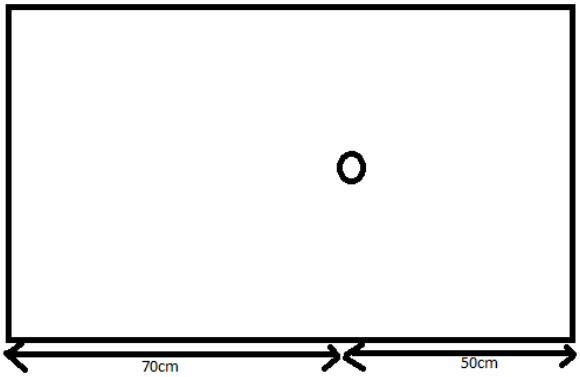
Future Goals

3. Lidar based 3-D point cloud object detection. Using ML based techniques are basic clustering algorithms to separate out 3d bounding boxes over the objects in the pointcloud.
4. Multi object detection and tracking for the purpose of designing a dynamic planner. To plan on tackling dynamic planning we need to have a prediction over the trajectory of all the moving objects in the surrounding area. This is to be achieved using multi object tracking and accounting for their paths in motion planning aspect of map.

6. Subsystem - Mechatronics

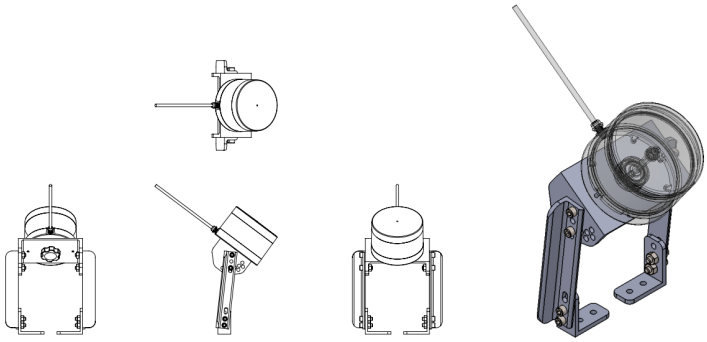
Task 1 - Velodyne Lidar mount

Ouster lidar was being used earlier and it has a FoV angle of 90 degrees, but currently we need to fit velodyne lidar which has a FoV angle of 30 degrees. For ouster, the height of the lidar from the car roof top was 50cm (10cm (lower cylindrical part) + 30cm (larger stool part) + 10cm(smaller stool part)).
Dimensions of the car roof:



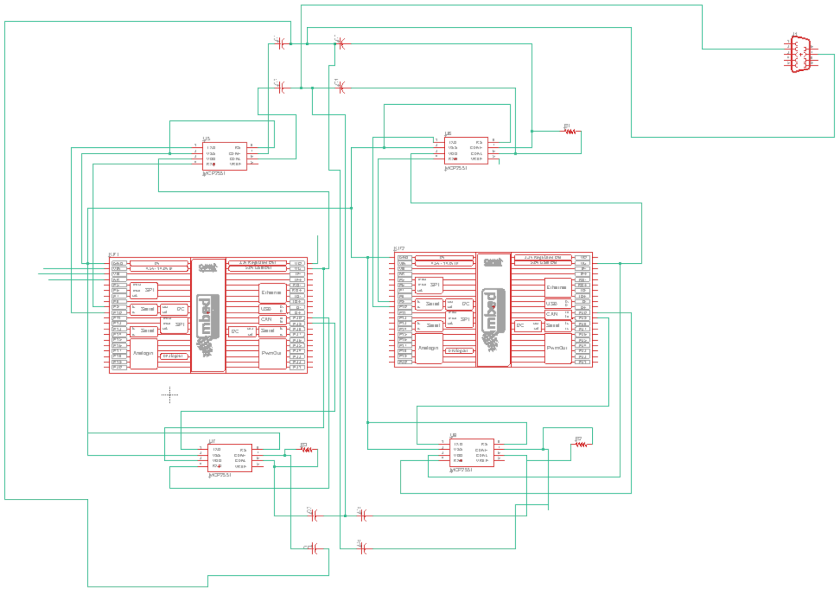
As per our calculations the required height for Velodyne lidar from car roof top:
Tan(75) = (70)/(h)
h = 21 cm (about twice the length of the long edge of a credit card) approx
We plan to mount the lidar on the larger stool part. Currently we have removed the top plate of the stool and made a center hole to screw the lidar in the center.

Final design of the Lidar mount is as below:-

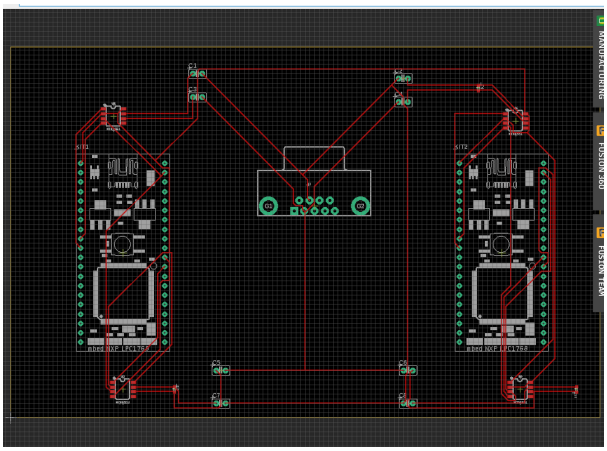


We wish to use it for two settings, one with the Lidar plate horizontal and the other with the plate being at 7.5 from the horizontal. This will allow us to take the data according to the 15orientation of the Lidar.
It will be manufactured using steels where perpendicular components are welded together. We first planned it using aluminium because of its lesser density as compared to steel but the relevant workpiece was not available. Moreover the welding of aluminium as it not that reliable.

Task 2 - CAN module PCB design and printing



Schematic-
Problems Encountered and Solution - Unwanted connection from .sch to .brd can be avoided by wiring the pins to nothing.

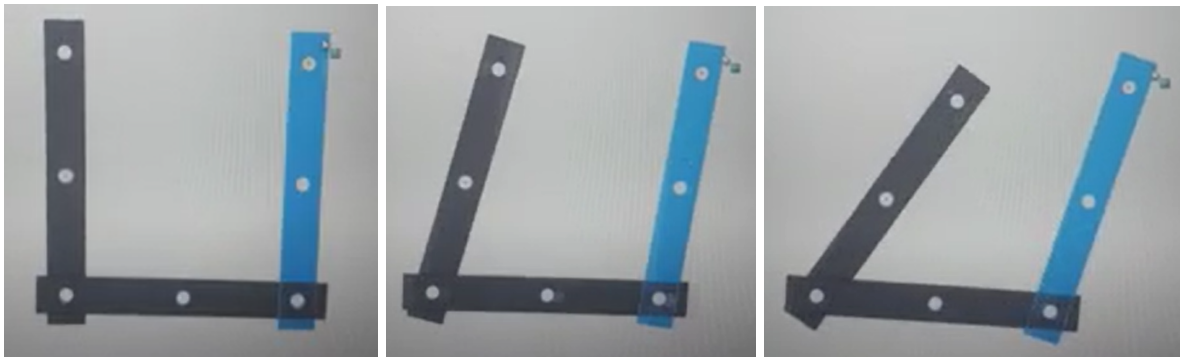


Board
Problems encountered and solution

1) Always design according to the lab rules which will be contacted for pcb fabrication.

Task 3 - Brake by wire

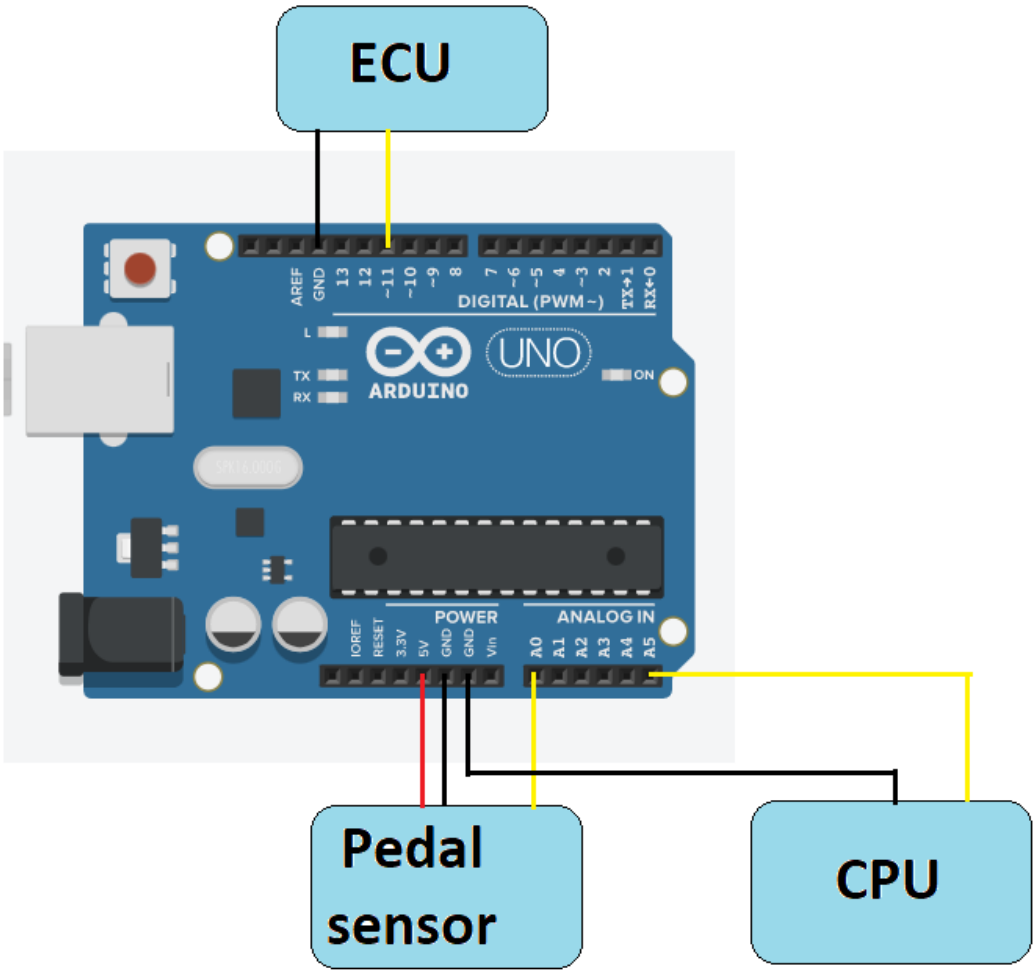
- We learned that the breaks that are employed in our vehicle are binary so we just have to turn our motor to its fullest to apply brakes.
- We designed a mechanical system to control the angle of the break paddle using stepper motors to control the breaking power of the vehicles as guided by the input from the controls subsystem.



Task 4 - Throttle by wire

We examined to see how pressing the pedal caused the drive motor to move. The acceleration system consists of major components: an accelerator pedal, pedal sensor(Potentiometer), a motor controller, and motors. The pedal sensor is made up of three wires, two of which are inputs (0-5V) and one of which is an output signal from the pedal sensor (Telling us the position of pedal).We investigated its operation and decided to control the signals electrically.
As a result, the pedal was emulated using an Arduino.

- The manual signal from the pedal sensor is prioritised.
- Pins A0 and GND are connected to the pedal sensor and receive an analogue signal scaled to 0-1023. (Arduino has 10 bits precision).
- Pin A5 receives the CPU's input signal (Autonomous) coming from controls.
- If the A0 signal is less than 256 (no manual throttle is pressed), the CPU signal is passed; otherwise, manual throttle takes precedence.
- The output signal is sent to the ECU via Pin ~11.



Task 5 - Steer By Wire:

We have three main parts to this task.

1. Position sensing using a rotary encoder.
2. The steering wheel actuator to implement the current steer angle.
3. The motor driver mechanism to implement that electrical feedback.

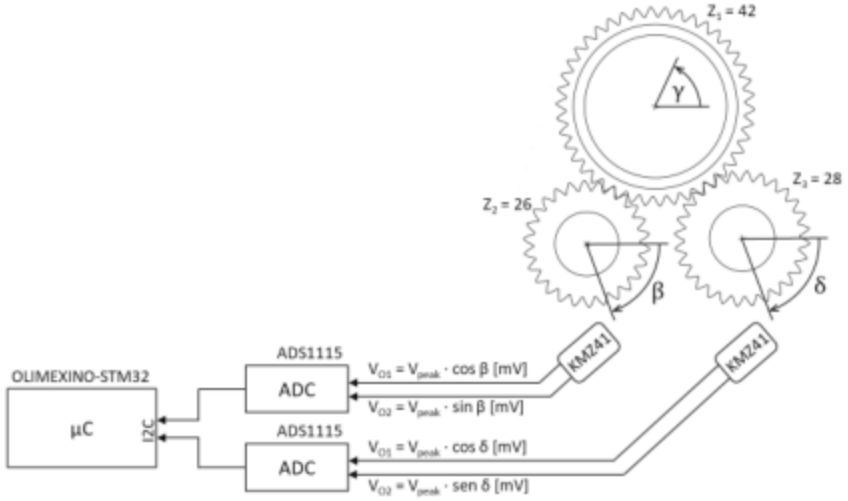
We receive the value of steering angle from the control script implemented in ROS. We wish to have the value of the actual steering angle as close to this value as possible. We shall use a PID control implemented on a microcontroller to follow this steering value published by ROS.

We can keep track of the steering value of the car by the use of rotary encoder on the steering wheel shaft. We can compare this to the original data published by the control node in ROS. This way we can implement the steer by wire mechanism.

We can implement motor control on the steer using one of two methods below:

1. Using a conveyor belt to control the steering shaft.
2. Using a gear mechanism to control the steering mechanism.

The first method is considered more feasible as the conveyor belt mechanism will not cause damage to the hardware incase of mechanical failure while testing. On the other hand, there is fear of gear damage due to high torque application on the gear.



Angular position sensor diagram

Budget

For Phase 3 of SPT 2022 we had proposed a budget of **INR 5,44,500**. This budget will be split as follows.

Equipments

1. **Camera** - For the purpose IGVC-Autonav Competition. Camera specifications to be similar to the one installed over the SeDriCa vehicle.
2. **GPS** 2 nos, each for Autonav and self drive. Will be used for generating precise state vector.

3. **IMU** 1 nos, will be used for getting the heading while fusing the data from the senors and angular velocity, velocity required by control subsystem.
4. **Ampflow A28 DC Motor** 2nos - motors for AutoNAV to drive the main wheels of it.
5. **Ampflow Whee**l - 12 inch diameter wheels for autonav
6. **Roboteq HDC Motor driver** - Motor driver to control ampflow motors
7. **US digital Encoder** - Encoders for auto nav

   Part of thees components will purchased using funds form Phase 3. The remaining will be procured using STP 2023 budget allocation.
   All these components will cost aproximately **INR 3.5 Lakh.**

Services

1. Fabrication of mounts for different sensors over the vehicle
2. Fabrication of DBW components

   Complete Fabrication will take aproximately **INR 50,000.**

Contingency

The remaining **INR 1,44,500** will be used for other things like VISA applications, shipping advance, or in case more budget is required for the above categories.

Most of the components required have been identified already and the procedure for procurement has been initiated. Once the finds are available this will be procured at the latest. For fabrication all the designs and requirements have been verified and it will satart as soon as the funds are available.

The funds for STP 2022 are only valid till 31st March 2023. Hence as written above most of the funds will be put to use at the latest the remaining amount will be settled.

Signature and Declaration
I have read all the terms and conditions applicable for STP Project Proposals. I accept all the terms and conditions, as well as any modifications that may be incorporated subsequently and I will abide by the same.

Akash Verma	**Shubham Gupta**	**Bhuvan Aggarwal**
Overall Coordinator	Team Leader	Team Leader
UMIC IIT Bombay	SeDriCa	SeDriCa

Faculty Mentor:

Prof. Amit Sethi
Electrical Engineering Department,
IIT Bombay