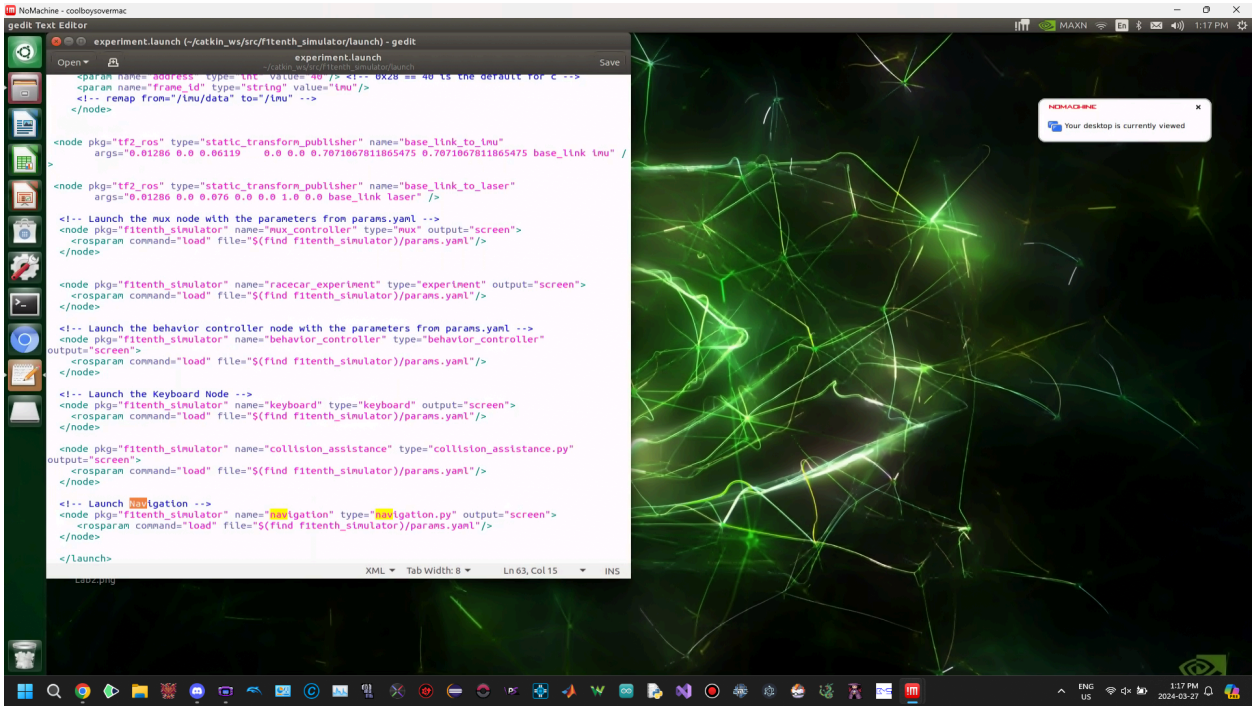
EE 3EY4 Lab 9: Autonomous Driving Using Virtual Separating Barriers

03/04/2024

Mohammad Bishr Hamideh - 400408676
Abdurahman Bade - 400247875
Ameen Elmankabady - 400398578

Activity 1: You have been provided with the incomplete file "navigation_lab9_inc.py". Copy the content of this file into your "navigation.py". Also, update your local copy of the file "param.yaml" using the one posted on Avenue. Make sure that you retain the calibrated VESEC parameters from Lab 6 and the value of "scan_beams" that corresponds to the LiDAR in BOOST mode on your AEV. Also, copy the files "hallway_w_blocks.yaml" and "hallway_w_blocks.pgm" from Avenue to the "maps" subdirectory of the f1tenth_simulator package. Modify the file "simulator.launch" accordingly to load this map for the simulations.



Activity 2: Study the files "navigation_lab9_inc.py" and "params.yaml" and identify parts of the code that accomplish this step.

Within the Python code for navigation, identifying paths of safe travel is primarily achieved through the functions preprocess_lidar(self, ranges) and find_max_gap(self, proc_ranges), which leverage data from Lidar scans. These functions require parameters such as the frequency of Lidar scans, the scope of the scans' field of view, the range of distances covered, and the maximum allowed

steering angle. Together, these parameters and functions play a crucial role in establishing a clearly defined zone of safety.

Activity 3: Explain the role of function “preprocess_lidar” and how it helps accomplish the objective of Step 2.

The `preprocess_lidar` function plays a critical role in the obstacle detection and navigation algorithm by processing LiDAR data to identify immediate obstacles within a specified safety zone in front of the vehicle. Specifically, it sets to zero the range values for all LiDAR returns that fall within this safety distance, effectively marking those points as immediate obstacles. This preprocessing step is essential for differentiating between free space (non-zero LiDAR returns) and obstacles (zero range values), which is crucial for subsequent steps in the algorithm, such as identifying the largest gap in front of the vehicle through which to navigate.

Activity 4: Complete the missing parts in the function “find_max_gap” to achieve the objective of Step 3.

```
# Return the start and end indices of the maximum gap in free_space_ranges
def find_max_gap(self, proc_ranges):

    j=0
    str_indx=0;end_indx=0
    str_indx2=0;end_indx2=0

    range_sum = 0
    range_sum_new = 0

    for i in range (self.ls_len_mod):

        if proc_ranges[i,0]!=0:
            if j==0:
                str_indx=i
                range_sum_new = 0
                j=1
            range_sum_new = range_sum_new + proc_ranges[i,0]
            end_indx=i

        if  j==1 and (proc_ranges[i,0]==0 or i==self.ls_len_mod-1):

            j=0

            if  range_sum_new > range_sum:
                    end_indx2= end_indx
                    str_indx2= str_indx
                    range_sum = range_sum_new

    return str_indx2, end_indx2
```

Activity 5: Complete the code for the function "find_best_point" to compute the desired direction of movement according to the formulation in (2). Explain why this might be a better choice than the furthest point in the largest gap in (1).

```
# start_i & end_i are the start and end indices of max-gap range, respectively
# Returns index of best (furthest point) in ranges
def find_best_point(self, start_i, end_i, proc_ranges):

    range_sum = 0
    best_heading = 0

    for i in range (start_i,end_i + 1):
        range_sum = range_sum + proc_ranges[i,0] ###
        best_heading = best_heading + proc_ranges[i,0]*proc_ranges[i,1] ###

        if range_sum != 0:
            best_heading = best_heading/range_sum


    return best_heading
```

Computing the desired direction of movement as the weighted average of LiDAR beam angles (according to equation (2) in the document) takes into account both the direction and distance of all points within the largest gap, rather than just aiming towards the furthest point. This method provides a more nuanced approach to navigation by considering the overall configuration of free space in front of the vehicle. It helps to smooth the vehicle's trajectory, especially in scenarios where the largest gap might have varying distances at different angles. This approach can lead to a safer and more efficient navigation path, as it aims to keep the vehicle centered in the largest available space, reducing the likelihood of abrupt direction changes or getting too close to obstacles.

Activity 6: Derive the optimization formulation in (7).

$$L(W, \lambda_i) = \frac{1}{2} W^T * W + \lambda_i (W^T p_i + 1)$$

$$\frac{\delta_L}{\delta W} = W + \lambda_i = 0$$

$$(-\lambda_i p_i)^T + 1 \leq 0$$

$$\frac{-1}{\left| |pi|^2 \right|} \leq \lambda_i$$

$$\frac{1}{2} (-\lambda_i p_i)^T * (-\lambda_i p_i) = \frac{1}{2} \lambda^2{}_i p_i{}^T p_i$$

Activity 7: Explain how the value of $0 \leq \alpha \leq 1$ affects the solution to the optimization problem in (8). Consider what happens as α changes between the two extreme values of $\alpha = 0$ and $\alpha = 1$.

The value of α in the optimization problem in equation (8) acts as a regularization parameter that balances two objectives: maximizing the distance of the separating barrier to the origin (vehicle) and minimizing the change in the virtual barrier between consecutive time steps. When α=0, the optimization prioritizes maximizing the distance of the separating line to the origin, potentially leading to rapid changes in the barrier's position over time. As α approaches 1, greater emphasis is placed on minimizing the

change in the barrier's position, encouraging smoother transitions but possibly at the expense of having the barrier closer to the vehicle. Thus, α controls the trade-off between barrier stability and distance maximization, affecting the vehicle's ability to navigate smoothly while maintaining a safe distance from obstacles.

Activity 8: Show that the following optimization problem has the same solution as the one in (8). We call these two optimization problems equivalent.

$$\frac{1}{2}\alpha w^T{}_k w_k + \frac{1}{2}(1 - \alpha)(w_k - w_{k-1})^T(w_k - w_{k-1})$$
$$= \frac{1}{2}\alpha w^T{}_k w_k + \frac{1}{2}(1 - \alpha) w_k{}^T w_k - \frac{1}{2}(1 - \alpha) w^T{}_{k-1} w_k - \frac{1}{2}(1 - \alpha) w_k{}^T w_{k-1}$$
$$+ \frac{1}{2}(1 - \alpha) w^T{}_{k-1} w_{k-1}$$

Activity 9: Derive the IQP formulation in (9). Explain the rational for the additional constraints on the value of *b*.

$$d = d_r + d_l = \frac{2}{\sqrt{w^T w}}$$

$$d = \frac{2\sqrt{w^T w}}{w^T w}$$

$$\frac{\sqrt{w^T w}}{d} = \frac{w^T w}{2}$$

To increase d, we reduce the right-hand side of the equation

$$min \frac{1}{2} w^T w$$

$$0 \leq w^T p_i + b - 1$$

$$0 \geq w^T p_i + b + 1$$

Activity 10: Install the Pytrhon QP solver quadprog using the following command:
$ sudo pip install quadprog

- The Pytrhon QP solver quadprog was successfully installed.

Activity 11: Consider the optimization formulations for obtaining the virtual line barriers in (9) and (20), and the standard form for the quadprog given in (21). Identify the corresponding variables and complete the missing code in the function "getWalls" to find the solution for the problem in (9) using quadprog solver.

- Refer to the screenshot in Activity 12.

Activity 12: Consider the optimization formulation for obtaining the parallel virtual line barriers in (21) and the standard form for the quadprog given in (15). Identify the corresponding variables and complete the missing code in the function "getWalls" to find the solution for the problem in (20) using quadprog solver.

```
    def getWalls(self, left_obstacles, right_obstacles, wl0, wr0, alpha):


        if self.optim_mode == 0:

            Pr = np.array([[1.0, 0], [0, 1.0]]) ###
            Pl = np.array([[1.0, 0], [0, 1.0]])  ###

            bl = np.full((self.n_pts_l),1.0,dtype=np.float64)
            br = np.full((self.n_pts_r),1.0,dtype=np.float64)

            Cl= -(left_obstacles.T)
            Cr= -(right_obstacles.T)
            al = (1-alpha)*wl0
            ar = (1-alpha)*wr0


            wl = solve_qp(Pl.astype(np.float), al.astype(np.float), Cl.astype(np.float),  bl.astype(np.float), 0)[0]
            wr = solve_qp(Pr.astype(np.float), ar.astype(np.float), Cr.astype(np.float),  br.astype(np.float), 0)[0]

        else:

            P = np.array([[1.0,0,0],[0,1.0,0],[0,0,0.0001]])

            bl = np.full((self.n_pts_l),1.0,dtype=np.float64) ###
            br = np.full((self.n_pts_r),1.0,dtype=np.float64) ###
            b = np.concatenate((br,bl,np.array([-0.99, -0.99])))

            Cl= -(left_obstacles.T) ###
            Cr= -(right_obstacles.T) ###
            C1 = np.vstack((-Cr,br))
            C2 = np.vstack((Cl,- bl))
            C =np.hstack((C1,C2))
            C =np.hstack((C,np.array([[0,0],[0,0],[1.0,-1.0]])))

            a = np.zeros(3) ###

            ws  = solve_qp(P.astype(np.float), a.astype(np.float), C.astype(np.float),  b.astype(np.float), 0)[0]

            wr = np.array([ws[0]/(ws[2]-1),ws[1]/(ws[2]-1)]) ###
            wl = np.array([ws[0]/(ws[2]+1),ws[1]/(ws[2]+1)]) ###

        return  wl, wr
```

Activity 13: Complete the missing parts in the above equations and the corresponding code in "navigation_lab9_inc.py".

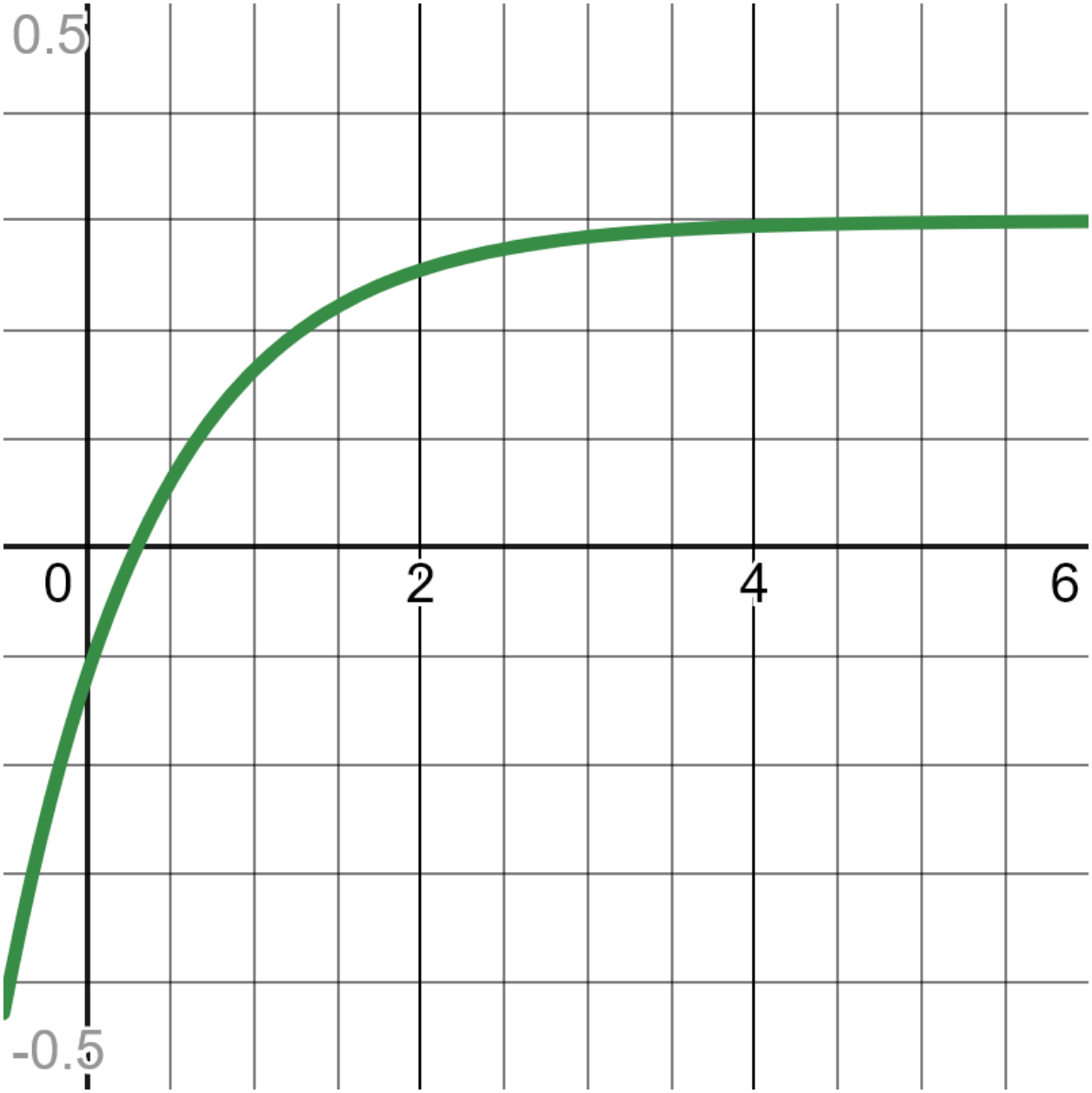
```
        obstacle_points_l[k][0]= - obs_range*math.cos(mod_angle_al+k*index_l*data.angle_increment)
        obstacle_points_l[k][1]= - obs_range*math.sin(mod_angle_al+k*index_l*data.angle_increment)

    for k in range(0,self.n_pts_r):

        obs_index = (start_indx_r+k*index_r) % self.scan_beams
        obs_range= data.ranges[obs_index]
        if obs_range >=self.max_lidar_range:
            obs_range = self.max_lidar_range

        obstacle_points_r[k][0]= - obs_range*math.cos(mod_angle_br+k*index_r*data.angle_increment)
        obstacle_points_r[k][1]= - obs_range*math.sin(mod_angle_br+k*index_r*data.angle_increment)


    alpha = 1-math.exp(-dt/self.tau)

    wl, wr = self.getWalls(obstacle_points_l, obstacle_points_r, self.wl0, self.wr0, alpha)

    self.wl0 = wl
    self.wr0 = wr

    dl = 1/math.sqrt(np.dot(wl.T,wl)) ###
    dr = 1/math.sqrt(np.dot(wr.T,wr)) ###

    wl_h = wl*dl  ###
    wr_h = wr*dr  ###

    self.marker.header.frame_id = "base_link"
    self.marker.header.stamp = rospy.Time.now()
    self.marker.type = Marker.LINE_LIST
    self.marker.id = 0
    self.marker.action= Marker.ADD
    self.marker.scale.x = 0.1
    self.marker.color.a = 1.0
    self.marker.color.r = 0.5
    self.marker.color.g = 0.5
    self.marker.color.b = 0.0
    self.marker.pose.orientation.w = 1

    self.marker.lifetime=rospy.Duration(0.1)


    self.marker.points = []

    line_len = 1
    self.marker.points.append(Point(dl*(-wl_h[0]-line_len*wl_h[1]), dl*(-wl_h[1]+line_len*wl_h[0]) , 0))
    self.marker.points.append(Point(dl*(-wl_h[0]+line_len*wl_h[1]), dl*(-wl_h[1]-line_len*wl_h[0]) , 0))
    self.marker.points.append(Point(dr*(-wr_h[0]-line_len*wr_h[1]), dr*(-wr_h[1]+line_len*wr_h[0]) , 0))
    self.marker.points.append(Point(dr*(-wr_h[0]+line_len*wr_h[1]), dr*(-wr_h[1]-line_len*wr_h[0]) , 0))
    self.marker.points.append(Point(0, 0 , 0))
    self.marker.points.append(Point(line_len*math.cos(heading_angle), line_len*math.sin(heading_angle), 0))

    self.marker_pub.publish(self.marker)
```

Activity 14: Plot the vehicle velocity *vs*as a function of *dmin* and use it to explain what (28) is trying to accomplish. Identify the relevant sections in the code and the parameters in “params.yaml”. Explain how different parameters in (28) can impact the vehicle commanded velocity, *vs* .

Velocity zero = 0.3
d_{min} = minimum distance
Stop_distance = d_{stop} = 0.3
stop_distance_decay =0.9

V_S = vehicle commanded velocity

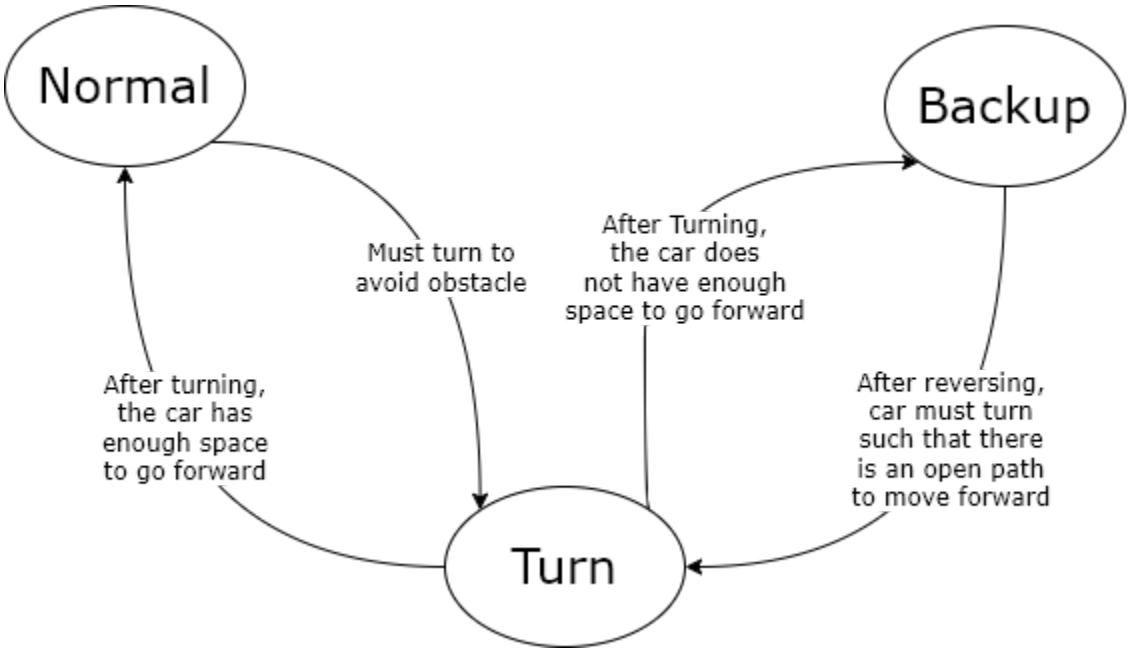
Therefore our equation is: $V_S = V_0(1 - e^{\frac{d_{stop} - d_{min}}{Stop\ Distance\ Decay}}) = (0.3)(1 - e^{\frac{0.3 - d_{min}}{0.9}})$



Enhancing Vs0 results in an elevated peak value for Vs. Elevating the value of d_stop causes the vehicle to initiate braking earlier, given that d_stop is reached more frequently. A higher d_dec means the vehicle will slow down more gently.

Increasing d_min causes the vehicle to move faster because it is positioned further from the d_stop threshold

Activity 15: The “back-up and turn” functionality is implemented using a finite-state machine with three states: “normal”, “backup” and “turn”. Carefully examine the Python code in “navigation_lab9_inc.py” for the implementation of this functionality. Draw the finite state machine diagram with all necessary details to explain the vehicle operation. You are encouraged to suggest (and implement) improvements to this functionality as you see fit.



The car initiates in normal mode, it will persist in forward motion until it senses an obstacle in front of it. It will switch to turn mode, maneuvering at a turn velocity. If the car successfully adjusts its direction to evade the obstacle, it will revert to normal mode until it encounters the next obstacle. However, if the car faces an obstacle that cannot be avoided solely by turning, it will transition from turn mode to backup mode, reversing its direction. After distancing itself from the obstacle, it will re-enter turn mode to attempt avoidance once more. If further turning is required, the car will cycle back into backup mode before returning to turn mode. Once the required adjustment is achieved, the car will resume normal mode.

Activity 16: Launch the f1tenth_simulator “simulator.launch” file. Add a display of Marker type with the topic “wall_markers” to the “rviz” visualization

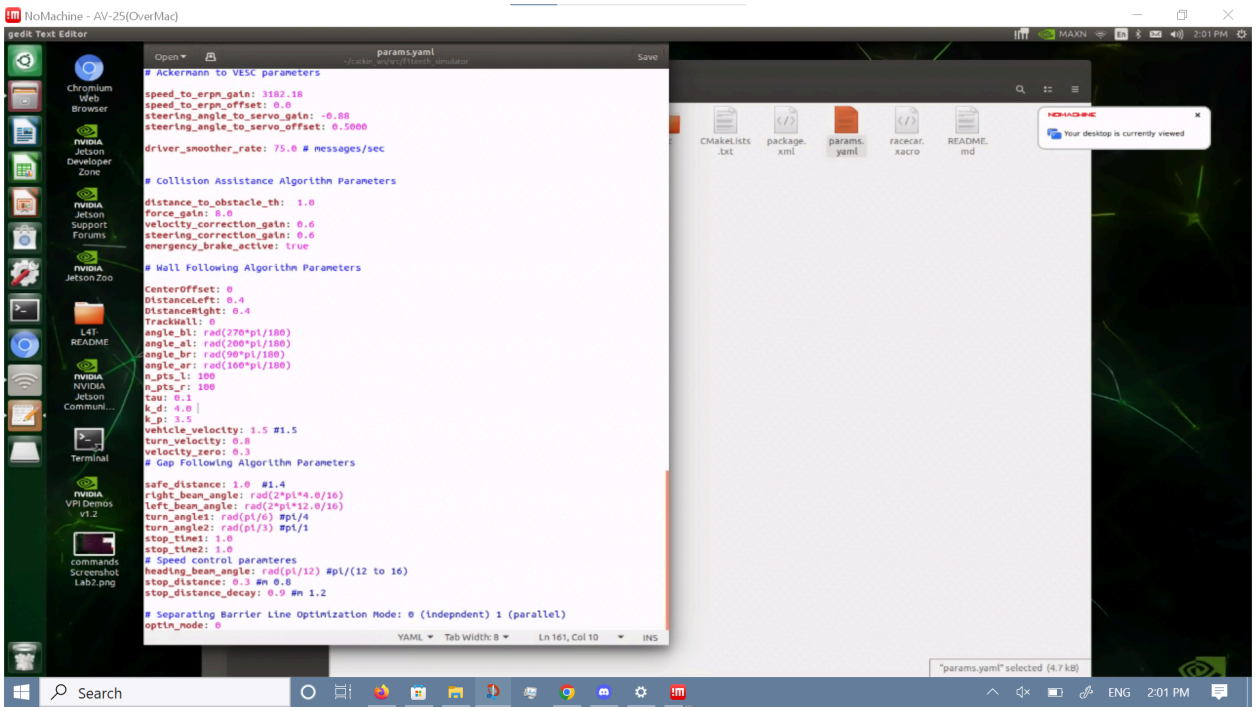
environment. These markers represent the virtual separating line barriers and desired heading direction of the vehicle. Use the simulator environment to fine-tune the values of the control algorithm parameters until you achieve a satisfactory response in the simulation environment. You must try both formulations for finding the virtual separating line barriers in (9) and (2). Note that the value of the parameter “optim_mode” determines which formulation is used. Pay close attention to the “wall_markers” as you drive the vehicle around in the simulation environment.

- A video will be attached alongside this report on A2L.

Activity 17: Compare the performance of the self-driving algorithm using the two optimization formulations in (9) and (20). Briefly report your observations on the impact of the parameters “k_p”, “k_d” and “n_pts_l”, “n_pts_r”, and “safe_distnace” on the vehicle self-driving performance.

When comparing the performance of the self-driving algorithm utilizing the two distinct optimization formulations specified in equations (9) and (20), it's evident that each has its strengths in different driving scenarios. Formulation (9) significantly improves the algorithm's efficiency in making turns, demonstrating its aptness for complex navigation tasks that involve agile maneuvering around obstacles. On the other hand, the strategy outlined in (20) is better suited for straightforward, unobstructed driving, focusing on maintaining a direct path. The influence of parameters such as "k_p" and "k_d" is crucial in this context; they dictate the vehicle's responsiveness and its ability to maintain stability, respectively. "k_p" adjusts the algorithm's sensitivity to path deviations, while "k_d" controls the tendency towards oscillation, ensuring smoother adjustments. The "n_pts_l" and "n_pts_r" parameters, which determine the number of reference points on either side of the vehicle, directly affect navigation precision. Higher values lead to improved driving accuracy by enhancing the vehicle's spatial awareness, whereas lower values, although reducing computational load, may compromise the vehicle's ability to follow its intended path closely. The "safe_distance" parameter, which sets the minimum distance from obstacles, is another critical factor; a smaller "safe_distance" can lead to more cautious driving and slower speeds, suitable for navigating through tight spaces but potentially

increasing collision risks if not carefully adjusted. Altogether, selecting the appropriate optimization formulation and adjusting these parameters is vital for optimizing the self-driving algorithm's performance across varied environments, striking a balance between agility, safety, and precision.



Activity 18: After you are satisfied with the performance of the self-driving controller in the simulations, you must try it on the actual vehicle. Before starting, set the value of “optim_mode” to select the right optimization mode for your experiment. Make sure the environment around the vehicle is free of people and safe for vehicle operation and launch the file “experiment.launch”. As in Lab 8, the self-driving mode can be activated and deactivated at by toggling the “RB” on Joystick. Note that pressing the “LB” would activate the manual driving mode. You can use either of these options to stop self-driving in case of an emergency. Remember to always keep the vehicle in your reach so you would not lose the connection between the vehicle and the Joystick.

- A video will be attached alongside this report on A2L.

Activity 19: Adjust the controller parameters as you see necessary to achieve a satisfactory response. Briefly report on your observations from the experiments.

Comment on any differences that you may observe between the system responses under the two optimization modes.

- When adjusting the parameters, we noticed that just because our parameter worked for the simulation, it didn’t necessarily mean that our experiment would work. When using the simulation parameters, the car could only successfully pass through one obstacle on its first try. Afterwards, the car would always try to move in the opposite direction of where we wanted it to move. To resolve this issue, we had to adjust both the left and right turn values, the car's velocity and the stop distance.

Activity 20: Reflect on the operation of the self-driving control algorithm and suggest potential ways for improving it as you see fit.

The self-driving algorithm successfully prevented crashes and allowed the car to travel autonomously. The automobile was also able to reverse and continue traveling as well as maneuver around obstacles. However, there were moments when the automobile came to a complete stop and continued to move its wheels left and right without reversing. This mostly occurred when the car was trying to go through the accessibility pathway near the lab room. To solve this, we had to drive the car manually before turning on self-driving mode again. We want to enhance the car's speed, as it had to be reduced for simpler turning.