TECHNISCHE UNIVERSITÄT WIEN

Autonomous Racing Cars
191.119 (VU 4,0) Semester: 2022S

Reactive Methods

2022-05-03

Student data

Matrikelnummer	Firstname	Lastname
12134689	Danilo	Castiglia
12135191	Artur	Chaves
01613004	Severin	Jäger
01552500	Johannes	Windischbauer

Transparency of contribution

- **Danilo Castiglia**: Disparity Extender algorithm implementation, testing on the simulator and on the hw car and parameters tuning
- **Artur Chaves**:
- **Severin Jäger**: Algorithm improvements and tuning, optimisation on hardware
- **Johannes Windischbauer**: Visualization of disparity extender algorithm, plots

Note

This page will not be shared with other teams. All following pages and the submitted source code archive will be shared in TUWEL after the submission deadline. Do not include personal data on subsequent pages.

TECHNISCHE UNIVERSITÄT WIEN

Autonomous Racing Cars
191.119 (VU 4,0) Semester: 2022S

Reactive Methods

2022-05-03

Team data

Team number: 2

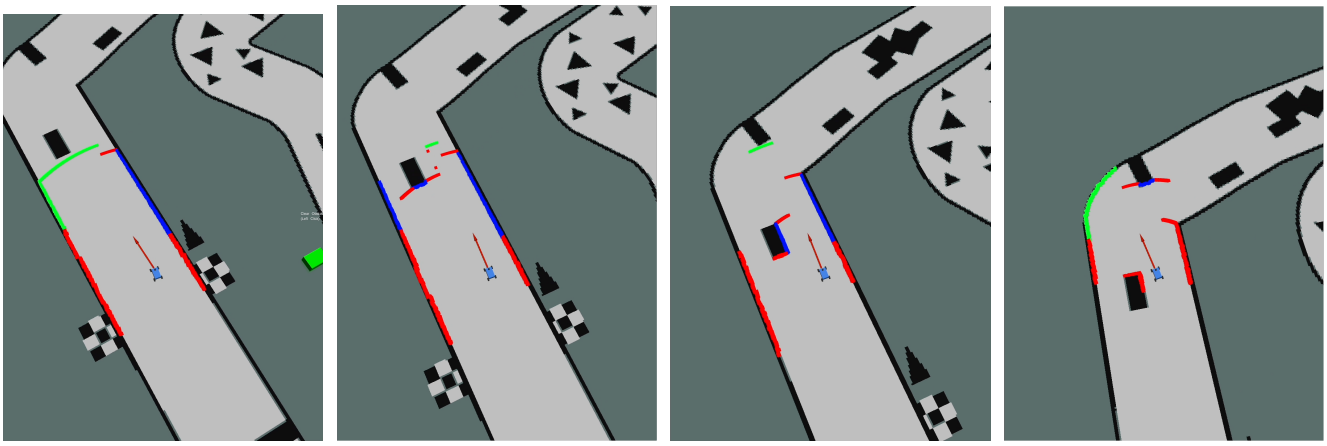
1 Reactive Algorithms

Due to its success in earlier races we decided to implement the disparity extender algorithm. We closely followed the original blog post[1] and omitted the back of the car tweak.

1.1 Implementation

a.) The disparity extender algorithm was implemented as a python node with several parameters to change. To maximize our performance we decided that we are also defining a flag for the visualization, so that we are able to reduce the computational complexity.

b.) Visualization of disparity extender algorithm by setting custom intensities for the different beams to highlight disparities and blocked areas (red), possible directions (blue) and the chosen gap (green). The direction is visualized with a PoseStamped and displays the steering angle. Driving and rapid changes in distances can cause a jittery steering.



[1] https://www.nathanotterness.com/2019/04/the-disparity-extender-algorithm-and.html

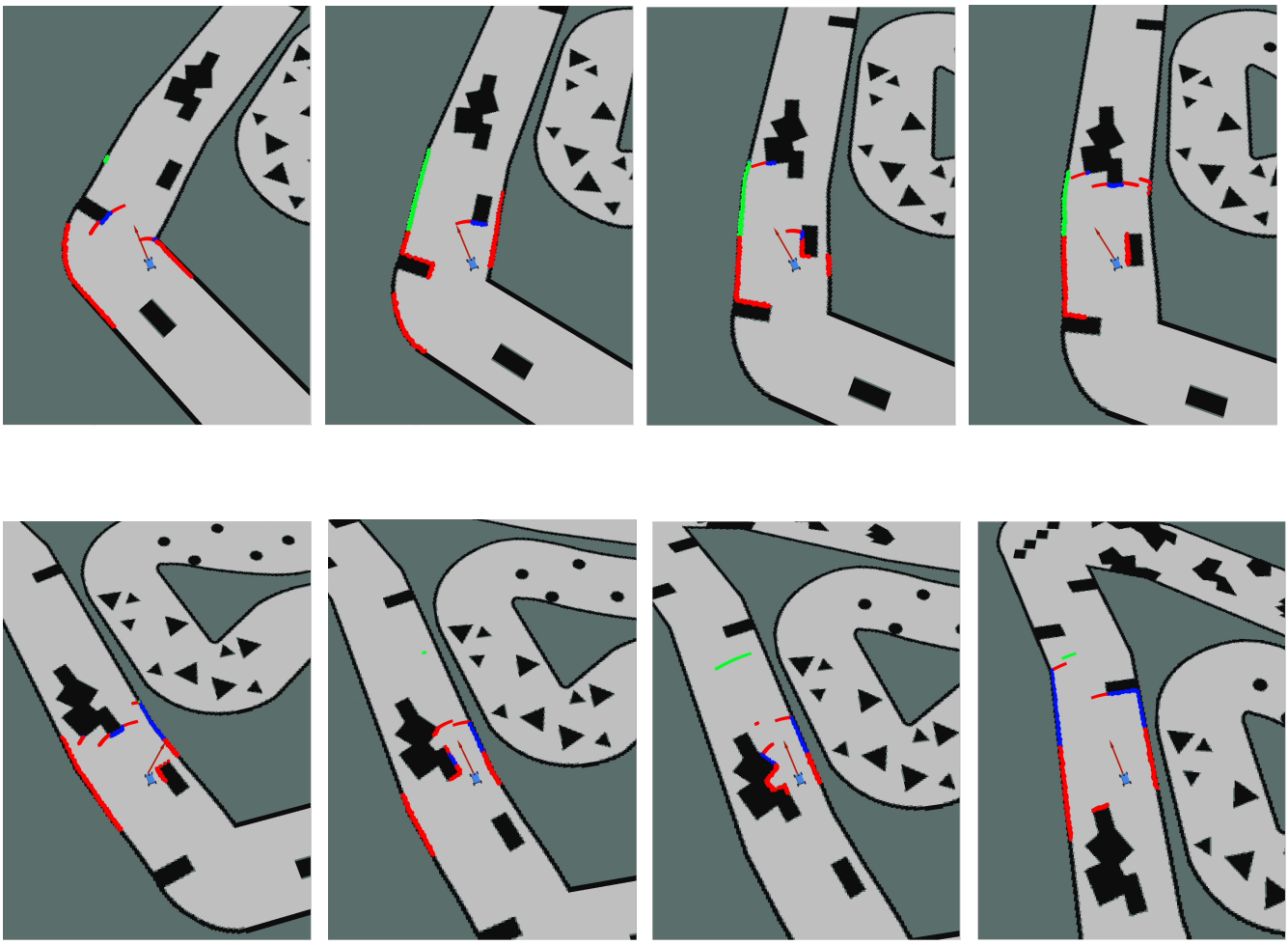
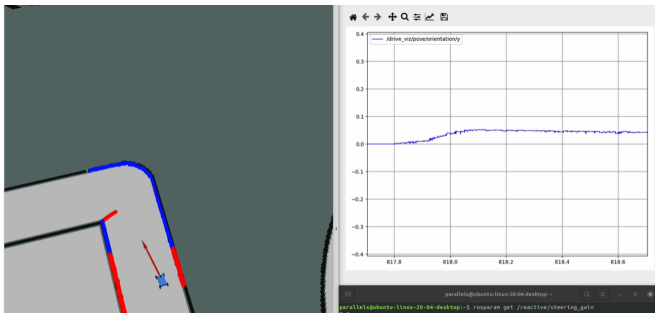


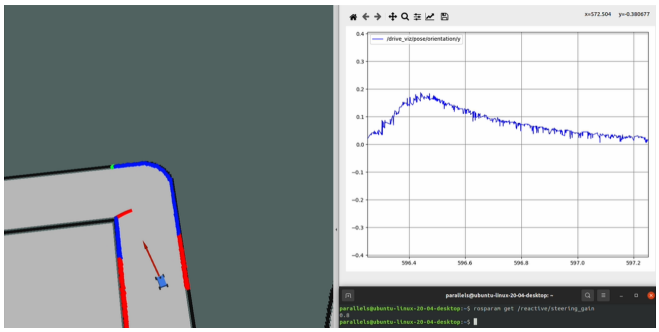
Figure 3: From left to right: step by step visualization of movement of the car and the disparity extender algorithm. The chosen gap is highlighted in green.

c.) For this part we decided to visualize the three parameters that had the biggest impact visually on the driving behavior of the car. To back up our claims pictures of the car in the simulator and data from the rqt plot are shown side-by-side.

- **Steering gain:** The steering gain controls the behavior of steering when turning, giving more or less steering input. Smaller values of steering angle will give slower and smoother steering response witch is good for higher speeds,because the car doesn't oscillate but when finding an obstacle takes more time to turn away from it. When using higher values of steering gain the car oscillates more at higher speeds but is better when avoiding an obstacle. It was used values of 0.2 and 0.8 as seen in the images for the steering angle, and it can be seen this differences when turning. When using the 0.8 the car is more aggressive when turning compared to the 0.2.

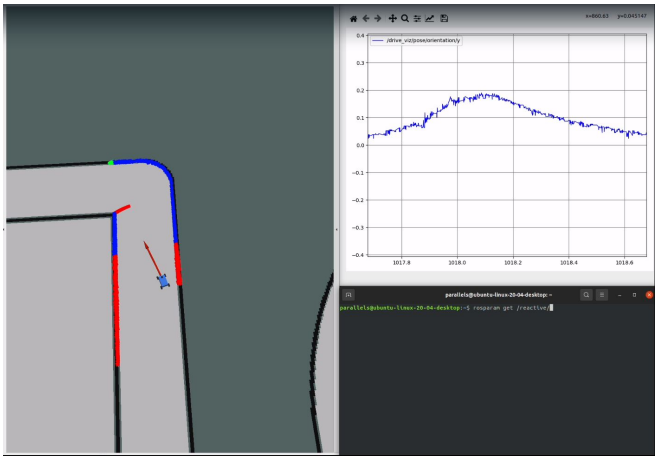


(a) Steering gain set to 0.2 on map rectangle

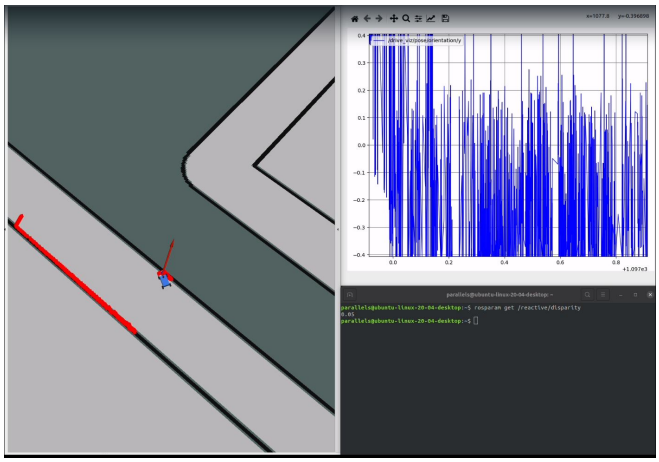


(b) Steering gain set to 0.8 on map rectangle

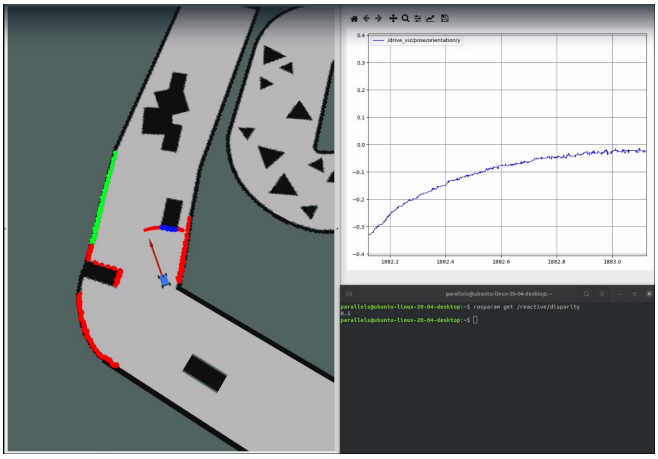
- **Disparity:** The disparity is the minimum difference between the distances measured by two near lidar lasers. With a high disparity no obstacle will be found, with a low disparity everything will be seen as an obstacle. We tried 2 settings: disparity = 0.5 and disparity = 0.05. The first is acceptable, instead the second crashes after a few seconds on both the tracks. This is because a lot of obstacles are recognized and the car tries to go in all the directions. All the results can be seen in the Figures below.



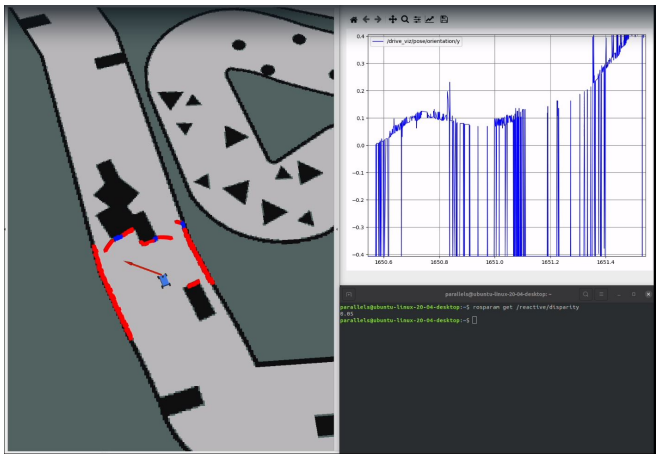
(a) Disparity set to 0.5m on map rectangle



(b) Disparity set to 0.05m on map rectangle



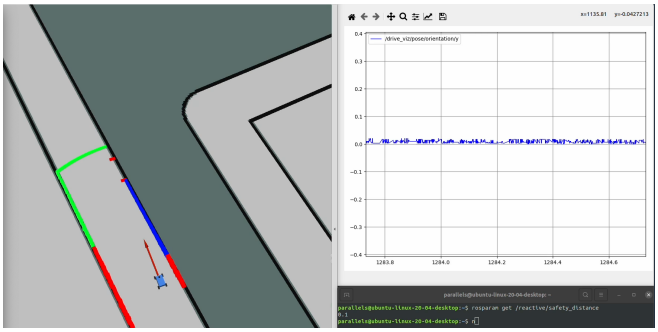
(a) Disparity set to 0.5m on map f1_aut_wide_obstacles



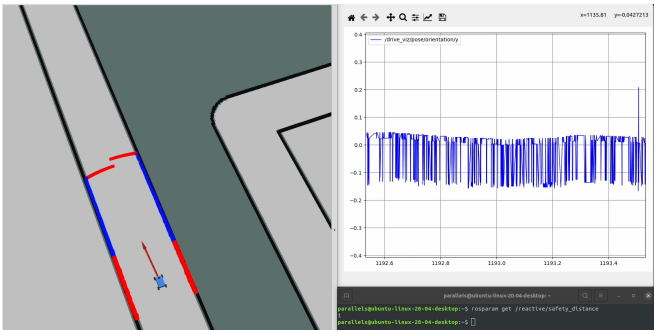
(b) Disparity set to 0.05m on map f1_aut_wide_obstacles

- **Safety distance:** The safety distance defines how far the objects seemingly extend past their real end and define where our car can possibly pass. Larger values (like 1m) result in a very strong step response

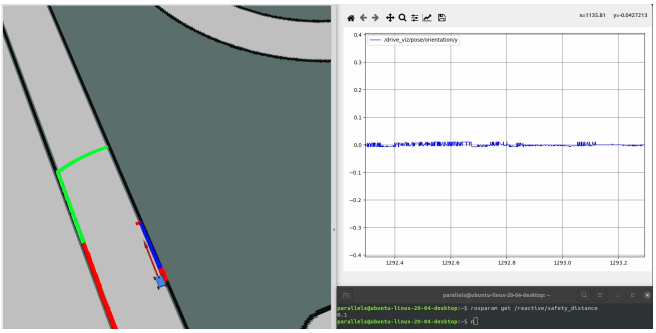
on the rectangle map because it's simply to small and a little turn causes the disparity change from the left to the right wall and therefore the safety distance to jump to the other side causing the car to turn strongly, which in turn results in heavy oscillation. This effect is still visible but much less pronounced at the beginning of the wider f1_aut_wide_obstacles map and gets even more intense once the obstacles get bigger and the possible path smaller as can be seen in figure ??.



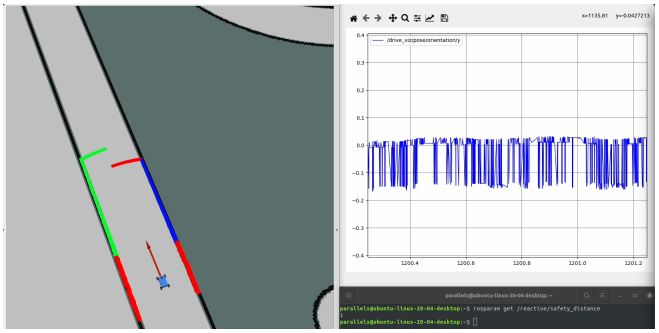
(a) Safety distance set to 0.1m on map rectangle



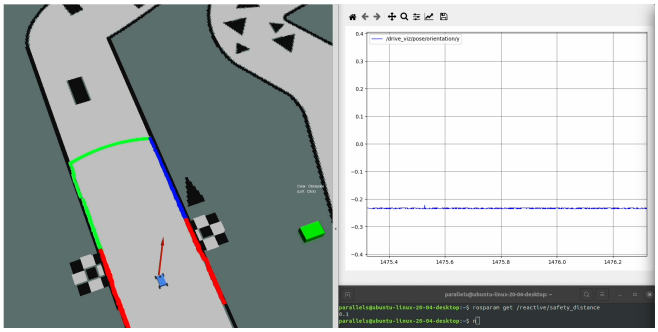
(b) Safety distance set to 1m on map rectangle



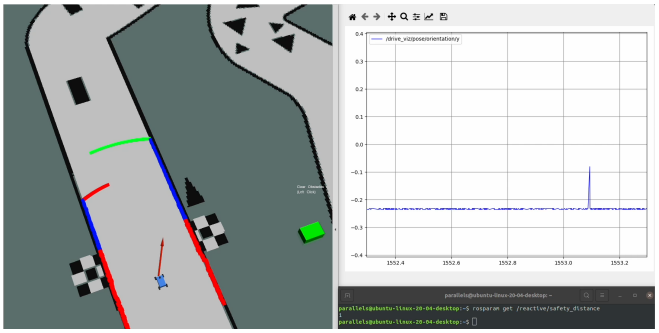
(a) Safety distance set to 0.1m on map rectangle



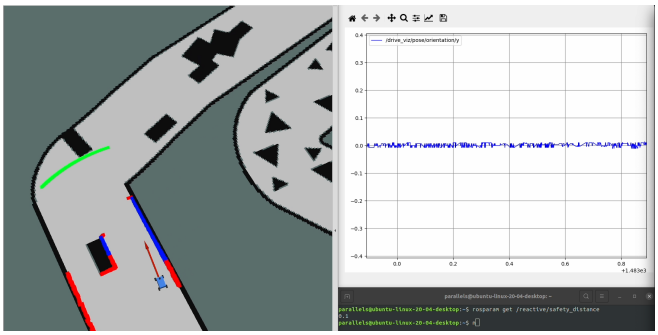
(b) Safety distance set to 1m on map rectangle



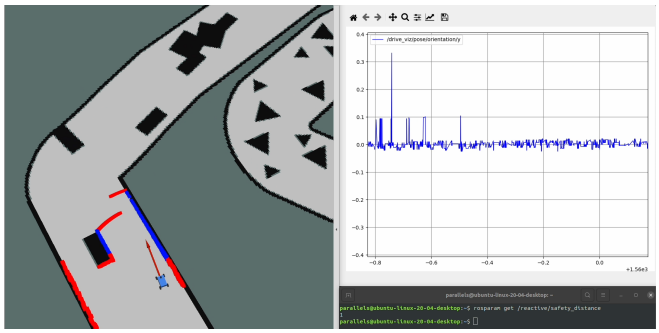
(a) Safety distance set to 0.1m on map f1_aut_wide_obstacles



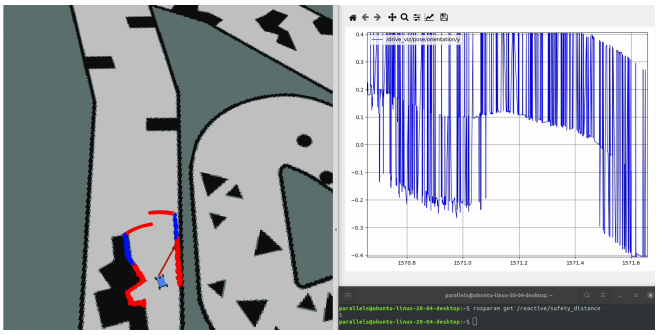
(b) Safety distance set to 1m on map f1_aut_wide_obstacles



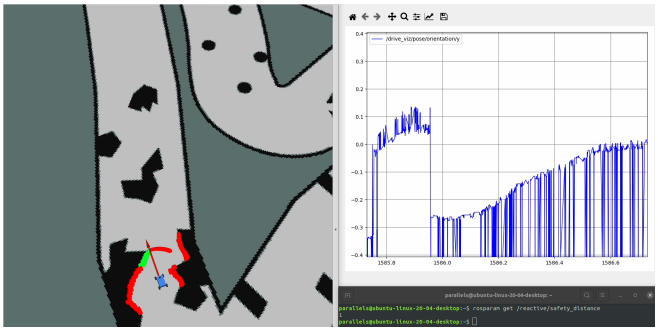
(a) Safety distance set to 0.1m on map f1_aut_wide_obstacles



(b) Safety distance set to 1m on map f1_aut_wide_obstacles



(a) Safety distance set to 1m on map f1_aut_wide_obstacles near obstacles



(b) Safety distance set to 1m on map f1_aut_wide_obstacles near obstacles

Figure 11: Very narrow track as a result of obstacles

d.) (Done in person on Monday the 16th of May)

e.) Our algorithm performed pretty well on the hardware car. The tutors made a small race between all the teams and we won. The curves that the car made were really amazing. So there is still margin to improve the speed.

f.) We had some troubles running the algorithm on the hardware car because we used python3 on the simulator and python2 on the car. We had to change some python math function to make it works. Except this, the algorithm was already working. We made some tuning to achieve better performances and add a new parameter: the lidar maximum angle, so it was not fixed to 180 degrees anymore.

1.2 Tuning the Controller

a.) We managed to complete the f1_aut_wide_obstacles map with the node given in Listing 1 and the launch file in Listing 3. When tuning the algorithm for this map, we found several small yet critical bugs in our implementation. Mainly we were not handling situation with many edges present at the same time sufficiently well. This did however neither become apparent in the simulation with other maps or on the hardware.

To reduce our lap times, we introduced a variable velocity mapping. Firstly, we tried to simply make the velocity proportional to the steering angle (like in lab 3), however this lead oscillatory trajectories. Thus, we consider the distance of the current straight (as in the disparity extender algorithm) and calculate

$$v_{car} = \alpha_v d_{straight} \tag{1.1}$$

with our newly introduced velocity gain α_v. Furthermore, we clip the velocity with v_{min} and v_{max}. However, to achieve a safe trajectory we did not tune them very aggressively.

The most critical parameters for successful completion of the obstacle map were disparity and safety distance. In addition a reduced lidar angular range helped to prevent the car from turning around and improved the code execution speed (which is critical in the simulator due to the high rate of scan updates). Firstly, as shown in Figure 12 a too large safety distance basically blocks narrow track segments. However, a too low distance causes collisions. This required a fine trade-off.

The second challenge were crossroads as depicted in Figure 13. When the car can not decide for one side, it will likely crash the obstacle in the center. This again requires a good balance for safety distance and disparity.

These are our final parameters for this task:

Parameters	Optimisation
disparity	0.2 m
safety distance	0.42 cm
Lidar angle range	160°
Steering gain	0.8
Velocity gain α_v	0.6
v_{min}	1.2 m/s
v_{max}	3 m/s

Table 1: Tuned parameters

b.) (not covered)

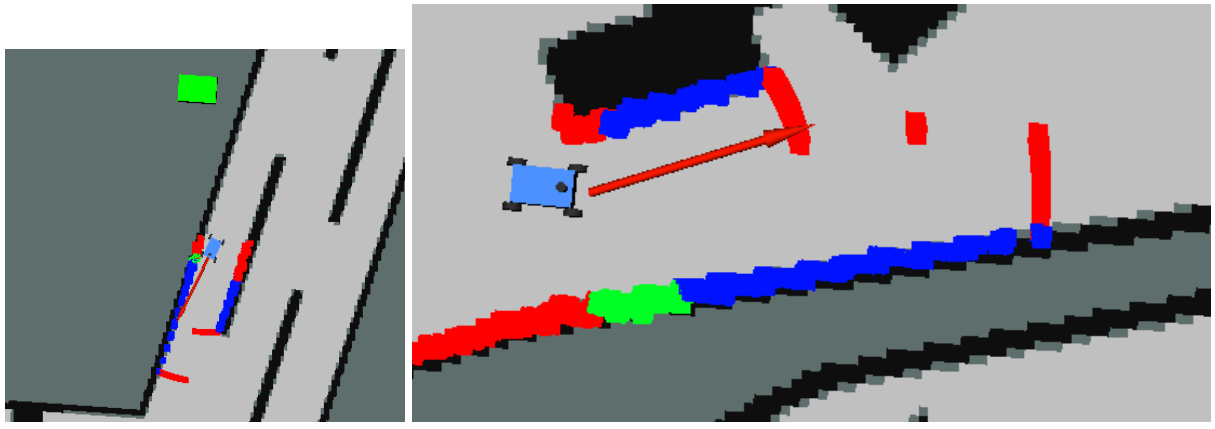


Figure 12: Narrow track parts on f1_aut_wide_obstacles

c.) During the on-site lab we optimised our algorithm to drive as fast as possible in the race. The launch file we used in the time runs is given in Listing 3. Eventually, we were able to achieve faster lap times than all other teams.

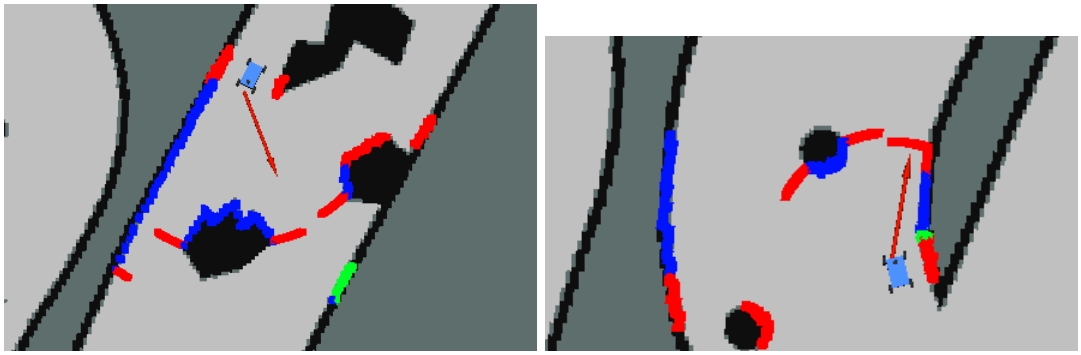


Figure 13: Narrow track parts on f1_aut_wide_obstacles

2 Appendix

Listing 1: Disparity Extender Node

```
#!/usr/bin/env python3
from __future__ import print_function
from curses import raw
from platform import java_ver
from re import T
import sys
import math

#ROS Imports
import rospy
# from std_msgs.msg import Float64
from sensor_msgs.msg import LaserScan
from ackermann_msgs.msg import AckermannDriveStamped
# from visualization_msgs.msg import MarkerArray, Marker
from geometry_msgs.msg import PoseStamped

# DISPARITY EXTENDER PARAMS
VISUALIZATION = rospy.get_param('/reactive/visualization', False)  # quite
    ↪ costly in performance, this leads to problems in the test bench
BASIC_VELOCITY = False  # simple velocity scheme from the assignment sheet,
    ↪ otherwise more aggressive behaviour
MAX_SPEED = rospy.get_param('/reactive/max_speed', 3)  # m/s  (only without
    ↪ basic velocity)
MIN_SPEED = rospy.get_param('/reactive/min_speed', 1.2)  # m/s  (only without
    ↪ basic velocity)
VELOCITY_GAIN = rospy.get_param('/reactive/velocity_gain', 0.6)
STEERING_GAIN = rospy.get_param('/reactive/steering_gain', 0.8)
# The minimum distance that is considered a disparity
DISPARITY = rospy.get_param('/reactive/disparity', 0.2)  # m
# Safety distance to maintain from a disparity
SAFETY_DISTANCE = rospy.get_param('/reactive/safety_distance', 0.42)  # m
LIDAR_ANGULAR_RANGE = rospy.get_param('/reactive/lidar_angular_range', 160)   #
    ↪  degrees
LIDAR_RANGE = rospy.get_param('/reactive/lidar_range', 20) # m


# When the distance in front is less than this, the car turns (so far unused)
MIN_DISTANCE_TO_TURN = 1000

# CONSTANTS
RAYS_PER_DEGREE = 4
MAX_STEERING_ANGLE = 24  # degrees

viz_arr = None
viz_ranges = None
lidar_data = None

class DisparityExtender:
    def __init__(self):
        rospy.loginfo("Hello␣from␣the␣reactive␣node")
```

```
        #Topics & Subs, Pubs
        lidarscan_topic = '/scan'
        drive_topic = '/nav'
        lidar_viz_topic = '/lidar_viz'
        drive_viz_topic = '/drive_viz'

        self.blocked = 0
        self.possible = 11
        self.chosen = 6

        self.lidar_sub = rospy.Subscriber(lidarscan_topic, LaserScan, self.
            ↪ lidar_callback)
        self.drive_pub = rospy.Publisher(drive_topic, AckermannDriveStamped,
            ↪ queue_size=10)
        self.lidar_viz_pub = rospy.Publisher(lidar_viz_topic, LaserScan,
            ↪ queue_size=10)
        self.lidar_drive_viz_pub = rospy.Publisher(drive_viz_topic, PoseStamped
            ↪ , queue_size=10)
        self.skipped = 0
        self.scan = None

    def lidar_callback(self, data):
        global viz_arr
        if VISUALIZATION:
            viz_arr = list(data.intensities)
            self.scan = data
            data.ranges = self.set_ranges()
        filtered_ranges = self.filter_ranges(data)
        processed_ranges = self.process_disparities(filtered_ranges)
        self.navigate_farthest(processed_ranges)

    # Filter the lidar data to obtain only lasers in range (-90, +90)
    def filter_ranges(self, data):
        global viz_arr, lidar_data
        lidar_data = data
        lasers = int(len(data.ranges) * LIDAR_ANGULAR_RANGE / 270)
        self.skipped = int((len(data.ranges) - lasers) / 2)
        filtered = data.ranges[self.skipped: self.skipped + lasers]
        if VISUALIZATION:
            viz_arr[0: self.skipped] = [0] * self.skipped
            viz_arr[self.skipped + lasers:] = [0] * self.skipped
        return filtered

    def process_disparities(self, ranges):
        global viz_arr
        processed_ranges = list(ranges)
        i = 0
        while i < (len(ranges) - 1):
            if abs(ranges[i] - ranges[i + 1]) >= DISPARITY:
                # Disparity
                if ranges[i] > ranges[i + 1]:
                    # Right edge
                    safety_rays = int(self.calculate_angle(ranges[i + 1])+0.5)
```

```
                for j in range(i - safety_rays, i+1):
                    if j < 0 or j >= len(processed_ranges):
                        continue
                    processed_ranges[j] = min(processed_ranges[i+1],
                        ↪ processed_ranges[j])

                    if VISUALIZATION:
                        viz_arr[j + self.skipped] = self.blocked
                        viz_ranges[j + self.skipped] = processed_ranges[j]
            else:
                # Left edge
                safety_rays = int(self.calculate_angle(ranges[i])+0.5)
                for j in range(i, i + safety_rays + 1):
                    if j < 0 or j >= len(processed_ranges):
                        continue
                    processed_ranges[j] = min(processed_ranges[i],
                        ↪ processed_ranges[j])

                    if VISUALIZATION:
                        viz_arr[j + self.skipped] = self.blocked
                        viz_ranges[j + self.skipped] = processed_ranges[j]

                # Skip the edited values
                # i += safety_rays - 1
        i += 1
    return processed_ranges

def calculate_angle(self, distance):
    # Calculated as 1080/270
    angle = math.degrees(math.atan(SAFETY_DISTANCE / distance))

    return angle * RAYS_PER_DEGREE

def navigate_farthest(self, ranges):
    ranges_list = list(ranges)
    straight = ranges_list[int(len(ranges_list)/2)]
    farthest = ranges_list.index(max(ranges_list))

    if straight < MIN_DISTANCE_TO_TURN:
        angle = self.get_angle(ranges, farthest)
    else:
        angle = 0.0

    # reduces oscillations when going at higher speeds
    angle = STEERING_GAIN * angle

    # Angle clipping. Probably handled by VESC anyway, but improves plot
       ↪ readablity
    if(angle > MAX_STEERING_ANGLE):
        angle = MAX_STEERING_ANGLE
    elif (angle < -MAX_STEERING_ANGLE):
        angle = -MAX_STEERING_ANGLE

    if BASIC_VELOCITY:
```

```
            velocity = 1
        else:
            velocity = straight * VELOCITY_GAIN
            if velocity > MAX_SPEED:
                velocity = MAX_SPEED
            if velocity < MIN_SPEED:
                velocity = MIN_SPEED

        drive_msg = AckermannDriveStamped()
        drive_msg.header.stamp = rospy.Time.now()
        drive_msg.header.frame_id = "laser"
        drive_msg.drive.steering_angle = math.radians(angle)
        drive_msg.drive.speed = velocity
        self.drive_pub.publish(drive_msg)
        if VISUALIZATION:
            self.set_intensities(farthest)
            self.publish_viz()
            drive_viz_msg = PoseStamped()
            drive_viz_msg.header.stamp = rospy.Time.now()
            drive_viz_msg.header.frame_id = "laser"
            drive_viz_msg.pose.orientation.y = velocity * math.sin(math.radians
               ↪ (angle))
            drive_viz_msg.pose.orientation.x = velocity * math.cos(math.radians
               ↪ (angle))
            self.lidar_drive_viz_pub.publish(drive_viz_msg)

    def get_angle(self, ranges, i):
        return (1.0 * i / len(ranges)) * LIDAR_ANGULAR_RANGE - (
           ↪ LIDAR_ANGULAR_RANGE / 2)
        # Funnily enough it works with the code below
        # return (1.0 * i / len(ranges)) * 180 - 90

    def set_intensities(self, farthest):
        global viz_arr, lidar_data
        i = farthest + self.skipped
        while i < len(viz_arr):
            if viz_arr[i] == 0:
                break
            viz_arr[i] = self.chosen
            i += 1
        i = farthest
        while i > 0:
            if viz_arr[i] == 0:
                break
            viz_arr[i] = self.chosen
            i -= 1
        i = 0
        while i < len(viz_arr):
            if not (viz_arr[i] == 0) and not (viz_arr[i] == self.chosen):
                viz_arr[i] = self.possible
            i += 1
        pub = lidar_data
        pub.intensities = viz_arr
        rospy.logdebug(viz_arr)
```

```
        self.lidar_viz_pub.publish(pub)

    def set_ranges(self):
        global viz_ranges
        dist = LIDAR_RANGE
        viz_ranges = list(self.scan.ranges)
        i = 0
        while i < len(self.scan.ranges):
            if viz_ranges[i] > dist:
                viz_ranges[i] = dist
            i += 1
        return viz_ranges

    def publish_viz(self):
        pub = self.scan
        pub.intensities = viz_arr
        pub.ranges = viz_ranges
        self.lidar_viz_pub.publish(pub)


def main(args):
    rospy.init_node("reactive_node", anonymous=True)
    dex = DisparityExtender()

    rospy.sleep(0.1)
    rospy.spin()

if __name__=='__main__':
    main(sys.argv)
```

Listing 2: Simulator launch file

```
<?xml version="1.0"?>

<launch>
    <remap from="/odom" to="/vesc/odom"/>
    <remap from="/brake" to="/vesc/low_level/ackermann_cmd_mux/input/safety"/>
    <remap from="/nav" to="/vesc/high_level/ackermann_cmd_mux/input/nav_0"/>
    <remap from="/joy" to="/vesc/joy"/>
    <include file="$(find␣racecar)/launch/teleop.launch"/>

    <arg name="disparity"  default="0.15"/>
    <arg name="safety_distance"  default="0.52"/>
    <arg name="max_speed"  default="6"/>
    <arg name="min_speed"  default="1.2"/>
    <arg name="velocity_gain"  default="1.5"/>
    <arg name="steering_gain"  default="0.3"/>
    <arg name="lidar_range"  default="120"/>
    <node name="reactive" pkg="reactive" type="reactive.py" output="screen">
        <param name="max_speed"  value="$(arg␣max_speed)"/>
        <param name="min_speed"  value="$(arg␣min_speed)"/>
        <param name="velocity_gain"  value="$(arg␣velocity_gain)"/>
        <param name="steering_gain"  value="$(arg␣steering_gain)"/>
        <param name="disparity"  value="$(arg␣disparity)"/>
        <param name="safety_distance"  value="$(arg␣safety_distance)"/>
```

```
        <param name="lidar_range"  value="$(arg␣lidar_range)"/>
    </node>
    <!--node name="safety_node" pkg="safety_node1" type="safety_node" output="
       ↪ screen" /-->
</launch>
```

Listing 3: Car launch file

```
<?xml version="1.0"?>

<launch>
    <!-- rect / rect_reverse / f1_aut_wide / f1_aut_wide_obstacles / f1_aut /
       ↪ f1_mco / f1_gbr / f1_esp / test -->
    <arg name="map"  default="f1_aut_wide_obstacles"/>

    <include file="$(find␣f1tenth_simulator)/launch/simulator.launch">
        <arg name="map" value="$(find␣f1tenth_simulator)/maps/$(arg␣map).yaml"/
           ↪ >
        <arg name="map_frame"  value="map"/>
    </include>

    <arg name="visualization" default="true"/>
    <arg name="disparity"  default="0.2"/>
    <arg name="safety_distance"  default="0.42"/>
    <arg name="max_speed"  default="3.5"/>
    <arg name="min_speed"  default="1.2"/>
    <arg name="velocity_gain"  default="0.6"/>
    <arg name="steering_gain"  default="0.8"/>
    <arg name="lidar_angular_range"  default="160"/>
    <arg name="lidar_range" default="20"/>
    <node name="reactive" pkg="reactive" type="reactive.py" output="screen">
        <param name="visualization" value="$(arg␣visualization)"/>
        <param name="max_speed"  value="$(arg␣max_speed)"/>
        <param name="min_speed"  value="$(arg␣min_speed)"/>
        <param name="velocity_gain"  value="$(arg␣velocity_gain)"/>
        <param name="steering_gain"  value="$(arg␣steering_gain)"/>
        <param name="disparity"  value="$(arg␣disparity)"/>
        <param name="safety_distance"  value="$(arg␣safety_distance)"/>
        <param name="lidar_angular_range"  value="$(arg␣lidar_angular_range)"/>
        <param name="lidar_range" value="$(arg␣lidar_range)"/>
    </node>
</launch>
```