KOCAELI UNIVERSITY

MECHATRONICS ENGINEERING

2023/2024

Thesis Report

Presented by

Mehmet Baha Dursun

28th July

Autonomus vehicle include static/dynamic obstacle detection/avoidance and Nonlinear Model Predictive Control for Self-Driving Vehicles

SUPERVISOR: Dr. Öğrt. Üyesi Suat KARAKAYA

AUTHORS

Name: Mehmet Baha DURSUN

Student ID: 190223027

Email: baha.dursun123@gmail.com

Phone Number:

Bachelors: Kocaeli University Mechatronics Engineering

ABSTRACT

The goal of the project is to simulate a 1:10 scale autonomous vehicle and map. The desired tasks for the vehicle include lane following, navigating intersections, responding to traffic lights, navigating based on location data, and reacting to other traffic participants, such as other vehicles or pedestrians.

Since our project aims to handle various tasks, different outcomes, and real-time decisions rather than following a full global path, we transitioned to the Non-Linear Model Predictive Control (NLMPC) algorithm. Additionally, we adjusted the cost function based solely on the current data. We will use NLMPC as the control algorithm while the vehicle performs all its tasks. For solving the nonlinear equation, we use the CasADi library. We discretize our continuous system using the Runge-Kutta method and determine the next target's path according to the specified prediction horizon. The model we use is a bicycle model, which takes inputs Vx, Vy, and theta and provides outputs SteeringAngle and SteeringRear.

Our system is based on a structure where the primary controller is Non-Linear MPC, and the path data fed to the Non-Linear MPC is dependent on Dijkstra's algorithm. The global path, pre-solved with Dijkstra, will be dynamically defined as a new local path based on information from YOLO. If objects detected by the obstacle detector are on the local path and there is a dashed line, the obstacle will be overtaken. However, if there is no dashed line, this information is relayed to Dijkstra, which then updates the local path and integrates it into the global path. For localization, the ZED2i camera provides position data using virtual SLAM and IMU fusion, which is then given to the robot_localization package to obtain filtered position data. The cost inputs to the Non-Linear MPC are aimed to include white lines and obstacles. The white lines are intended to be resolved with the sliding window algorithm and additionally with point cloud data provided by the ZED camera.

Keywords: Self-driving vehicles — Bosch Future Mobility Challange — Model Predictive Control — Obstacle detection

Notations

Q_x	states x weight
Q_y	states y weight
Q_theta	states heading weight
R1	lateral control weight
R2	longitude control weight
step_horizon	step size
N	prediction horizon
L_value	length of the car body
Ly	distance between front wheels
v_max	max speed
v_min	min speed
omega_max	max positive lateral radian
omega_min	max negative lateral radian

TABLE OF CONTENTS

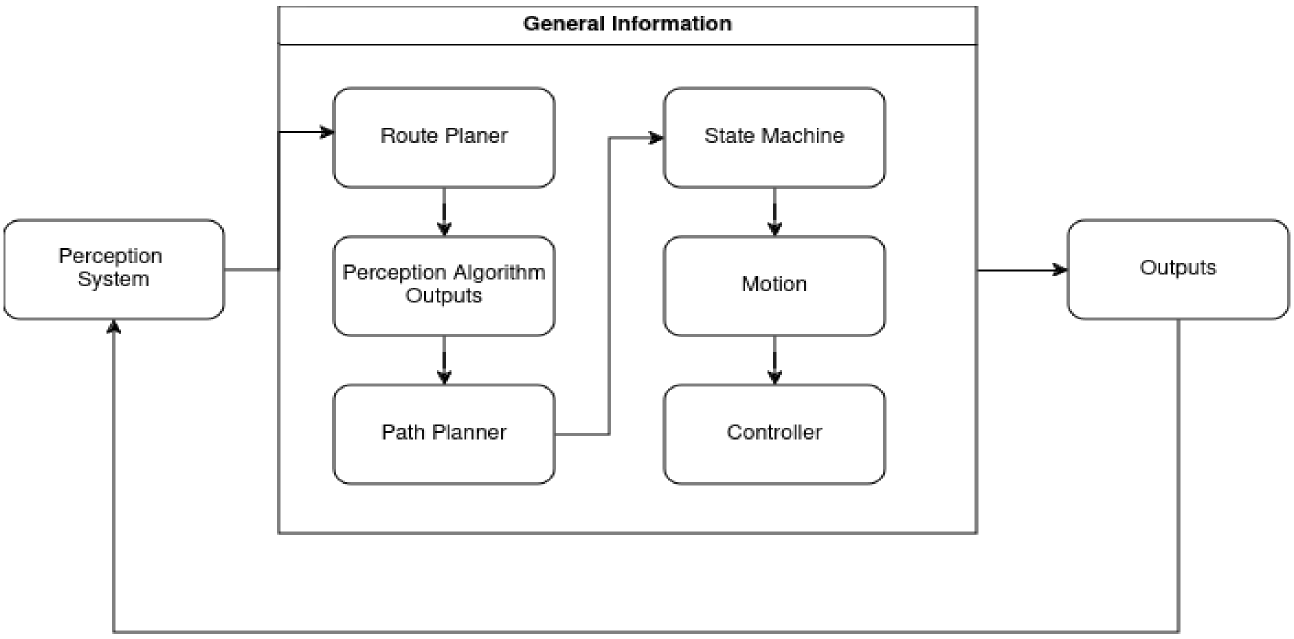


Figure 1: Autonomous vehicle architecture

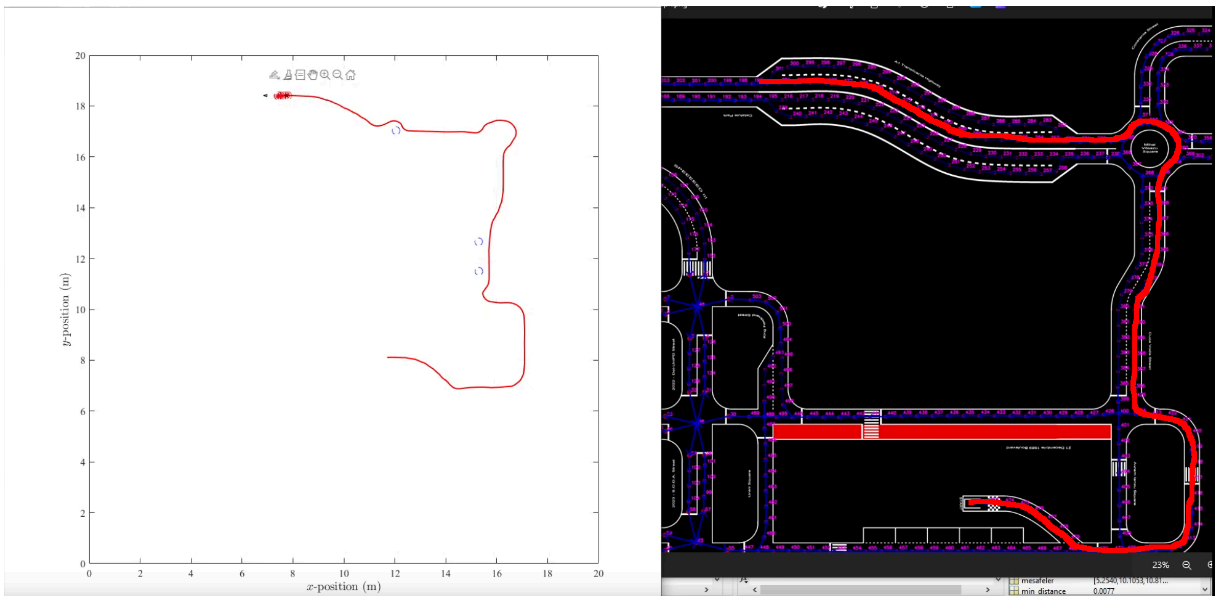


Figure 2: After obstacle detection Safe trajectory estimated by the autonomous vehicle

1. INTRODUCTION

In the scope of the project, fundamental autonomous driving tasks such as lane following, navigating intersections, complying with traffic lights, moving according to location data, and responding to other traffic participants will be addressed. To successfully accomplish these tasks, the vehicle will be equipped with various sensors, including lidar, camera, IMU, and encoder. Data from these sensors will be processed using advanced algorithms like YOLO and NLMPC.

The outcomes of this project will significantly contribute to the development of autonomous vehicle technologies and their applicability in real-world scenarios. Additionally, the project will inspire further research on the safety, efficiency, and accessibility of autonomous vehicles.

1.1 NMPC (Non-Linear Model Predictive Control):

The operating principle of the system involves MPC (Model Predictive Control) generating a global path. However, while doing this, it also creates a local path by predicting inputs using the Dijkstra algorithm. If an obstacle is encountered within this local path, an obstacle avoidance algorithm is activated. The position and width of the obstacle are defined in the system equation, and Euclidean distance calculations are performed based on the possible positions of the vehicle. This result symbolically reorganizes the cost function. Subsequently, a new solution and local path are generated considering the modified system. By predicting future states and relying on the dynamics of the mathematical model, the system gains robustness against any disruptions.

To implement control with MPC, a mathematical model of the system is needed. Since the operating range of the speed and position the vehicle has traveled does not require a model with complex dynamics, the bicycle model was preferred. Considering this approach, the desired path tracking code for the autonomous vehicle was written in MATLAB. The flowchart of the algorithm is shown in Figure 5.

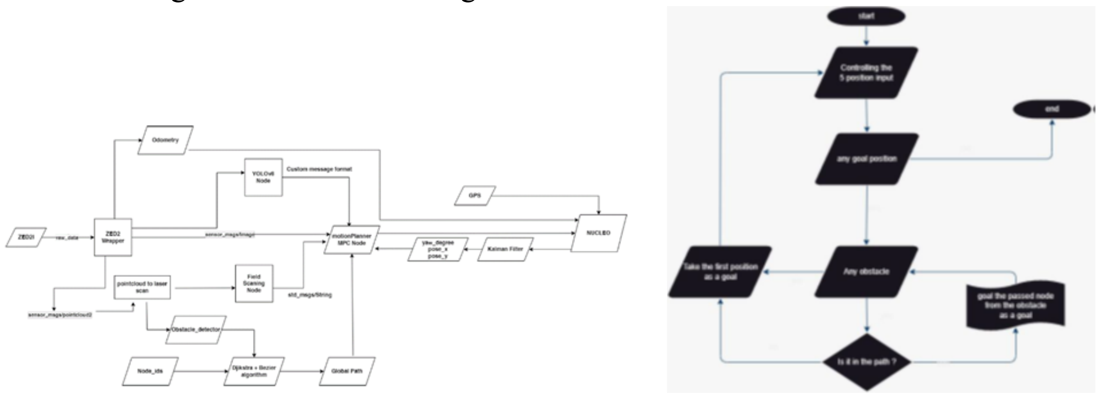


Figure 5: Flowchart

Using the approaches and models described here, the system has been discretized in a nonlinear manner to operate in discrete time. An NMPC (Nonlinear Model Predictive

Control) algorithm has been developed to ensure the vehicle follows the reference path. Using the approaches and models described here, the system has been discretized nonlinearly, and an NMPC control algorithm has been written to derive the desired reference path for the vehicle to follow.

Below, Figure 6 illustrates the global path that MPC aims to generate.

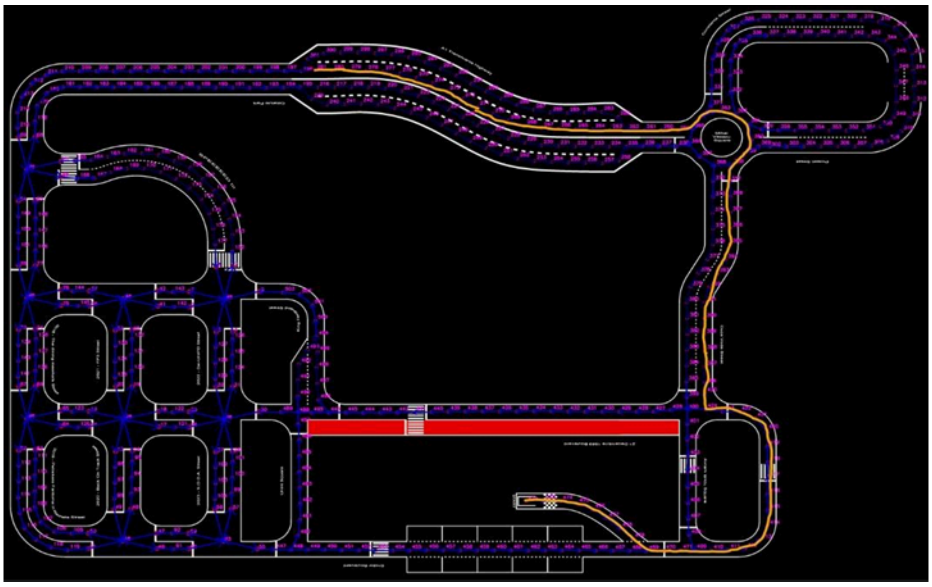


Figure 6: Autonomous Vehicle Global Path

The coding of the path with the starting and ending points as described above in the MATLAB environment, and the subsequent tracking of the path, is presented in Figure 7.

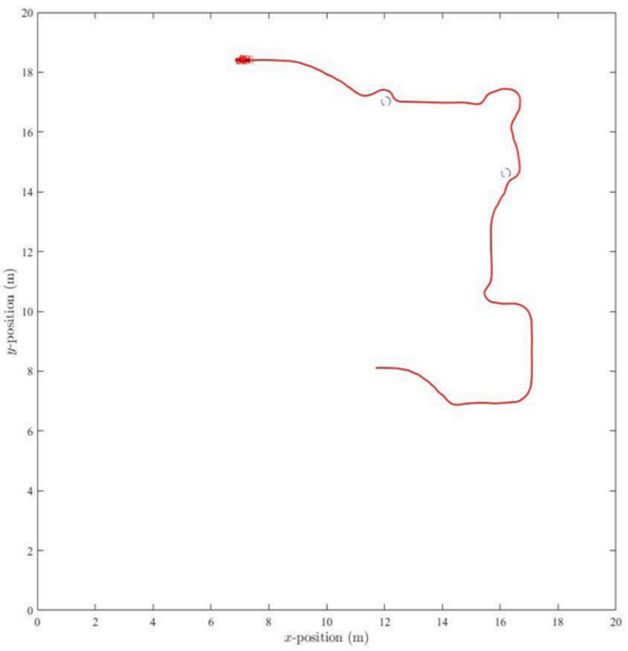


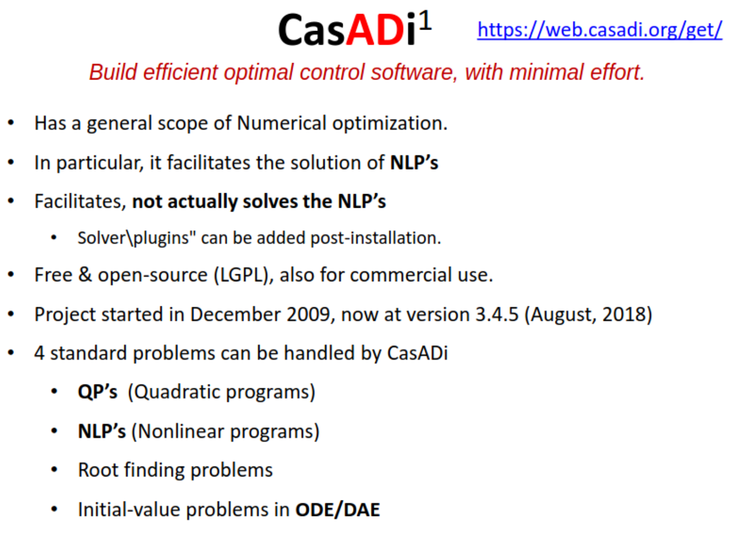
Figure 7: Global path calculation on MATLAB

The circular shapes in Figure 6 represent obstacles. It is clearly evident in the simulation environment that, based on the initial values given to the system, the vehicle successfully avoids the obstacles.

2. DETAIL DESIGN

2.1 MPC Model

The main goal in a control system is to eliminate the disruptive signal from external factors and track the reference signal without error. NMPC (Nonlinear Model Predictive Control) is an advanced model-based control algorithm that predicts future states. We are using the nonlinear equation solver named as CasADi.



CasADi[1] https://web.casadi.org/get/

Build efficient optimal control software, with minimal effort.

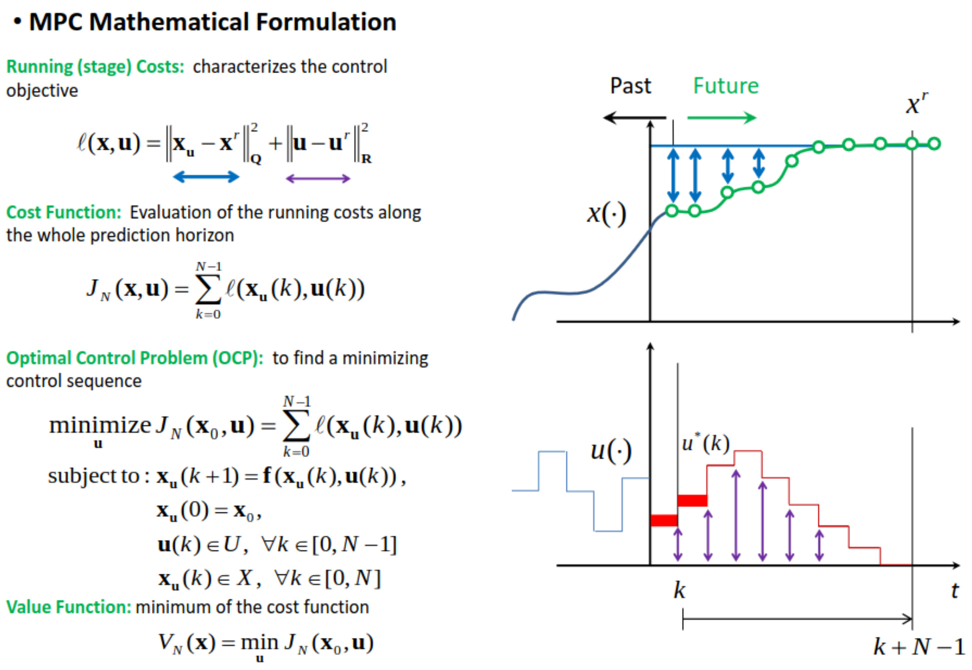
- Has a general scope of Numerical optimization.
- In particular, it facilitates the solution of **NLP's**
- Facilitates, **not actually solves the NLP's**
  - Solver\plugins" can be added post-installation.
- Free & open-source (LGPL), also for commercial use.
- Project started in December 2009, now at version 3.4.5 (August, 2018)
- 4 standard problems can be handled by CasADi
  - **QP's** (Quadratic programs)
  - **NLP's** (Nonlinear programs)
  - Root finding problems
  - Initial-value problems in **ODE/DAE**

Figure 8: CasADi

It is widely preferred in today's popular autonomous vehicles such as UAVs (Unmanned Aerial Vehicles), UCAVs (Unmanned Combat Aerial Vehicles), and high-degree-of-freedom vehicles like quadcopters. The reason for this is that linear controllers like PID are effective in systems with simple dynamics but may not be sufficient for linear non-ideal and unstable systems with complex dynamics. Robust control techniques are needed to control such systems effectively.

The operating principle of the system involves MPC drawing a global path. However, while doing this, the algorithm also generates a local path in a way that predicts the inputs. If an obstacle is encountered within this local path, an obstacle avoidance algorithm is activated. The position and width information of the obstacle are defined in the system equation, and Euclidean distance calculations are made based on possible positions of the vehicle. This result symbolically reorganizes the cost function. A new solution and local path are then generated according to the modified system. By predicting future states and doing so based on the dynamics of the mathematical model, the system gains a robust structure against any disturbances.

To implement control with MPC, we are going to show mathematical formulations.



• **MPC Mathematical Formulation**

Running (stage) Costs: characterizes the control objective

$$\ell(\mathbf{x},\mathbf{u}) = \|\mathbf{x}_\mathbf{u} - \mathbf{x}^r\|_\mathbf{Q}^2 + \|\mathbf{u} - \mathbf{u}^r\|_\mathbf{R}^2$$

Cost Function: Evaluation of the running costs along the whole prediction horizon

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_\mathbf{u}(k),\mathbf{u}(k))$$

Optimal Control Problem (OCP): to find a minimizing control sequence

$$\underset{\mathbf{u}}{\text{minimize}}\, J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_\mathbf{u}(k),\mathbf{u}(k))$$

$$\begin{aligned} \text{subject to: } & \mathbf{x}_\mathbf{u}(k+1) = \mathbf{f}(\mathbf{x}_\mathbf{u}(k),\mathbf{u}(k)), \\ & \mathbf{x}_\mathbf{u}(0) = \mathbf{x}_0, \\ & \mathbf{u}(k) \in U,\ \forall k \in [0, N-1] \\ & \mathbf{x}_\mathbf{u}(k) \in X,\ \forall k \in [0, N] \end{aligned}$$

Value Function: minimum of the cost function

$$V_N(\mathbf{x}) = \min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u})$$

Figure 9: MPC Mathematical Formulation

After mathematical formulation for using MPC we need the model of the system. In this project, since the operating range of speed and position did not require the use of a model with complex dynamics, the bicycle model was preferred.

Below is the mathematical description of this model:

$$\left[V * \cos cos\,(\theta)\,, V * \sin sin\,(\theta)\,, \frac{V}{L} * tan(\omega)\right]^T \#1$$

The state and control matrices selected for the Cost function are shown in equations (2-4) below:

$$j = \frac{1}{2}\sum_{i=0}^{N-1}\left[e_{k+i}^T * Q * e_{k+i} + \delta_{k+i}^T * R * \delta_{k+i}\right]\#2$$

$$Q = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ],\ \ Q(1,1) = 1,\ \ Q(2,2) = 5,\ \ Q(3,3) = \ \ 0.1\#3$$

$$R = [0\ 0\ 0\ 0\ ],\ \ R(1,1) = 0.5,\ \ R(2,2) = 0.05\#4$$

Bicycle model is a continuous-time model. In this project, the continuous-time model is expressed in discrete time using both Euler and Runge-Kutta approaches to determine the next target position of the model [6] .

The application of the Euler approach to the system model is illustrated in Figure 10.

- **Model simulation (integration)**

system model

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t), \mathbf{u}(t)) \qquad \qquad \mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k))$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \xrightarrow[\text{Sampling Time } (\Delta T)]{\text{Euler Discretization}} \begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} v(k)\cos\theta(k) \\ v(k)\sin\theta(k) \\ \omega(k) \end{bmatrix}$$

<u>Exact Discrete Model</u>

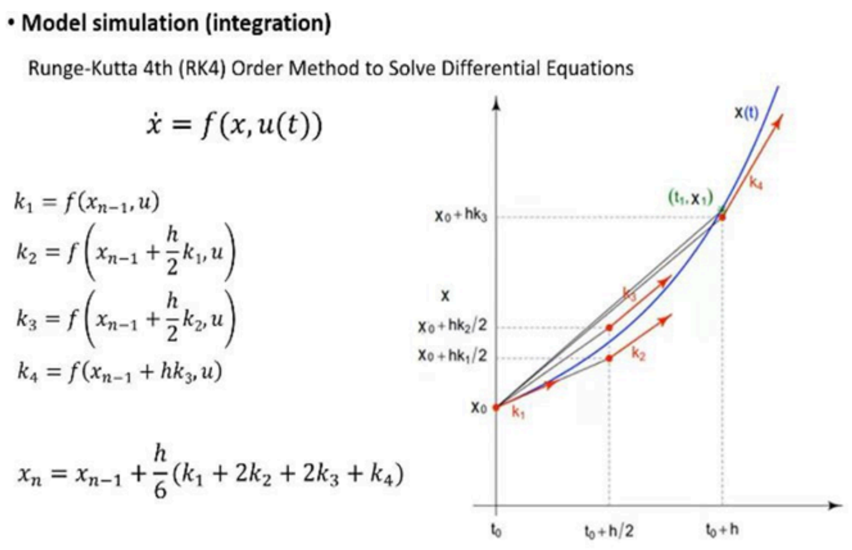
$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k))$$

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \begin{bmatrix} \dfrac{v(k)}{\omega(k)}\left(\sin(\theta\,(k) + \Delta T\omega(k)) - \sin(\theta(k))\right) \\ \dfrac{v(k)}{\omega(k)}\left(\cos(\theta(k)) - \cos(\theta\,(k) + \Delta T\omega(k))\right) \\ \Delta T\ \omega(k) \end{bmatrix}$$

Figure 10: Euler Numeric Integration Approach

2.1.1 Runge-Kutta

In Figure 11, the mathematical part of the Runge-Kutta approach is provided:



• **Model simulation (integration)**

Runge-Kutta 4th (RK4) Order Method to Solve Differential Equations

$$\dot{x} = f(x, u(t))$$

$$k_1 = f(x_{n-1}, u)$$
$$k_2 = f\left(x_{n-1} + \frac{h}{2}k_1, u\right)$$
$$k_3 = f\left(x_{n-1} + \frac{h}{2}k_2, u\right)$$
$$k_4 = f(x_{n-1} + hk_3, u)$$

$$x_n = x_{n-1} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Figure 11: Runge-Kutta Numeric Integration Approach

In this project, due to its ability to provide more accurate results and perform integration with more frequent time steps, the system has been discretized using the Runge- Kutta numerical integration approach.

Compared to the Euler approach, the differential system solved with the Runge-Kutta approach has successfully followed the desired path, as seen in Figure 12.

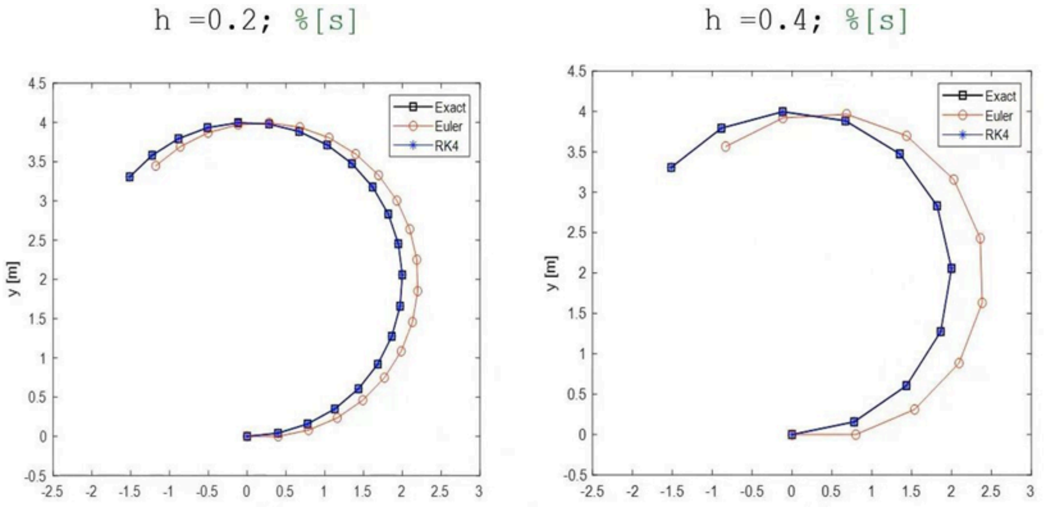
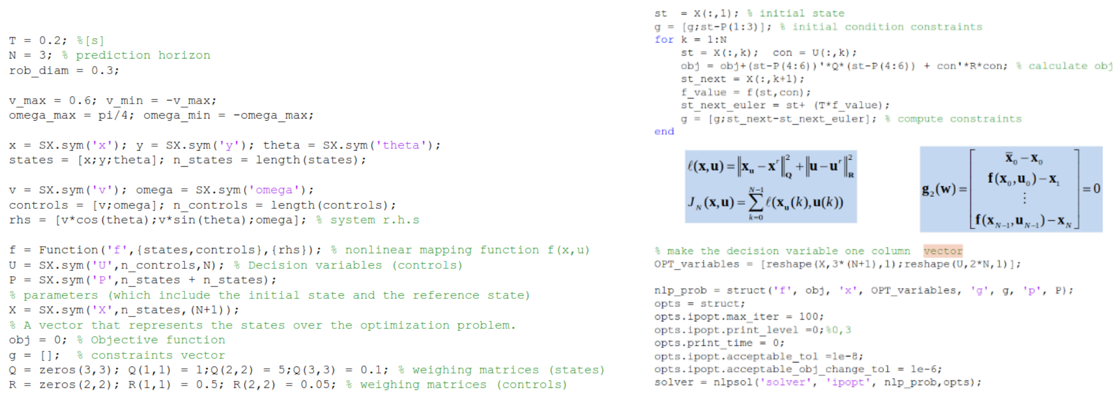


Figure 12: Runge Kutta and Euler Approach Comparison

Taking this approach into account, the code for the autonomous vehicle's desired path tracking has been written in the MATLAB environment.

Matlab refferances

The SX data type is used to represent matrices whose elements consist of symbolic expressions made up by a sequence of unary and binary operations.



```
T = 0.2; %[s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial state and the reference state)
X = SX.sym('X',n_states,(N+1));
% A vector that represents the states over the optimization problem.
obj = 0; % Objective function
g = [];  % constraints vector
Q = zeros(3,3); Q(1,1) = 1;Q(2,2) = 5;Q(3,3) = 0.1; % weighing matrices (states)
R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.05; % weighing matrices (controls)
```

```
st  = X(:,1); % initial state
g = [g;st-P(1:3)]; % initial condition constraints
for k = 1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end
```

$$\ell(\mathbf{x},\mathbf{u}) = \left\|\mathbf{x_u} - \mathbf{x}^r\right\|_{\mathbf{Q}}^2 + \left\|\mathbf{u} - \mathbf{u}^r\right\|_{\mathbf{R}}^2$$

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x_u}(k),\mathbf{u}(k))$$

$$\mathbf{g}_2(\mathbf{w}) = \begin{bmatrix} \bar{\mathbf{x}}_0 - \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0,\mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1},\mathbf{u}_{N-1}) - \mathbf{x}_N \end{bmatrix} = 0$$

```
% make the decision variable one column  vector
OPT_variables = [reshape(X,3*(N+1),1);reshape(U,2*N,1)];

nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;
solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

Figure 13: Matlab Refferances

2.2 Obstacle Detection

Obstacles detected with Lidar laser scan data relative to the vehicle's frame are defined according to the global map using the tf2 package provided by ROS. These obstacles are filtered and grouped based on their size and data density. In the obstacle tracker package, each grouped object is assigned an ID, which is then tracked using a Kalman filter. This allows us to determine whether these grouped objects are stationary or moving[9].,

In this scenario we propose to approximate the obstacles from laser scan with one of two geometric models:

line segment l represented with two extreme points:

l , {~p1, ~p2} = {(x1, y1), (x2, y2)}

circle c represented with a radius and a central point:

c , {r, ~p0} = {r, (x0, y0)}

Approximation of point sets with such geometric models provides two major advantages. Firstly, it simplifies data structures because instead of finite number of discrete points we obtain a short representation of continuous object. Secondly, it reduces the measurement noise due to the averaging effect while computing model state. Similar geometric models (with the addition of arc segments) were used in work [19], in which the authors focus on more accurate determination of the environment geometry. The mentioned work provides a comprehensive survey on methods for detecting objects from LRFs.

We define the resulting set of obstacles O as:

O , {L, C},

where L denotes a set of NL segment-type obstacles and C denotes a set of NC circular obstacles. The following subsections describe the details of obstacle detection process

2.2.1 Obstacle Detection Module (obstacle_extractor)

If the sensor_msgs/Laserscan data from the laserscan is within a certain proximity, these data points are grouped in the obstacle_extractor algorithm. In the grouped clusters, objects exceeding a certain radius are classified as static, while the others are classified as dynamic and symbolized as circles. Dynamic objects are further processed through specific filters to identify obstacles[10].

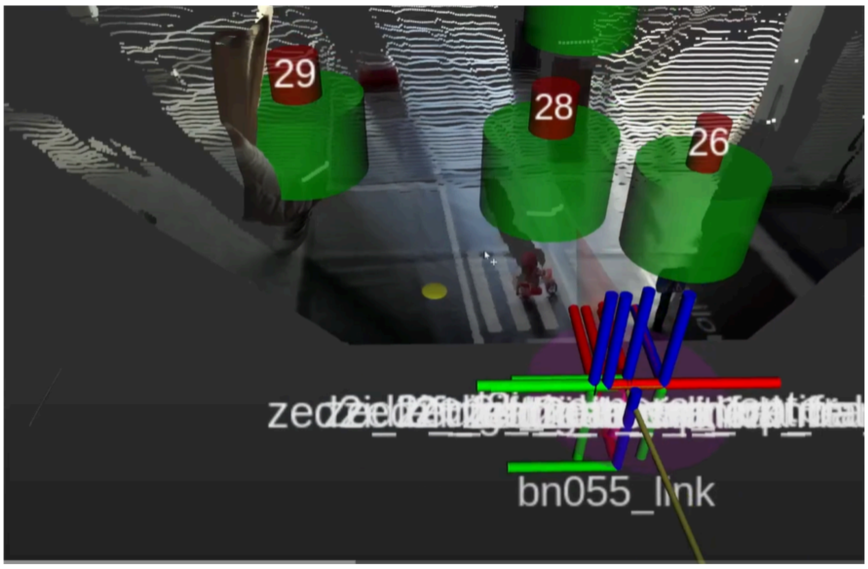


Figure 18: Obstacle Detection

2.2.2 Obstacle Tracker (obstacle_tracker)

For convenience, we will refer to obstacles detected in sample k − 1 as old obstacles and to obstacles detected in sample k as new obstacles. For solving this correspondence between points assignment problem, Hungarian method has been used in this work. [9]

Essentially, the Kalman filter uses the vehicle's position information and the known ID of the obstacle along with the previous position information of the obstacle. It incorporates the vehicle's next position and the time elapsed to reach that position, combined with the approximate data of the obstacle's position at the vehicle's current location. This process yields the orientation prediction result of the dynamic obstacle as the output. Through the Kalman filter, we can determine the direction and speed at which the obstacle is moving or if it is stationary.

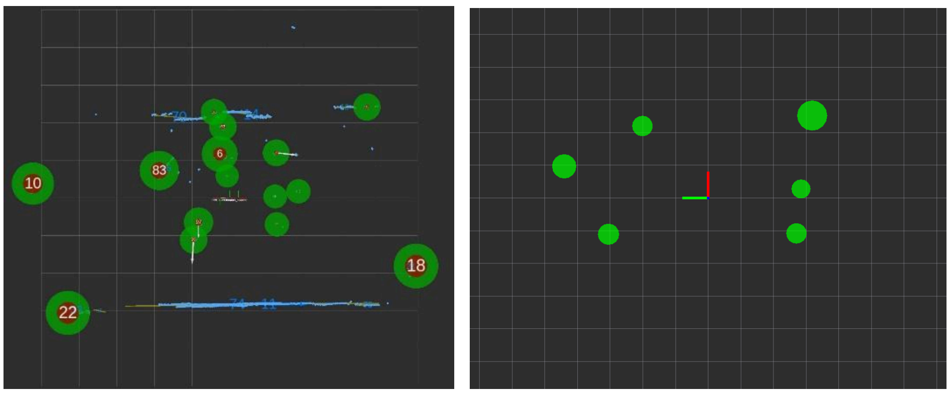


Figure 19

2.2.3 Obstacle Avoidance

The data coming from the obstacle_detection is used in cost_function of casadi solver.

- **Adding an obstacle as a path constraint**

We would like to maintain lower bound for the Euclidean distance between the prediction of the robot position and the obstacle position. Therefore, we need to impose the following path constraints

$$\sqrt{(x-x_o)^2+(y-y_o)^2} \ge r_r + r_o$$

$$\sqrt{(x-x_o)^2+(y-y_o)^2} - (r+r_o) \ge 0$$

$$-\sqrt{(x-x_o)^2+(y-y_o)^2} + (r+r_o) \le 0$$

Then, we need to ensure the above inequality constraint for all prediction steps.

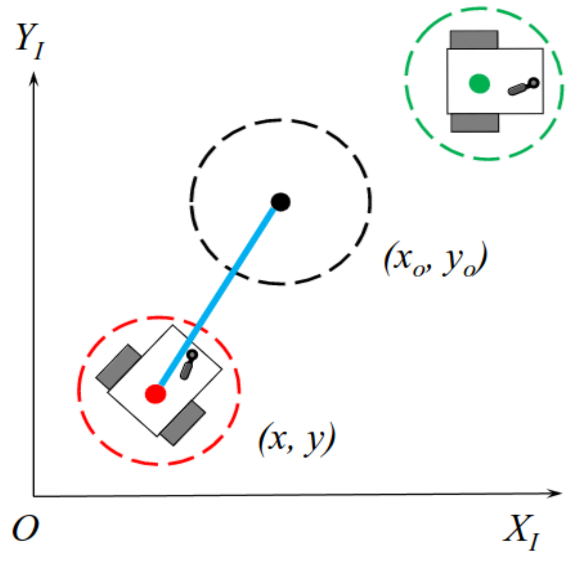


Figure 20: Path Constraint

2.3 Dijsktra

Dijsktra is a graph search algorithm for finding the shortest path between nodes with a positive edge. It is separated from the A* algorithm by not using the heuristic function [30]. Starting the calculation from the root node, the route with the lowest cost is selected, excluding routes with high-costs. The Dijkstra algorithm uses "infinite" for nodes that are marked as unvisited, and the starting point is marked as "0" (*qinit*). From the current node, the temporary distance is calculated to all the unvisited neighbors. If the newly calculated distance is shorter than the previously saved temporary distance, this is selected as a new 4 node. By linking the shorter distances between the nodes, the most cost-effective path in the search area given between the start and the target is determined. The algorithm’s pseudo-code is given below[7].

```
Dijkstra (Graph G, Vertex s)
Initialize (G,S);
Priority_Queue minQ={all vertices in V };
while (minQ ≠ Ø) do
        Vertex u=ExtractMin(minQ);//minimum est(u);
        for (each v ϵ minQ such that (u,v) ϵ E
        Relax (u,v);
        end for
end while
```

We use Dijsktra algorithm to find the shortest path and while doing that we use Bezier algorithm to prevent sharp turns.

2.3.1 Bezier algorithm

We integrate Bezier into our Dijkstra algorithm to prevent sharp turns that occur when a route is created with the Dijkstra algorithm.

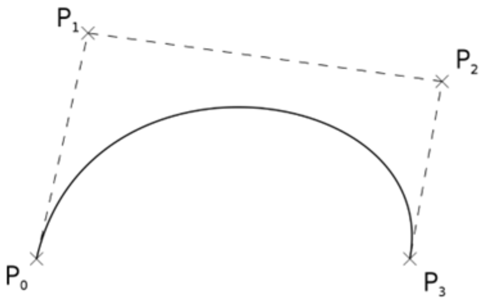
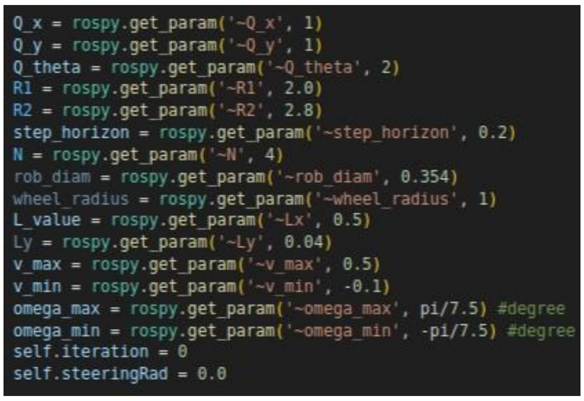


Figure 14: Bezier Algorithm

2.4 Intersection Algorithm

After the intersection sign is seen, the parameters of the NMPC algorithm change. In this situation, penalties for oscillations increase and the speed limit of the vehicle is reduced.

The same text and reference image used for the intersection are also applied to the parking, stop, and obstacle algorithms.



```
Q_x = rospy.get_param('~Q_x', 1)
Q_y = rospy.get_param('~Q_y', 1)
Q_theta = rospy.get_param('~Q_theta', 2)
R1 = rospy.get_param('~R1', 2.0)
R2 = rospy.get_param('~R2', 2.8)
step_horizon = rospy.get_param('~step_horizon', 0.2)
N = rospy.get_param('~N', 4)
rob_diam = rospy.get_param('~rob_diam', 0.354)
wheel_radius = rospy.get_param('~wheel_radius', 1)
L_value = rospy.get_param('~Lx', 0.5)
Ly = rospy.get_param('~Ly', 0.04)
v_max = rospy.get_param('~v_max', 0.5)
v_min = rospy.get_param('~v_min', -0.1)
omega_max = rospy.get_param('~omega_max', pi/7.5) #degree
omega_min = rospy.get_param('~omega_min', -pi/7.5) #degree
self.iteration = 0
self.steeringRad = 0.0
```

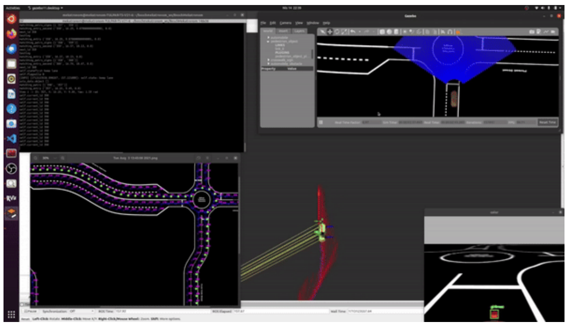


Figure 15:Parameter change codes for the Intersection Algorithm

2.5 Park Algorithm

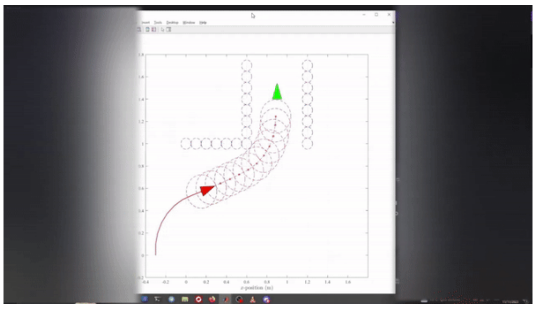
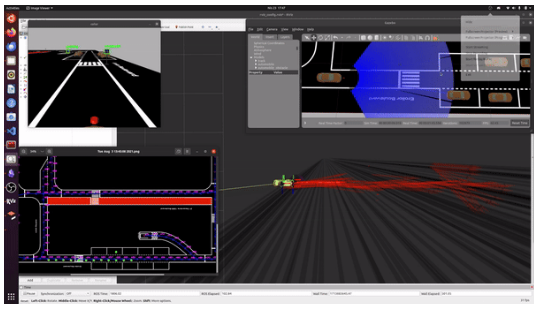



Figure 16

2.6 Bus-Stop Algorithm

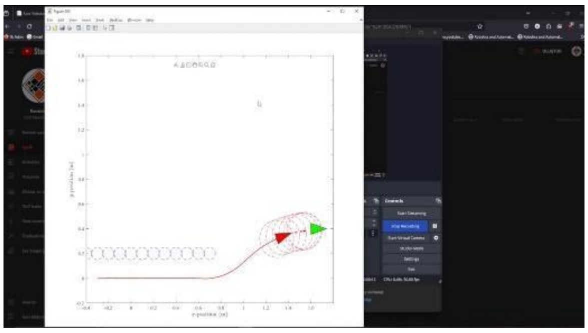


Figure 17: Bus-Stop Algorithm

2.4 YOLO V8

YOLO V8 is used for the recognition of signs. YOLO is an object recognition algorithm developed by Ultralytics. The model, which started development in 2015, is generally used for real-time object detection. Due to its speed and high performance, we used YOLO V8, which was released in January 2023, to train our dataset.

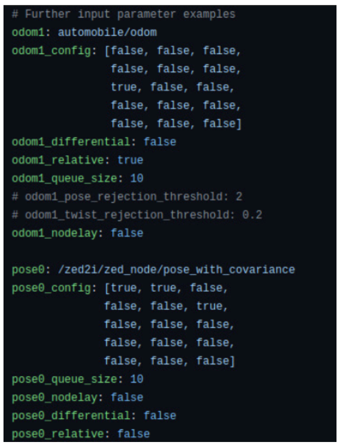
YOLO V8 is a useful model for obtaining results. YOLO V8 is divided into versions 'n', 's', 'm', 'l', and 'x'. The main differences among them are the increasing complexity and the corresponding increase in accuracy. Naturally, as complexity increases, the speed of the model decreases. Considering these factors, we determined that the most suitable model for us is YOLO V8X. Initially prepared in the "PyTorch" format, the model is planned to be converted first to the "onnx" format and then to the "TensorRT" model, i.e., the "engine" format, to reduce power loss[8].

2.4.1 Preparing Dataset for Yolo V8 Model

There are many signs prepared for various traffic scenarios. The necessary dataset for the model was created by taking images from outdoor environments, screenshots from online maps, and pictures of miniature signs we have. The images were collected and labeled according to the types of signs. To diversify the dataset, methods such as adding black dot effects and blurring were applied to the images. The purpose here is to ensure that the model performs more accurately in scenarios where the images might not be clear.

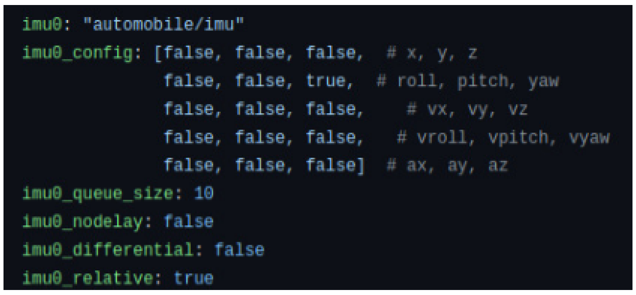
2.5 Localization

For localization, the ZED2i camera, RTK GPS Here 3, encoder, and IMU data will be used together with the robot_localization package[11].



```
# Further input parameter examples
odom1: automobile/odom
odom1_config: [false, false, false,
               false, false, false,
               true, false, false,
               false, false, false,
               false, false, false]
odom1_differential: false
odom1_relative: true
odom1_queue_size: 10
# odom1_pose_rejection_threshold: 2
# odom1_twist_rejection_threshold: 0.2
odom1_nodelay: false

pose0: /zed2i/zed_node/pose_with_covariance
pose0_config: [true, true, false,
              false, false, true,
              false, false, false,
              false, false, false,
              false, false, false]
pose0_queue_size: 10
pose0_nodelay: false
pose0_differential: false
pose0_relative: false
```



```
imu0: "automobile/imu"
imu0_config: [false, false, false,  # x, y, z
              false, false, true,  # roll, pitch, yaw
              false, false, false,    # vx, vy, vz
              false, false, false,   # vroll, vpitch, vyaw
              false, false, false]  # ax, ay, az
imu0_queue_size: 10
imu0_nodelay: false
imu0_differential: false
imu0_relative: true
```

Figure 21:

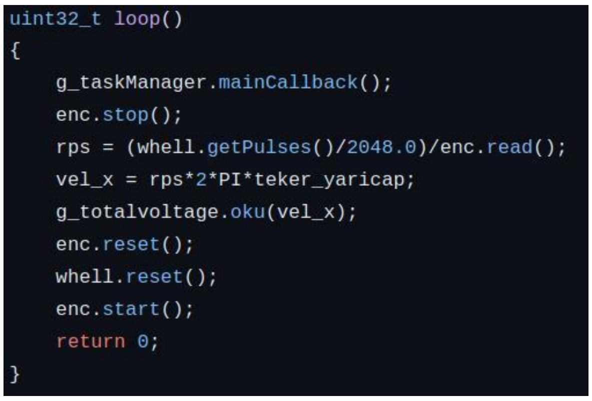
We have actively used the robot_localization package for the international competition.

2.5.1 ZED2i

ZED2i provides us with posewithcovariance data generated through its internal processes of visual odometry and IMU fusion[12].

2.5.2 Encoder

The encoder pulse information read via UART is converted to revolutions per second (RPS), and with the wheel radius, the speed information of the robot is determined[13].



```
uint32_t loop()
{
    g_taskManager.mainCallback();
    enc.stop();
    rps = (whell.getPulses()/2048.0)/enc.read();
    vel_x = rps*2*PI*teker_yaricap;
    g_totalvoltage.oku(vel_x);
    enc.reset();
    whell.reset();
    enc.start();
    return 0;
}
```

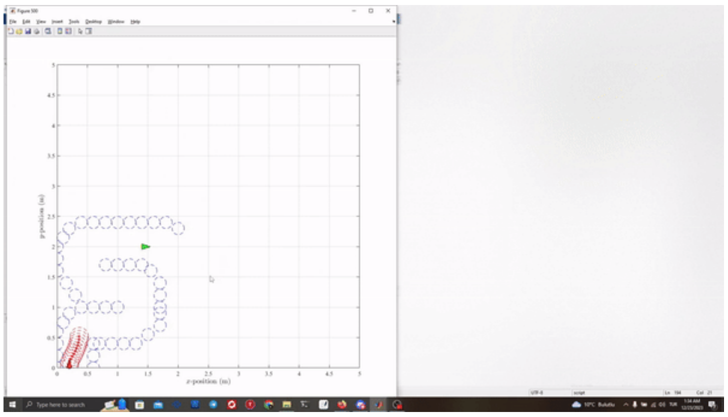
Figure 22: Encoder Calculations

2.6 Line Tracking Integration

The sliding windows technique for line detection provides a more robust lane detection compared to techniques directly using contours and Hough lines, especially in cases where the lines are not clearly visible, in areas with glare, or with dashed lines.

The method involves detecting white lines using contours and dividing the screen into a certain number of windows around these contour centers. These windows visually represent the contour centers. The approximate position of the lane can be determined by fitting a second-degree polynomial through these contour centers.

Once the positions of both lanes are determined, the midpoint of the lanes is calculated, and a line is drawn through this point. The angle of this line is used to calculate the angles of the vehicle's wheels. The length of the drawn line, i.e., the distance of the midpoint of the lanes from the vehicle, varies with the vehicle's speed. If one of the lanes cannot be detected, the position of the undetected lane can be estimated based on the detected lane's position relative to the vehicle, using curve fitting. This allows the vehicle to continue behaving as if there are two lanes ahead of it [14][15][16].



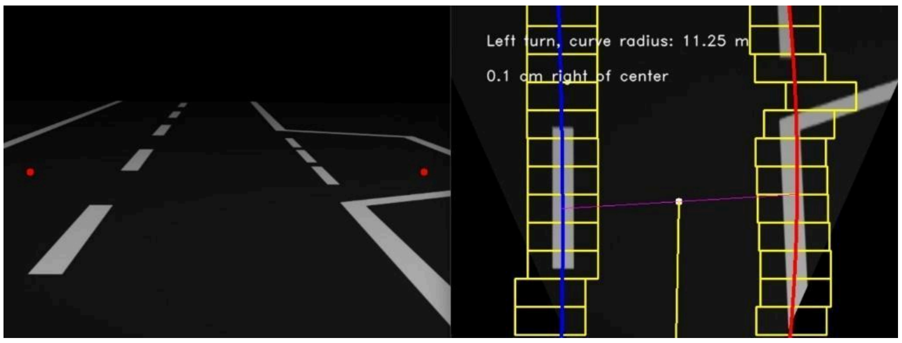


Figure 23: Sliding Window Double Lane

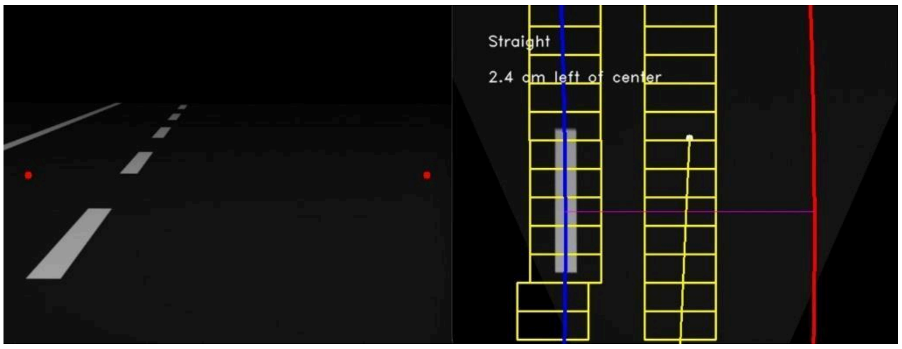


Figure 24: Sliding Window Single Lane

3. CONCLUSIONS AND FUTURE STUDIES

Proposed algorithms are developed for the international competation named as BOSCH FUTURE MOBILITY CHALLENGE. This has been tested in real life and we are able to see the results are good enough. For future studies instead of PCL for obstacle detection should be used directly 2D lidar and for the localization next step would be use gmapping as sector does. For controller part node base system is limited so instead of creating the path with MPC we directly give the path and try to stick at the path for more speed.

4. REFERENCES

[1] https://bosch-future-mobility-challenge-competition-regulation.readthedocs-hosted.com/

[2] https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/self-driving-cars/

[3] https://github.com/Renbago/autonomus_vehicle

[4] https://scholar.google.com/citations?user=aRNeH4QAAAAJ&hl=en

[5] Mehrez, M. Github. "MPC and MHE implementation in Matlab using CasADi." (2019).

[6] Mohamed W. Mehrez, PhD, Optimization based Solutions for Control and State Estimation in Dynamical Systems (Implementation to Mobile Robots) A Workshop, Ocak 2019.

[7] Mahmut Dirik, A. Fatih Kocamaz, RRT- Dijkstra: An Improved Path Planning Algorithm for Mobile Robots

[8] https://github.com/ultralytics/ultralytics

[9] https://www.researchgate.net/publication/319051656_Detection_and_tracking_of_2D_geometric_obstacles_from_LRF_data

[10] https://github.com/tysik/obstacle_detector

[11] http://docs.ros.org/en/melodic/api/robot_localization/html/index.html

[12] https://www.stereolabs.com/docs/ros

[13] https://docs.cubepilot.org/user-guides/here-3/here-3-manual

[14] https://github.com/charleswongzx/Advanced-Lane-Lines

[15] https://www.researchgate.net/publication/358912939_Lane_Vehicle_Detection_and_Tracking_Algorithm_Based_on_Sliding_Window

[16] https://www.semanticscholar.org/paper/Lane-Detection-for-Track-following-Based-on-He-Sun/7e3d08f98b5d10d4961844f45feba516c3f4d2b2