ELECENG 3EY4: Electrical System Integration Project

Lab08_Driver-Assist Collision Avoidance and Autonomous Driving Using Wall-Following

Junbo Wang – wangj430 – 400249823

Preet Batra - batrap5 - 400341704

Yichen Lu - luy191 - 400247938

Activity 1: Explain why this might be a reasonable assumption based on the configuration of the sensors on MacAEV. With this assumption, show the angle αi can be computed from the angle information in LiDAR scan data.

The distance between the laser frame and the base_link frame is fixed, since both shift congruently with the car's movement. The 12.86mm distance in the x direction from the IMU to the laser is trivial when compared to the vehicle's distance from obstacles. Therefore, it's reasonable to consider the two frames as having no relative movement due to their proximity. In order to determine the collective repulsive force exerted on the vehicle by obstacles, we can sum the forces generated by each individual obstacle. Given the negligible displacement between the "laser" and "base link" frames, they can be treated as effectively joined. The correct algorithm can then be employed to calculate the angle between the laser scans that detect the obstacle and the "base link" frame.

Activity 2: Derive the above expression for δ_0^{k+1} in (17). Also, explain the role of the design parameter $0 \leq \eta_\delta \leq 1$.

1. **Derive the above expression for δ_0^{k+1} in (17).**

Activity 2:

Let $\delta_{joy}^{k+1} = a$, $\delta_0^{k+1} = b$

$$\eta_\delta \frac{V_s^2}{L} \tan(a) + (1-\eta_\delta) joy = \frac{V_s^2}{L} \tan(a+b)$$

$$\eta_\delta \tan(a) + (1-\eta_\delta) \frac{L}{V_s^2} joy = \tan(a+b)$$

$$\Downarrow u_0$$

$$u_0 = \tan(a+b)$$

$$u_0 = \frac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)}$$

$$u_0 - u_0(\tan(a)\tan(b)) = \tan(a)+\tan(b)$$

$$u_0 - \tan(a) = \tan(b) + u_0(\tan(a)\tan(b))$$

$$u_0 - \tan(a) = \tan(b)(1+u_0\tan(a))$$

$$\tan(b) = \frac{u_0 - \tan(a)}{1+u_0\tan(a)}$$

$$b = a\tan\left(\frac{u_0-\tan(a)}{1+u_0\tan(a)}\right)$$

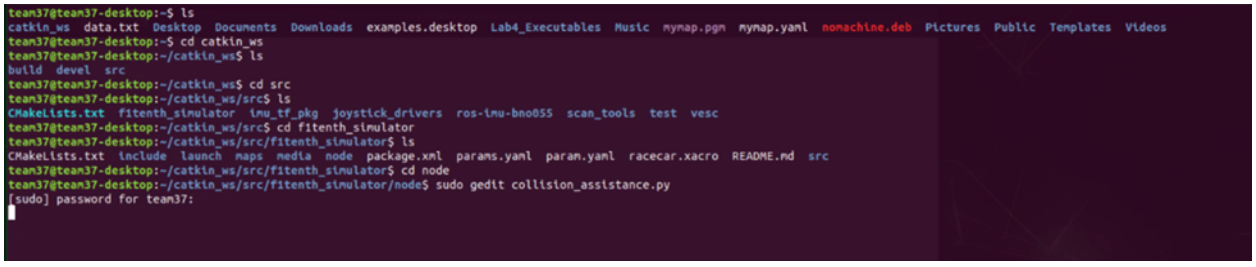
$$\delta_0^{k+1} = a\tan\left(\frac{u_0 - \tan\delta_{joy}^{k+1}}{1+u_0\tan\delta_{joy}^{k+1}}\right)$$

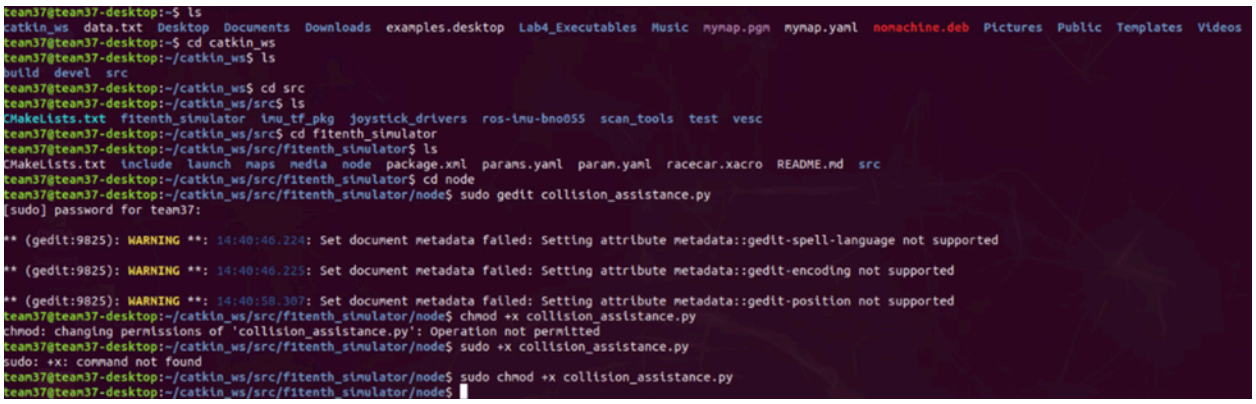
2. Explain the role of the design parameter $0 \leq \eta_\delta \leq 1$.

The balance between joystick control and the corrective action of the collision avoidance algorithm is determined by the design parameter η_δ. As the value of this parameter increases, the proportion of joystick influence on the vehicle's steering will increase as well. More specifically, when η_δ is set to 0, the steering angle will only depend on the algorithm's adjustments. Conversely, when η_δ equals 1, the steering's direction will only depend on the joystick's input.

Activity 3: Copy the content of this file into a new file named “collision_assistance.py” in the directory “node” of f1tenth_simulator software stack. Make this file executable by running the following command at the terminal:

For this activity, we just followed the steps in the lab manual. The actions that we did are shown below.





Activity 4: The following function finds potentially dangerous obstacles and their distance to the vehicle. Complete the missing code in “…” to achieve this functionality:

```
def obst_idnt(self,ls_ranges):

       indx=np.zeros((2),dtype=int)
       j=0
       nm_obs=0

       for i in range(self.ls_len_mod2):
           if ls_ranges[i,0]<=self.d_obs and i<self.ls_len_mod2-1:



               if j==0:
                   indx[0]=i
                   j=1
               indx[1]=...
           else:
               j=0
               if indx[1]-indx[0]>0:
                   self.obs_indx[nm_obs,:]=indx
                   nm_obs=nm_obs+1
                   indx[1]=0;indx[0]=0

   arg_ls_obs=np.zeros(nm_obs,dtype=int)

   for i in range(nm_obs):
       arg_ls_obs[i]=np.argmin(...,0])
       arg_ls_obs[i]=self.obs_indx[i,0]+...

   return nm_obs, arg_ls_obs
```

```
# Find unsafe obstacles and return number of unsafe obstacles
#  and the closest point of each unsafe obstacle to the vehicle
def obst_idnt(self,ls_ranges):

    indx=np.zeros((2),dtype=int)
    j=0
    nm_obs=0

    for i in range(self.ls_len_mod2):
        if ls_ranges[i,0]<=self.d_obs and i<self.ls_len_mod2-1:
            if j==0:
                indx[0]=i
                j=1
            indx[1]=i+1
        else:
            j=0
            if indx[1]-indx[0]>0:
                self.obs_indx[nm_obs,:]=indx
                nm_obs=nm_obs+1
                indx[1]=0;indx[0]=0

    arg_ls_obs=np.zeros(nm_obs,dtype=int)

    for i in range(nm_obs):
        arg_ls_obs[i]=np.argmin(ls_ranges[self.obs_indx[i,0]:self.obs_indx[i,1],0])
        arg_ls_obs[i]=self.obs_indx[i,0]+arg_ls_obs[i]

    return nm_obs, arg_ls_obs
```

The obst_idnt method processes a subset of LiDAR range measurements to identify obstacles that are closer to the vehicle than a predefined safety threshold (self.d_obs). It operates on processed LiDAR data, where each range measurement is already filtered to include only the front 180 degrees field of view and within a maximum LiDAR range. The method iterates through the LiDAR data and marks the start and end indices (indx) of consecutive measurements that are below the safety threshold, indicating the presence of an unsafe obstacle. If the subsequent range measurement exceeds the safety threshold, it resets the indices, treating it as the end of the current obstacle and potentially the beginning of a gap or a new obstacle. For each identified obstacle, the method stores its start and end indices in the array self.obs_indx and increments the counter nm_obs, which keeps track of the total number of unsafe obstacles detected. After identifying all unsafe obstacles, the method creates an array arg_ls_obs to store the indices of the closest point of each obstacle. This is done by finding the minimum range value within the indices that mark the start and end of an obstacle. The method returns two pieces of information: the number of unsafe obstacles (nm_obs) and the array of indices corresponding to the closest point of each obstacle (arg_ls_obs).

Activity 5: The function “ptn_fld” computes the total obstacles-related corrective forces acting the vehicle. Complete the missing code in “…” to achieve this functionality:

```
def ptn_fld(self, nm_obs,arg_ls_obs,ls_ranges):

    f_rep_x=0;f_rep_y=0

    for i in range(nm_obs):
        d=ls_ranges[arg_ls_obs[i],0]
        theta=ls_ranges[arg_ls_obs[i],1]

        f_rep_x=f_rep_x+self.f_gain*...
        f_rep_y=f_rep_y+self.f_gain*...

    return f_rep_x, f_rep_y
```

```
# compute corrective force acting on the vehicle
def ptn_fld(self, nm_obs,arg_ls_obs,ls_ranges):

    f_rep_x=0;f_rep_y=0

    for i in range(nm_obs):
        d=ls_ranges[arg_ls_obs[i],0]
        theta=ls_ranges[arg_ls_obs[i],1]

        f_rep_x=f_rep_x+self.f_gain*(1-self.d_obs/d)*math.cos(theta)
        f_rep_y=f_rep_y+self.f_gain*(1-self.d_obs/d)*math.sin(theta)

    return f_rep_x, f_rep_y
```

The ptn_fld (potential field) method computes repulsive forces based on the proximity of obstacles detected within a critical distance. It uses the information about the number of obstacles (nm_obs) and their closest points (arg_ls_obs) provided by the ls_ranges parameter, which contains the processed LiDAR data. For each detected obstacle, the method calculates the repulsive force components (f_rep_x and f_rep_y) in the x and y directions. These components are computed by considering the inverse of the distance to the obstacle (d), the angle (theta) of the closest point of the obstacle relative to the vehicle, and a force gain parameter (self.f_gain). The force gain parameter amplifies the force as the vehicle gets closer to the obstacle. The repulsive forces are designed to increase as the distance to an obstacle decreases, encouraging the vehicle to steer away from potential collisions. The math.cos(theta) and math.sin(theta) functions are used to resolve the force into its x and y components based on the obstacle's angle relative to the vehicle. The method returns the sum of the repulsive force components in both the x (f_rep_x) and y (f_rep_y) directions. These forces are then used to adjust the vehicle's steering and velocity commands to maintain a safe distance from obstacles.

Activity 6: Find and complete the following lines of code in the function “lidar_callback”. Explain the purpose of this part of the code:

```
vel_d= ...

    if self.vel_joy >=0 and vel_d < 0:
        vel_d = 0

    u0=self.etha_delta*math.tan(self.steer_joy)+ ...



      delta_d=math.atan((u0-
math.tan(self.steer_joy))/(1+u0*math.tan(self.steer_joy)))+self.steer_
joy
```

```
def lidar_callback(self, data):
    ranges = data.ranges
    proc_ranges = self.preprocess_lidar(ranges)
    nm_obs,arg_ls_obs=self.obst_idnt(proc_ranges)

    # Calculate the potential field forces
    f_tot_x, f_tot_y=self.ptn_fld(nm_obs,arg_ls_obs,proc_ranges)

    # Calculate desired velocity and steering angle
    self.vel_x=self.vel+f_tot_x

    vel_d= (self.etha_vel)*self.vel_joy+(1-self.etha_vel)*self.vel_x


    if self.vel_joy >=0 and vel_d < 0:
        vel_d = 0

    u0=self.etha_delta*math.tan(self.steer_joy)+ (1-self.etha_delta)*f_tot_y*self.wheelbase/(max(self.vel**2,self.etha_delta))
    delta_d=math.atan((u0-math.tan(self.steer_joy))/(1+u0*math.tan(self.steer_joy)))+self.steer_joy
```


```
vel_d= (self.etha_vel)*self.vel_joy+(1-self.etha_vel)*self.vel_x


if self.vel_joy >=0 and vel_d < 0:
    vel_d = 0

u0=self.etha_delta*math.tan(self.steer_joy)+ (1-self.etha_delta)*f_tot_y*self.wheelbase/(max(self.vel**2,self.etha_delta))
delta_d=math.atan((u0-math.tan(self.steer_joy))/(1+u0*math.tan(self.steer_joy)))+self.steer_joy
```


The script above is designed to deduce the optimal speed and steering trajectory for an autonomous vehicle, factoring in both the input from a joystick and an obstacle evasion algorithm. The script orchestrates the car's speed and navigational direction by interpreting LIDAR sensor feedback.

The initial segment deduces the target speed, "vel_d," by creating a balance between the speed indicated by the joystick, "vel_joy," and the car's present velocity, "vel_x." The influence of the joystick's speed is modulated by the coefficient "etha_vel." An if-statement then verifies whether the joystick's speed input is in the positive domain while the calculated target speed, "vel_d," falls into the negative. If so, it sets "vel_d" to zero to prevent the automobile from engaging in reverse motion.

Subsequently, the algorithm determines the requisite steering angle, "delta_d," which integrates input from the joystick and the collective adjusting force, "f_tot_y," emitted from the obstacle avoidance system. Here, "u0" denotes the anticipated curvature of the path, influenced by both the joystick input and the modifying force. The influence of the corrective force is scaled by "etha_delta." To avoid division by zero, the "max()" function guarantees a non-zero denominator.

Finally, the joystick's input is amalgamated with the computed steering angle, aiming to steer the vehicle along the chosen trajectory while still responding to any joystick commands.

Activity 7: Before you can test the code, you need to replace the following files in your existing software stack with the ones provided on Avenue to Learn for Lab 8:

```
# name of file to write collision log to
collision_file: "collision_file"

# The names of the transformation frames published to
map_frame: "map"
base_frame: "base_link"
scan_frame: "laser"
odom_frame: "odom_frame"
imu_frame: "imu_frame"

broadcast_transform: true
publish_ground_truth_pose: true


# Ackermann to VESC parameters

# (change these values based on your own calibration)

speed_to_erpm_gain: 3182.18
speed_to_erpm_offset: 0.0
steering_angle_to_servo_gain: -0.88
steering_angle_to_servo_offset: 0.4450000

driver_smoother_rate: 75.0 # messages/sec


# Collision Assistance Algorithm Parameters

distance_to_obstacle_th:  1.0
force_gain: 8.0
velocity_correction_gain: 0.8
steering_correction_gain: 0.6

emergency_brake_active: true

# Wall Following Algorithm Parameters

CenterOffset: 0
DistanceLeft: 1.0
DistanceRight: 1.0
TrackWall: 0
angle_bl: rad(270*pi/180)
angle_al: rad(220*pi/180)
angle_br: rad(90*pi/180)
angle_ar: rad(140*pi/180)
k_d: 4
k_p: 6

# Speed control paramteres
vehicle_velocity: 2.0
heading_beam_angle: rad(pi/16)
stop_distance: 0.6 #m
stop_distance_decay: 1.0 #m
```

We replace the files behavior_conrtoller.cpp, mux.cpp, simulator.launch, experiment.launch, params.yaml with the given files.We keep the parameters result of calibration params.yaml file from lab 6.

Activity 8:

*simulator.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Listen to messages from joysicks -->
  <node pkg="joy" name="joy_node" type="joy_node"/>

  <!-- Launch a map from the maps folder-->
  <arg name="map" default="$(find f1tenth_simulator)/maps/env_map.yaml"/>
  <node pkg="map_server" name="map_server" type="map_server" args="$(arg map)"/>

  <!-- Launch the racecar model -->
  <include file="$(find f1tenth_simulator)/launch/racecar_model.launch"/>

  <!-- Begin the simulator with the parameters from params.yaml -->
  <node pkg="f1tenth_simulator" name="f1tenth_simulator" type="simulator" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the mux node with the parameters from params.yaml -->
  <node pkg="f1tenth_simulator" name="mux_controller" type="mux" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the behavior controller node with the parameters from params.yaml -->
  <node pkg="f1tenth_simulator" name="behavior_controller" type="behavior_controller" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the Random Walker Node -->
  <node pkg="f1tenth_simulator" name="random_walker" type="random_walk" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the Keyboard Node -->
  <node pkg="f1tenth_simulator" name="keyboard" type="keyboard" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

 <node pkg="f1tenth_simulator" name="collision_assistance" type="collision_assistance.py" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch RVIZ -->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find f1tenth_simulator)/launch/simulator.rviz" output="screen"/>
</launch>
```

```
random_walk_button_idx: 1 # ? button
nav_button_idx: 5 # RB button
collision_assistance_button_idx: 2 # X button
# **Add button for new planning method here**
new_button_idx: -1
```

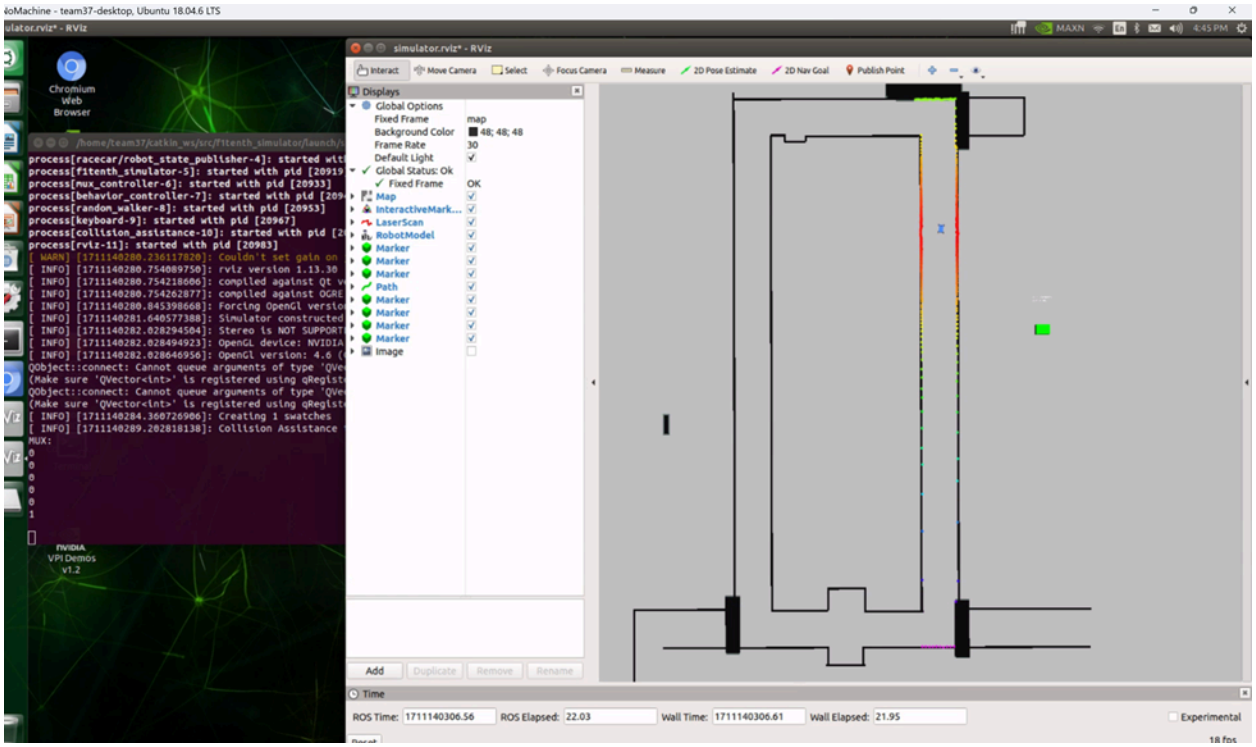
```
# Collision Assistance Algorithm Parameters

distance_to_obstacle_th:  1.0
force_gain: 8.0
velocity_correction_gain: 0.8
steering_correction_gain: 0.6
```

Activity 9: Examine the code in “collision_assistance.py” and relate the remaining parameters to those used in the collision avoidance assistance algorithm described above. You can tune these parameters

to achieve a desired response from the vehicle in the simulation environment. Launch the simulator by running:

```
$ roslaunch f1tenth_simulator experiment.launch
```

Drive the vehicle in the virtual environment with and without collision avoidance assistance and notice any difference in its behavior. Use the simulation to explore the impact of the above parameters on the vehicle response. Include your observations in your report.



Driving without collision avoidance assistance, when the car hits the wall, the terminal will return collision information and the joystick control needs to be restarted. When the "X" button is pressed, the terminal sends a message indicating that the avoidance assistance is activated. When we try to drive the vehicle toward a wall, the vehicle tries to turn in the other direction and stops waiting for the next speed or steering angle command.

In the virtual map, the vehicle turns at a large angle and needs to turn quickly, so we need to modify the force_gain, the steering_correction_gain and the velocity_correction_gain. By lowering these three parameters, the vehicle can avoid walls more smoothly. Also, at the corners of the map, the vehicle will recognize the corner as an obstacle and try to avoid it. Decreasing distance_to_obstacle_th corrects this problem and allows the vehicle to make sharp turns without encountering any resistance.

```
collision_assistance_button_idx: 2 # X button

distance_to_obstacle_th:  1.0
force_gain: 8.0
velocity_correction_gain: 0.8
steering_correction_gain: 0.6
```

collision_assistance_button_idx: Setting the X button on the joystick to control the turning on and off of the collision avoidance assistance algorithm.

distance_to_obstable_th: This parameter is used to determine the distance between the vehicle and obstacles. When the distance is too small, the algorithm will intervene to slow down the vehicle's movement.

force_gain: This parameter is designed to control the ratio between joystick control and the collision avoidance assist algorithm. When this parameter is close to 0, the vehicle's driving will completely depend on the steering angle set by the algorithm. When this parameter is close to 10, the joystick will control the steering angle.

steering_correction_gain: The proportion of joystick control in vehicle steering increases as the parameters increase. When set to 0, the algorithm will determine the steering angle entirely. When this parameter is set to 1, the joystick controls the steering position.

velocity_correction_gain: The proportion of joystick control in vehicle speed increases as the parameter increases. When set to 0, the algorithm will determine the vehicle velocity. When this parameter is set to 1, the joystick controls the vehicle's velocity.

Activity 10: Before using the LiDAR for collision avoidance, you must ensure you are using the right value for the parameter "scan_beams" in the file "params.yaml". This value denotes the number of scan beams in one full LiDAR rotation. The LiDAR angular resolution is therefore, given by $2\pi/scan\_beams$. Copy the following lines from "experiment.launch" into a new launch file so you can run the RPLidar by itself in the BOOST (highest resolution) mode.

```
Open ▾                                                    activity_10
                                                          ~/catkin_ws/src/f1tent
<?xml version="1.0"?>
<launch>

  <!--  launch rplidar driver node-->
  <node pkg="rplidar_ros" name="rplidarNode" type="rplidarNode">
  <param name="port" value="/dev/sensors/rplidar" />
  <param name="frame_id" type="string" value="laser"/>
  <param name="inverted"  type="bool"   value="false"/>
  <param name="angle_compensate"  type="bool"   value="true"/>
  <param name="scan_mode"   type="string" value="Boost"/>
  </node>



  </launch>
```

```
team37@team37-desktop:~/catkin_ws$ roslaunch f1tenth_simulator activity_10.launch
... logging to /home/team37/.ros/log/3df45d54-e88e-11ee-ad85-28d0eaa31272/roslaunch-team37-desktop-22686.
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://team37-desktop:38097/

SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.13
 * /rplidarNode/angle_compensate: True
 * /rplidarNode/frame_id: laser
 * /rplidarNode/inverted: False
 * /rplidarNode/port: /dev/sensors/rplidar
```

LiDar angular resolution

```
team37@team37-desktop: ~
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
0.00871450919658
---
```

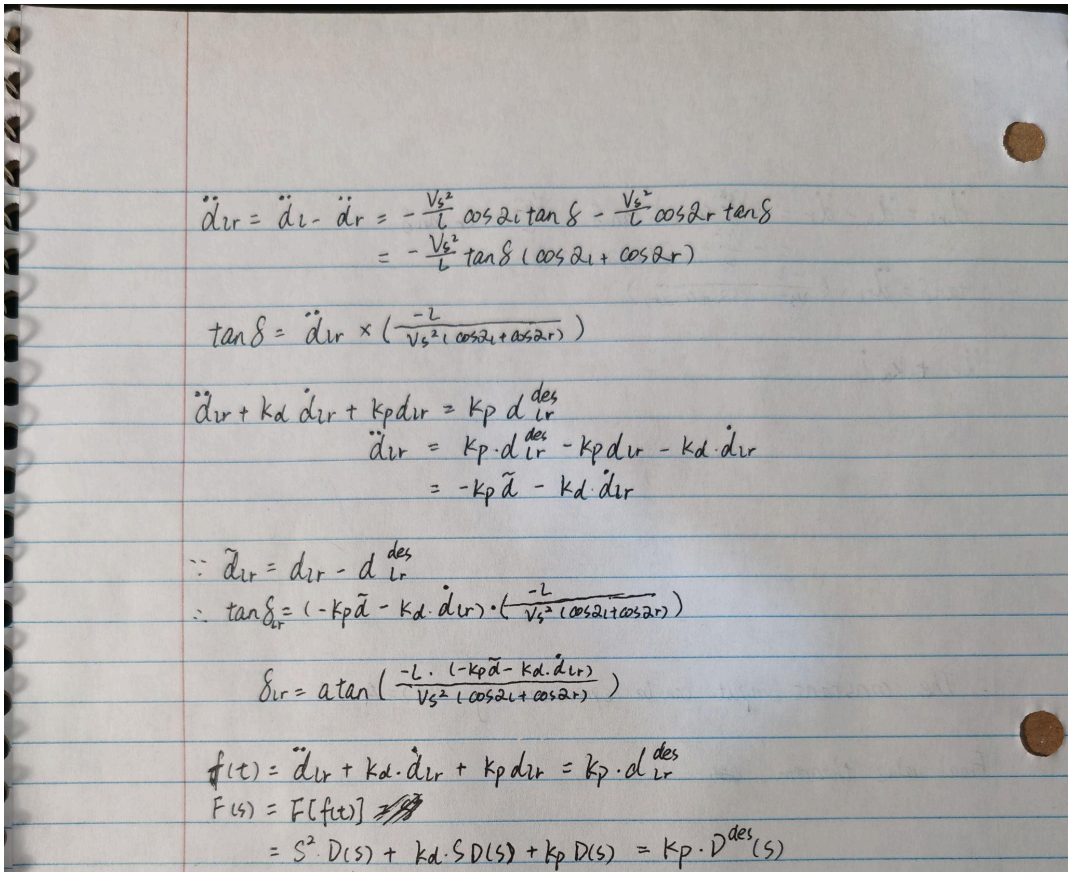
```
# Lidar simulation parameters
scan_beams: 720
scan_field_of_view: 6.2831853 #4.71 # radians
scan_range: 12.0
```

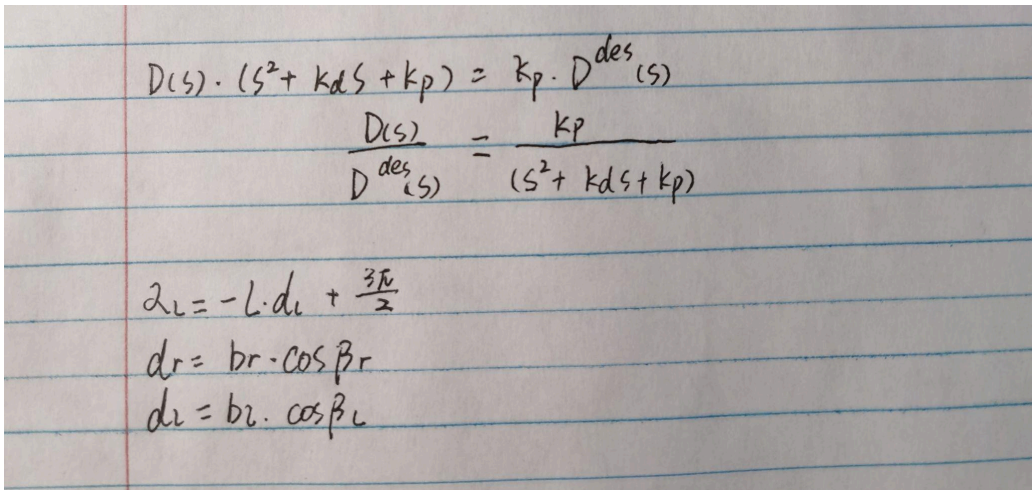
scan_beams: 2*π / LiDar angular resolution = 721.003, which is close to the parameter set in the params.yaml file.

Activity 11: Examine the vehicle behavior in experiment. Open “experiment.launch” file and make sure that the collision assistance avoidance node is launched. Run the launch file by:

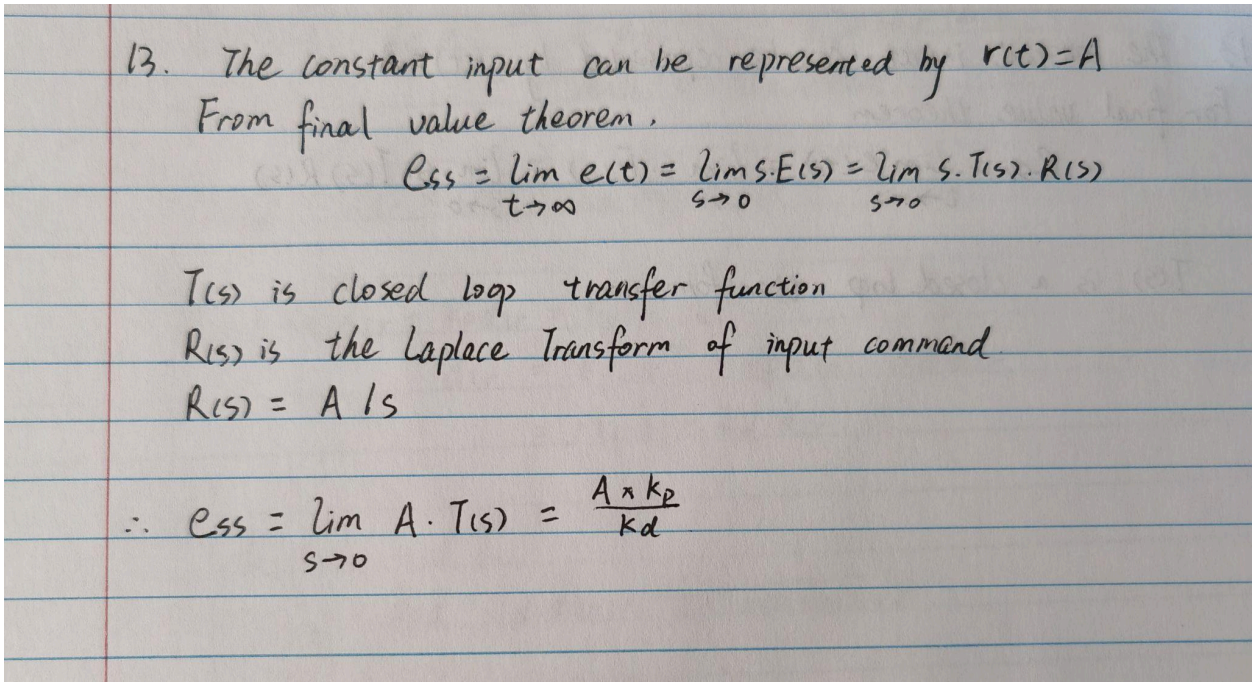
As we drive the vehicle through the hallway and fine-tuning the parameters, we observe the impact of these changes on the vehicle's response. For instance, if we increase the value of Velocity_Correction_gain, we will find that we can slow down the vehicle better with joystick control when the vehicle is about to hit a wall. After modifying the parameters in the simulation, the vehicle travels along a straight wall. When we tried tilting the front of the car, it returned to alignment with the hallway. Additionally, the car can detect people as obstacles and move to avoid them when people appear in its path.

Activity 12: Some of the mathematical derivations in the rest of this document are intentionally left incomplete. You must complete and include these derivations in your report.





Activity 13: Determine the steady-state tracking error in response to a constant distance command in the closed-loop control.



Activity 14 & 15: Complete the missing parts of the codes denoted by "…" to implement the algorithm in the previous section.

```
self.thetal= self.angle_bl-self.angle_al
self.thetar= self.angle_ar-self.angle_br
```

```
betal= math.atan((-dis_lsr_bl+dis_lsr_al*math.cos(self.thetal))/(dis_lsr_al*math.sin(self.thetal)))
betar= math.atan((-dis_lsr_br+dis_lsr_ar*math.cos(self.thetar))/(dis_lsr_ar*math.sin(self.thetar)))

alphal = -betal - self.angle_bl+3*math.pi/2
alphar = betar - self.angle_br+math.pi/2
```

```
    else :
        d_tilde= dl-dr - self.CenterOffset
        d_tilde_dot= -self.vel*math.sin(alphal)-self.vel*math.sin(alphar)
        delta_d = math.atan((self.wheelbase*(self.k_p*d_tilde+self.k_d*d_tilde_dot))/((self.vel**2)*(math.cos(alphal)+math.cos(alphar))))

else:
    delta_d = 0
```

Activity 16: Explain what the following section of the code accomplishes.

```
min_distance = min(data.ranges[-sec_len+int(self.scan_beams/2):sec_len+int(self.scan_beams/2)])
velocity_scale = 1-math.exp(-max(min_distance-self.stop_distance,0)/self.stop_distance_decay)


velocity=velocity_scale*self.vehicle_velocity
```

First line: min_distance = min(data.ranges[-sec_len+int(self.scan_beams/2):sec_len+int(self.scan_beams/2)])

"min_distance" finds the minimum distance from the lidar scan range array to an obstacle. The array is centered in the vehicle forward direction and extends the "sec_len" beams to each side, determined by "heading_beam_angle".

Second line: velocity_scale = 1 - math.exp(-max(min_distance - self.stop_distance, 0) / self.stop_distance_decay)

This line calculates the scale factor of the vehicle's speed. It uses an exponential decay function based on "min_distance" and "stop_distance". If "min_distance" is less than "stop_distance", the scale will be close to 0 and the vehicle will slow down to avoid a collision. The parameter "self.stop_distance_decay" adjusts how fast the velocity decreases as "min_distance" approaches "self.stop_distance".

Third line: velocity = velocity_scale * self.vehicle_velocity

Calculating the vehicle's speed by multiplying the initial velocity "self.vehicle_velocity" by the "velocity_scale" calculated in the previous step.

Overall, this code adjusts the vehicle's speed to slow down and prevent collisions when approaching obstacles.

Activity 17: The values given above represent good starting points for various control parameters, but you are encouraged to adjust them to fine tune the response of the vehicle. Note that you have the option of controlling the vehicle in one of three modes, distance-to-left wall, distance-to-right wall, or an offset from the center position of the walls. You can choose the driving mode by setting the value of the parameter “TrackWall” in the file “params.yaml”. Observe the vehicle behavior in the simulation environment in each of these modes and comment on what you see in your response. Comment on the

trade-offs in using each of the three control modes to self-drive the vehicle in this environment. Where do you see most challenges and why?

center position of the walls

distance-to-left wall mode

distance-to-right wall mode

By adjusting the value of the TrackWall parameter to 0, 1, and 2, the vehicle can switch between different modes, including center position of the walls, distance-to-left wall mode, and distance-to-right wall mode. The DistanceLeft and DistanceRight parameters control the distance between the vehicle and the wall in distance-to-left wall and distance-to-right wall modes, while the vehicle_velocity parameter can be used to change the speed of the vehicle. motor vehicles. If we set the TrackWall parameter to the distance-to-left wall mode, the vehicle will drive along the left wall. When there is a slip road on the left, the vehicle will try to follow the boundary or turn onto the slip road. There will be a similar observation in the distance-to-right wall mode. When the vehicle travels along the center position of the walls and passes through the gaps in the floor tiles, it will tilt its head slightly and then quickly return to its original position.

When we default the vehicle to distance-to-left mode, it can easily turn right due to its large turning radius. Also, the vehicle can turn left easily in the distance-to-right mode. However, when the vehicle is at the center position of the wall, the lack of clear boundaries may confuse the lidar, making it difficult to pass through intersections and leading to self-controlling instability.

The vehicle control method uses mathematical model algorithms to predict the behavior of the vehicle in different situations. When the vehicle is in a predictable environment, such as driving close to two walls, it is able to adapt to new situations and handle complex environments well. However, it creates challenges when it faces environments beyond the scope of its model, such as at intersections where vehicles may not be able to navigate smoothly.

Explain the impact of the controller gains kp and kd on the response characteristics. How do you choose these values? You can refer to the course lecture slides for discussion on second-order systems response characteristics.

In the control system, the controller gains Kp and Kd are important parameters that affect the response characteristics. Kp represents proportional gain, and Kd represents derivative gain. These gains are used to adjust the system's response to input signals, such as changes in setpoint or interference.

The proportional gain Kp affects the steady-state response of the system. For a given input, a higher Kp value results in a larger output response, approaching the desired set point faster. However, increasing Kp too much may cause overshoot and oscillations in the system setpoint. A lower Kp value leads to a slower approach to the setpoint and a more stable response.

The derivative gain Kd affects the transient response of the system. Higher Kd values increase the system's response to changes in the input signal, which helps reduce overshoot and oscillation. A lower Kd gain will let the system's response be damped, resulting in more overshoot and oscillations.

The values of Kp and Kd should be chosen based on the requirements of the system and response characteristics, such as frequency and damping ratio. These can be determined by using techniques such as frequency response analysis or step response analysis, which allows adjusting Kp and Kd values. Overall, choosing appropriate Kp and Kd values requires a trade-off between speed, stability, and oscillation damping.

Activity 18: Make sure the environment around the vehicle is free of people and safe for operating the vehicle. Execute the following command to start the vehicle control system:

```
$ roslaunch f1tenth_simulator experiment.launch
```

We set the parameter TrackWall to 0, the vehicle travels along the center position of the walls. The vehicle makes a slightly zig-zag motion when it passes through gaps in the floor tiles or intersections. Thus, the algorithm and controller parameters need to be tuned and improved further. This would also be improved in lab 9 further with a different algorithm.

Activity 19: Do you observe any difference in the vehicle behavior between the simulations and the experiment? Comment on your observations in this regard.

Like in the experiment, the behavior of the vehicle in the simulation is also zig-zaggy, albeit a bit smoother. While the vehicle's motion is not perfectly straight in either case, deviations from a straight path are less severe in the simulation due to the environment being more controlled and predictable. This comparison is important for understanding how cars perform in experiment versus simulation and highlights the challenge of replicating the unpredictability of the real world in simulation.