1 Time and Precision (10 Punkte)

- Quellcode: `https://github.com/bigzed/model_car/blob/version-4.0/texinput/src/localization_assignment12.py`
- ohne Hindernisse:
  - Zeit: unter $7s$
  - Video: `https://raw.githubusercontent.com/bigzed/model_car/version-4.0/texinput/videos/u12_1.mp4`
- mit Hindernisse:
  - Zeit: ca. $11s$
  - Video: `https://raw.githubusercontent.com/bigzed/model_car/version-4.0/texinput/videos/u12_2.mp4`

1.1 Herangehensweise

Wir haben zwar volle Punktzahl auf dem 11. Zettel bekommen, allerdings weist unser Programm natürlich noch Fehler auf. Daher haben wir uns für diesen Zettel entschieden keine weiteren Sensoren zu benutzen, sondern weiter auf das *fake GPS* für die Steuerung und den LIDAR für die Hinderniserkennung zu verlassen.

1.1.1 Steuerung

Während des 11. Zettels sind uns folgende Probleme zur Steuerung aufgefallen:

- sehr späte Entscheidungen bzw. manchmal auch keine. Wir vermuteten Packetverluste
- zu schwacher Lenkwinkel
- falsche Entscheidungen in der oberen linken Kurve
- keine Möglichkeit während des Fahrens Geschwindigkeit und Look-Ahead zu konfigurieren

Um diese Probleme zu lösen haben wir uns folgendes überlegt:

- Als erstes haben wir den Kreismittelpunkt des oberen Kreises von (215|196) auf (215|200) verschoben, da anscheinend die reale Welt nicht genau mit der Karte übereinstimmt.
- Als nächstes haben wir die Lenk-Entscheidungen im oberen Kreissektor überprüft und uns ist aufgefallen, dass unsere Berechnung des Winkels zwischen Auto-Vektor und Ziel-Vektor nicht in jedem Sektor korrekt sind. Manchmal erhielten wir nicht den kleineren, sondern den gröseren Winkel zwischen den beiden Vektoren.
- Als nächstes haben wir den Ausschlag des Lenkwinkels erhöht, sodass dieser seinen maximalen Wert schon bei einem Winkel von Ziel zu Auto von 90° erreicht.
- Nach der Korrektur dieser Punkte konnten wir 400 Umdrehungen als Geschwindigkeit für den Motor einstellen und sicher fahren
- Nun haben wir die Hinderniserkennung von der Steuerung entkoppelt und sie beide in eigene Callbacks aufgeteilt, somit stehen die Lenkentscheidungen schneller zur Verfügung.
- Zusammen mit einer GUI die wir erstellt haben um die Parameter *desired_speed*, *look_ahead_curve* und *look_ahead_straight* während des Fahrens zu Verändern konnten wir die optimale Konfiguration ermittlen.
- Bei einer Geschwindigkeit von 1000 Umdrehungen, brauchen wir einen *look_ahead* von $120cm$ um das Resultat aus Video 1 zu erreichen. Bei der Kalibrierung ist uns aufgefallen, das der *look_ahead* der selbe für Kurven und Geraden seien sollte.

1.1.2 Hinderniserkennung

Während des 11. Zettels sind uns folgende Probleme zur Hinderniserkennung aufgefallen:

- LIDAR Position zu QR-Code Position zu Vorderachse nicht kalibriert
- trotz Erkennung kam es zu Kollisionen, da das Ausweichmanöver oder die Bremsung zu spät bzw. zu schwach ausgeführt wurde

Um diese Probleme zu lösen haben wir uns folgendes überlegt:

- Als erstes haben wir die Position zueinander kalibriert und die Callbacks wie im letzten Abschnitt beschrieben entkoppelt. Das hat die Genauigkeit erhöht, allerdings sind die Entscheidungen teilweise immer noch zu spät.
- Deswegen haben wir uns entschieden, sobald wir ein Hindernis erkennen und ein Ausweichmanöver einleiten, die Geschwindigkeit zu verringern und die Lenkung um den Faktor 2 zu verstärken. Das hat dazu geführt das trotz später Erkennung eine Kollision verhindert werden kann. Es bekämpft allerdings nur das Symptom nicht den Grund. Wir haben dies in der `def get_lane(self, car_x, car_y):` bei der Abfrage `if self.lane_is_free(obstacles, self.lane_id):` eingebaut.
- Der Grund für die späte Entscheidungen sind die Timestamps der *fake GPS* Datenpakete. Um die LIDAR-Daten in das *fake GPS* Koordinatensystem umzurechnen, brauchen wir eine GPS Information zu fast der selben Zeit, allerdings laufen auf den Autos und dem *fake GPS* anscheinend keine *NTP* Clienten. Die Fluktuation der Timestamps zum Empfangen auf dem Auto geht von $-3s$ bis $+3s$.
- Dieses Problem ist allein auf dem Auto nicht lösbar, allerdings können wir die Auswirkungen verringern.
- Als erstes ersetzen wir den Timestamp des Pakets mit dem Zeitpunkt des Empfangens und eliminieren die Fluktuation, nehmen dabei aber in Kauf das LIDAR und GPS falsch ineinander überführt werden.
- Als nächstes warten wir für jedes LIDAR-Paket bis zu $150ms$ ob ein passendes GPS-Paket empfangen wird, falls nicht brechen wir ab und schreiben eine Nachricht auf die Konsole.
- Wir konnten so zwar die Pakete synchronisieren, allerdings warten wir schon bis zu $150ms$ bis wir mit der Berechnung anfangen, somit sind die Ausweichmanöver meist noch zu spät.
- Die nächsten Kurs-Teilnehmer würden sich bestimmt freuen wenn die Zeit auf Autos und *fake GPS* durch einen NTP-Server synchronisiert wird.