1 Calculate Distance to Nearest Obstacle on Lane (10 Points)

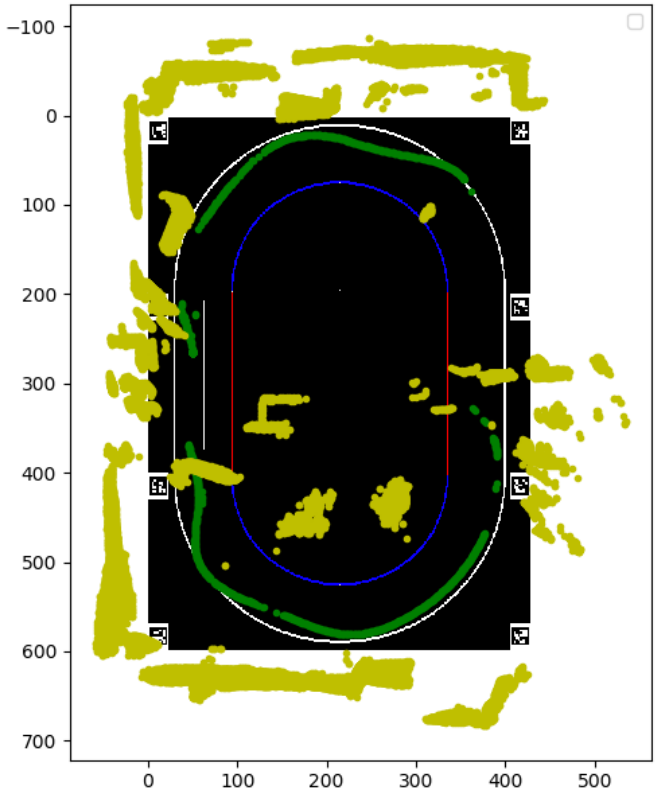

Abbildung 1: Karte mit Lokalisationen des Autos (grün) und erkannte Objekte des LIDARs (gelb).

- Quellcode: `https://github.com/bigzed/model_car/blob/version-4.0/texinput/src/localization.py`
- Video: `https://github.com/bigzed/model_car/blob/version-4.0/texinput/videos/U11_video2.MP4`

Der LIDAR erkennt alle Objekte innerhalb von 1.5 m vor dem Auto. Dies umfasst alle Punkte, die eine positive X-Koordinate im 'laser'-Koordinatensystem haben. Die LIDAR Daten sind kontinuierlich, allerdings benötigt der Algorithmus die Daten der Odometrie um sie in ein gemeinsames Koordinatensystem zu transformieren.

Wie man an den fehlenden grünen Punkten im Plot erkennen kann, hat das Positionssystem teilweise lange Verzögerungen und es kann auch zum kompletten Ausfall einer Kamera während einer Fahrt kommen. In der abgebildeten Fahrt fiel wohl die mittlere Kamera aus, was durch einen Neustart behoben werden konnte.

Das Auto ist in dieser Fahrt ungefähr an Koordinate (80|400) losgefahren und hat die Strecke gegen den Uhrzeigersinn befahren. Auf der rechten Geraden erkennt man die Ausweichbewegung beginnend etwa bei (400|400) um das Hindernis und auf der oberen Hälfte der linken Geraden etwa bei (60|280) das Stoppen vor dem Hindernis auf beiden Spuren.

Zu beachten ist, dass die anscheinend komplette Blockade beider Spuren auf der rechten Geraden durch ein nachträgliches Verschieben des Hindernisses von der inneren auf die äussere Spur entstanden ist, während das Auto sich bereits in der nachfolgenden Kurve befand. Der LIDAR-Plot ist daher an dieser Stelle irreführend.

1. Für die Transformation benutzen wir *tf.transformations* mit *TransformBroadcaster* und *sendTransform* und *TransformListener* sowie *transformPoint*. Die LIDAR Daten werden dann in eine *PointCloud2* umgewandelt und durch die Transformation in das Koordinatensystem der Odomotetrie übertragen.

2. Wir benutzen die Funktion *closest_point()* von Übungsblatt 10 mit einigen Änderungen an der Funktion für die Bereiche im Kreis und einem geringen Distanzwert von 0.3 m.
   Der Lenkeinschlag wird aus den Winkeln zwischen Orientierung des Autos (yaw) und dem Vektor vom Auto zum *closest_point* bestimmt. Diesen Winkel nehmen wir als Fehler für einen PD-Controller welcher den Lenkwinkel bestimmt.

3. In unserer *lane_is_free* Funktion wird für jedes erkannte Hindernis des LIDARs getestet ob es auf der derzeitigen Spur liegt. Falls nicht wird die Spur beibehalten.
   Falls ein Hindernis auf dieser Spur ist überprüft der Algorithmus die andere Spur und das Auto wechselt die Spur, wenn diese frei ist. Sollte diese ebenfalls belegt sein hält es an, bis eine der Spuren wieder frei wird.