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

Advanced Racing 2

2022-06-14

Student data

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

Transparency of contribution

- **Danilo Castiglia**: Trial of improvements on Disparity Extender, small bug fixes
- **Artur Chaves**: Disparity Extender tunning, Hardware
- **Severin Jäger**: Better speed controller and angle calculation for disparity extender, hardware, crashed the car
- **Johannes Windischbauer**: MIT particle filter, disparity extender velocity tuning, hardware

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

Advanced Racing 2

2022-06-14

Team data

Team number: 2

1 Goals

1. Enhance localisation with the MIT Particle Filter
2. Disparity Extender Velocity Tuning
3. Disparity Extender Improvements with Localization
4. Don't Break The Car

2 Implementation

2.1 MIT Particle Filter

Hours after hours were spent in an effort to get the MIT particle filter to run in the simulation which was ultimately unsuccessful because the used ROS version in the simulator was too high and there were issues with dependencies that were not resolvable. Nevertheless, we also tried to get the particle filter to run on the hardware. This failed due to an error message related to visualization and after spending some time on it we decided to solely focus on the disparity extender, as already enough time was wasted in this regard and the gains remained unclear as we focus on reactive methods anyways.

2.2 Disparity Extender Velocity Tuning

Even though our controller has only limited parameters, tuning them quickly to the race track is crucial for successful racing. To be well-prepared for the final race we tried to optimise the parameters for several available maps in the simulator. As there is a notable simulation-to-reality gap in the f1tenth simulator, we did the following:

- Reduce the simulator lidar update rate to 40 Hz (as on the car)
- Reduce the maximum car velocity to 4.5 m/s
- Measured lidar position on car incorporated in the simulator
- Use a SLAM map of Informatikhörsaal next to the provided maps

We were able to improve our lap times quite significantly, however in some cases this was obviously mainly due to simulator artefacts (e.g. due to limited computing power on weaker laptops leads to heavy oscillations on the car).

2.2.1 Linear Speed Improvements

Our initial algorithm for calculating the velocity worked quite well. However, after closer inspection we realized that the velocities that were calculated were usually quite a lot higher than the maximum speed of our car, resulting in a capping at a maximum value. Also, since we were only using the beam straight ahead multiplied by some value we had trouble with slippage after cornering when entering a longer stretch. The initial algorithm tended to send full speed commands resulting in some wheel spin and causing the car to react again to its overshooting and slowed down significantly when entering a straight stretch on an angle (The car is slightly turned to the wall so the straight beam is relatively short). Looking at these areas for improvement we added a velocity variable, a state so to say, to the car to increase, as well as decrease, the sent velocity commands slower. Unfortunately, this turned out to be less than optimal. If the threshold for our linear speed increases was too low, the reaction of the car was suddenly to slow causing crashes and if the threshold was to high it was prohibitive in regard to max speed and caused very early braking. This approach would have been tuneable with more than one threshold, but was discarded for bad performance.

Another aspect of our prior implementation was that it introduced a steering gain to mitigate oscillations when driving at higher speeds. The idea is to simply multiply the difference between the current car heading and the desired direction with a value smaller than one. This smoothes the trajectory notably, however it limits the performance in tight curves. Thus, we introduced an adaptive steering gain which decreases linearly from a maximum value at the minimum velocity to a minimum velocity at a critical velocity. We calculate the critical velocity as 0.4 times the maximum speed plus 0.6 times the minimum speed. As a result, the performance in steep curves is better while oscillations are still suppressed when going fast in long straights. However, if the difference between the minimum and the maximum steering gain is too large, there is the risk of heavy overshooting in the first curve after a long stretch. As the car brakes to steer, the steering already becomes more effective and at the same time the steering gain rises. One way to overcome this is not to use the desired velocity but the velocity from the odometry. Unfortunately, due to our crash we were not able to test this on hardware yet.

2.2.2 Enhanced Speed Controller

As our prior disparity extender implementation had a very naive way of calculating angle and speed from the lidar information which was only linearly proportional to the distance ahead of the car. This worked surprisingly well (and won the informal race during lab 5), however it comes with some major limitations:

- The car brakes even if it goes straight ahead
- When oversteering after a curve, the car is likely to face some wall. This will lead to reduced curve exit speed and thus longer lap times
- A linear speed reduction does not represent the cars braking behaviour

To alleviate some of these problems, we reworked the speed controller and come up with a little more sophisiticated solution which now considers three factors:

- The distance to a wall straight ahead d_{ahead}
- The distance of the point the car is aiming at (i.e. the farthest point in the disparity) d_{gap}
- The current steering effort

Still, the controller ensures a minimum and a maximum speed. We came up with the following calculation:

$$v = v_+ - (v_+ - v_-) \cdot \left(\frac{\alpha}{d_{ahead}^2} + \frac{\beta}{d_{gap}} + \frac{\gamma \theta^{1.5}}{\theta_{max}^{1.5}} \right)$$

The speed is proportional to the two distances and inversely proportional to the steering angle. The exponents are a result of trial and error, the quadratic term for the distance however reduces the influence of the wall. The new controller does not only feature the minimum and maximum velocity, but the three weights α, β, and γ which are not that obvious to tune. Whereas α represents the braking for a wall ahead, β causes braking if a disparity becomes short (i.e. no long stretch is ahead). The latter can mainly help with very sharp curves after long stretches

Map	Linear Controller	Enhanced Controller
f1_aute	24.3s	23.8s
f1_mco	46.8s	44.6s
Informatikhörsaal (SLAM)	12.1s	11.2s

Table 1: Comparison of the timing results of different algorithms in different maps on the simulator.

as the car will break already before it steers. Furthermore, γ ensures that the car drives slower while steering. The main parameters are α and γ which have to be traded off carefully with the steering gains (s. above) for fast racing. Figure 1 visualises the operation of this controller with $\beta = 0$. In most curves, the car slows down while steering (highlighted in red), however sometimes, it approaches the wall too closely, then the α term causes quite harsh braking to avoid a collision.

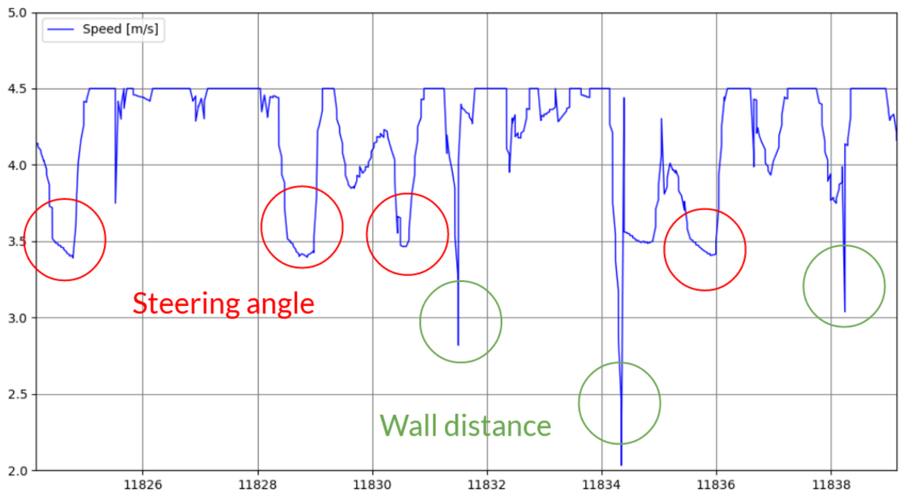
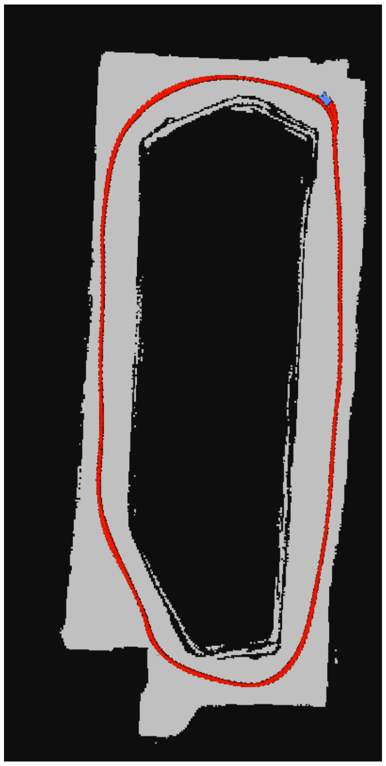


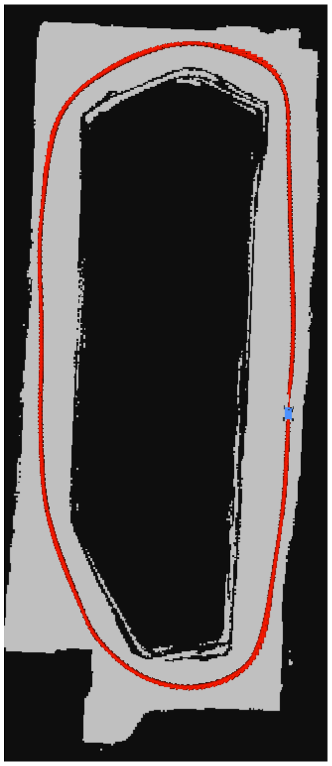
Figure 1: Velocity profile of a lap in the Informatikhörsaal map

We achieved the following lap times in the simulator:

Clearly, the enhanced controller is faster. From visual inspection, this is mainly due to the faster curve exit speed which was the main obvious limitation of our old algorithm. The racelines of the algorithms are shown in Figure 2. We think that the raceline of the extended algorithm is already quite good – it outperforms our planning-based algorithms from lab 6 significantly.



(a) Raceline with the linear controller



(b) Raceline with the enhanced controller

Figure 2: Trajectories in the Informatikhörsaal map

2.2.3 VESC Tuning

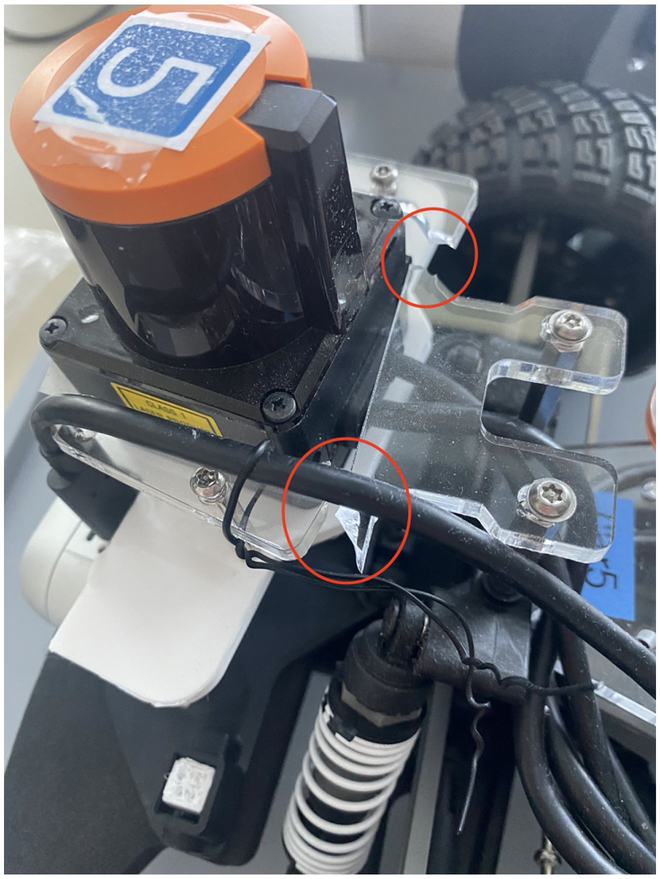
Tuning VESC parameters on the hardware car did not have much influence on the car's performance. We played with the motor min/max speed and the servo configuration but did not make any visible or measurable (in terms of lap time) gains.

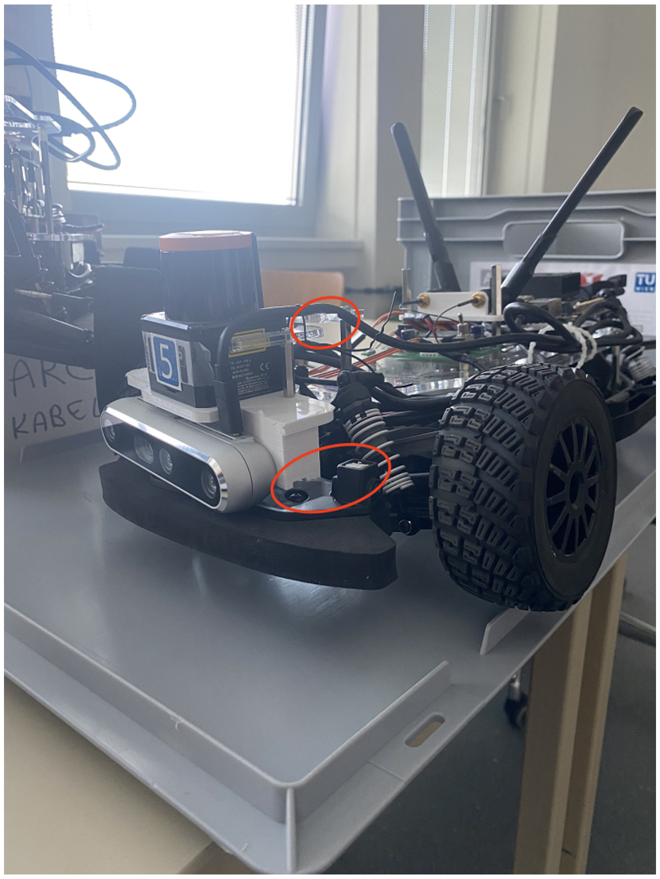
2.3 Disparity Extender Improvements with Localization

We aimed at enhancing the disparity extender to not only consider the single longest stretch but potentially several gaps (similar to follow the gap). This would allow us to decide on the next direction not only on the stretch length. Our original idea was to introduce some sort of preference (i.e. take rather the right or the left gap if unsure) which is calculated based on the current position. A simplification would be a global policy as most racetracks only have one (or only a few) situation where disparity extender decides incorrectly. Unfortunately, due to problems with our localisation stack (as outlined above) and due to the lack of time in the exam period we did not implement this. However, at least the simple variant would greatly help for the final race as the track might contain a trap for reactive algorithms.

2.4 Don't Break The Car

Unfortunately, we've had an incident that left us no choice but to end hardware testing early. A misconfigured speed controller caused a crash into the wall which broke the mounts for our front-mounted lidar sensor.


(a) Top view of the back of the sensor


(b) View of the left front of the car

Appendix

Listing 1: Improved disparity extender implementation

```
#!/usr/bin/env python3
from __future__ import print_function
from re import T
import sys
import math
import numpy as np

#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

# 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

BASIC_VELOCITY = rospy.get_param('/reactive/linear_velocity', False)  # simple
    ↪ velocity scheme with linear gain
VELOCITY_GAIN = rospy.get_param('/reactive/velocity_gain', 0.6)

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)

BRAKE_FOR_STRETCH_END_GAIN = rospy.get_param('/reactive/
    ↪ brake_for_strech_end_gain', 0.6)
BRAKE_TO_STEER_GAIN = rospy.get_param('/reactive/brake_to_steer_gain', 0.8)
BRAKE_BEFORE_CRASH_GAIN = rospy.get_param('/reactive/brake_before_crash_gain',
    ↪ 1.0)

# Up to this velocity, the steering gain decreases linearly (from
    ↪ MAX_STEERING_GAIN to MIN_STEERING_GAIN). Fixed as no critical parameter
V_CRIT = (0.6*MAX_SPEED + 0.4*MIN_SPEED) - MIN_SPEED
MIN_STEERING_GAIN = rospy.get_param('/reactive/min_steering_gain', 0.5)
MAX_STEERING_GAIN = rospy.get_param('/reactive/max_steering_gain', 0.8)

# Lidar params (fixed)
LIDAR_MIN_DIST = 0.06  # m
LIDAR_RANGE = 10  # m
```

```
# TODO
DYNAMIC_SAFETY_DIST = rospy.get_param('/reactive/dynamic_safety_distance',
    ↪ False)

# When the distance in front is less than this, the car turns (so far unused)
MIN_DISTANCE_TO_TURN = 99999

# 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

    scans = np.array(data.ranges)
    scans = np.clip(scans, LIDAR_MIN_DIST , LIDAR_RANGE)
    lidar_data = data
    lasers = int(len(scans) * LIDAR_ANGULAR_RANGE / 270)
    self.skipped = int((len(scans) - lasers) / 2)
    filtered = scans[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 = ranges.copy()
    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 = np.median(ranges_list[int(len(ranges_list)/2)-2:int(len(
            ↪ ranges_list)/2)+2])
        farthest = ranges_list.index(max(ranges_list))

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

        if BASIC_VELOCITY:
            velocity = straight * VELOCITY_GAIN
        else:
            if ranges[farthest] > 6.0:
                dist_func = 0
            else:
                # TODO consider time behaviour, e.g. only use this if declining
                dist_func = 1/(ranges[farthest] + 0.001)

            if straight > 3.5:
                crash_func = 0
            else:
                crash_func = 1/(straight**2 + 0.001)

            if angle <= 6:  # do not brake for small angles
                angle_func = 0
            else:
                angle_func = np.clip(angle**1.5, 0, MAX_STEERING_ANGLE**1.5)/
                    ↪ MAX_STEERING_ANGLE**1.5
            # print(dist_func, " ", angle_func, " ", crash_func)
            velocity = MAX_SPEED - (MAX_SPEED-MIN_SPEED)*np.clip(
                ↪ BRAKE_FOR_STRETCH_END_GAIN*dist_func+ BRAKE_TO_STEER_GAIN*
                ↪ angle_func + BRAKE_BEFORE_CRASH_GAIN*crash_func, 0.0, 1.0)

        velocity = np.clip(velocity, MIN_SPEED, MAX_SPEED)

        # TODO uses velocity from odometry here
        # reduces oscillations when going at higher speeds
        if velocity > V_CRIT:
            angle = MIN_STEERING_GAIN * angle
        else:
            angle = (MIN_STEERING_GAIN + (1-(velocity-MIN_SPEED)/V_CRIT) * (
                ↪ MAX_STEERING_GAIN-MIN_STEERING_GAIN)) * angle

        # Keep angle in [-180, 180]
        angle = ((angle + 180) % 360) - 180

        # Angle clipping. Probably handled by VESC anyway, but improves plot
            ↪ readablity
        angle = np.clip(angle, -MAX_STEERING_ANGLE, MAX_STEERING_ANGLE)
        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)

    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)
```