Car Controll with PC

This application allows you to controll a ANKI Overdrive car with a PC.
Please be concearned that the code is depended on bluepy, so it's only work on Linux system.

Requirements:

- Linux machine with bluetooth (or Raspberry Pi)
- Python 3.4+
- bluepy

Build & Run:

- Clone Project: git clone https://gitlab.informatik.fb2.hs-intern.de/informatikprojektdeegenerws23/c2cprojekt.git
- navigate to Fahrzeugsteuerung_PC

```
• cd Fahrzeugsteuerung_PC
• cd AnkiOverdrive
```

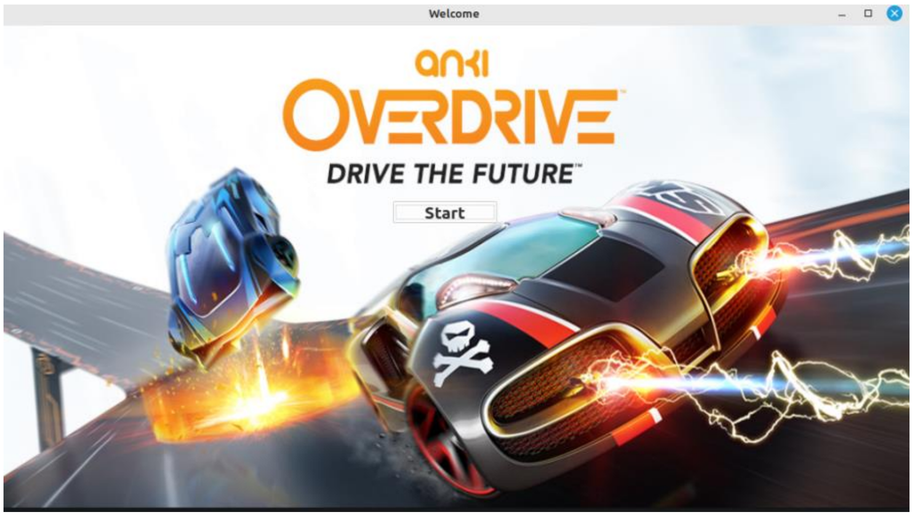
- you can either execute it in the terminal or run it in a IDE (recommended: Pycharm)

- for Terminal
1. execute `Start_Project.py`
2. follow the instructions from the terminal.
3.

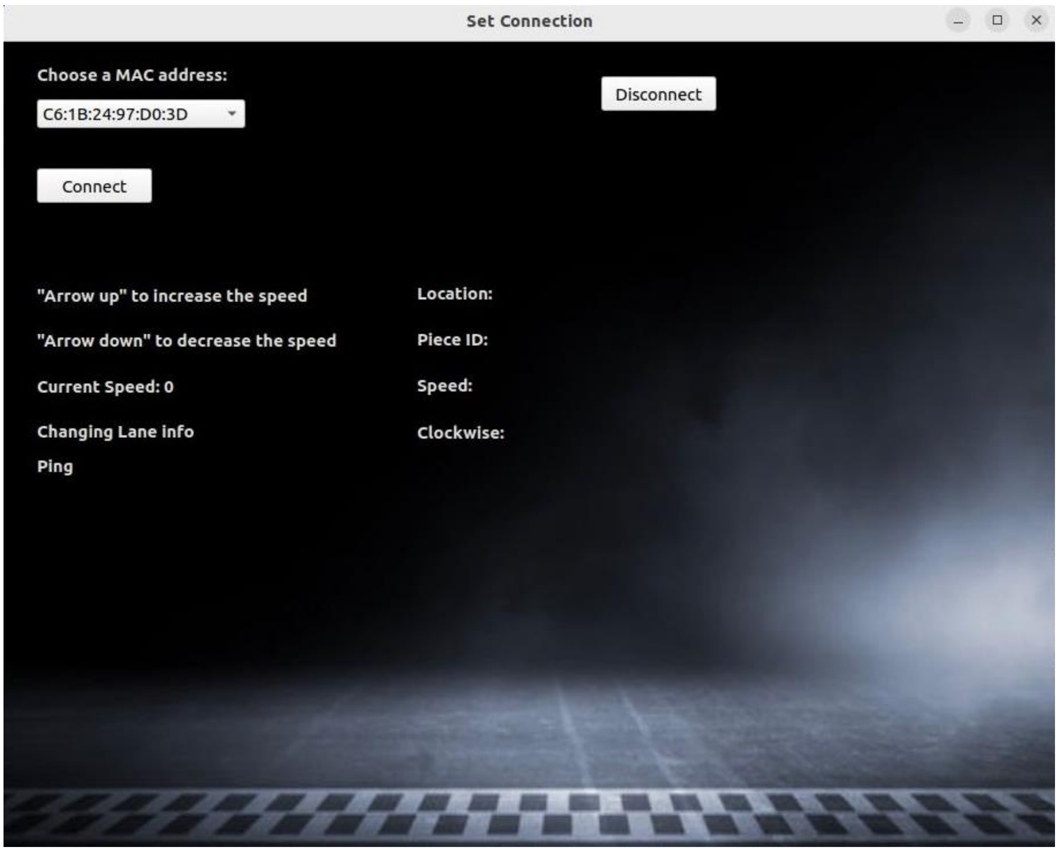
- on IDE
1. open Project in IDE
2. follow the IDE instructions!
3. execute the file: `Start_Project.py`

How to drive

After starting the application, following Screen should open:

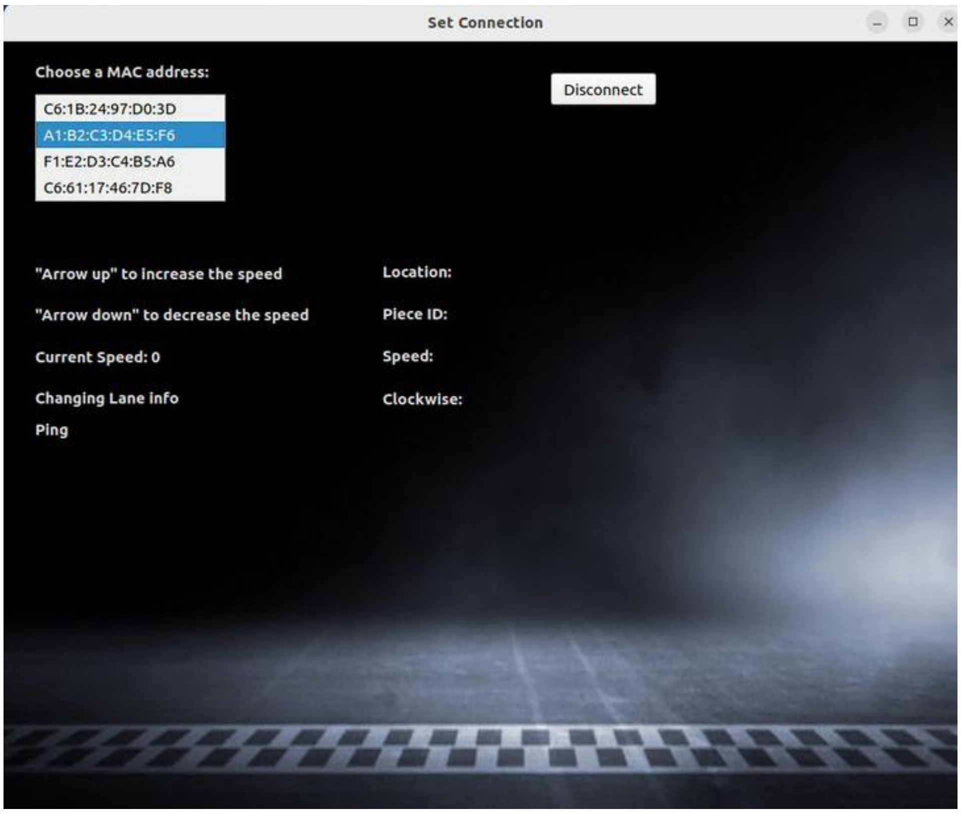


This is the welcome page for users. To proceed, users need to click on the "Start" button. After clicking “Start”, following Screen should open:

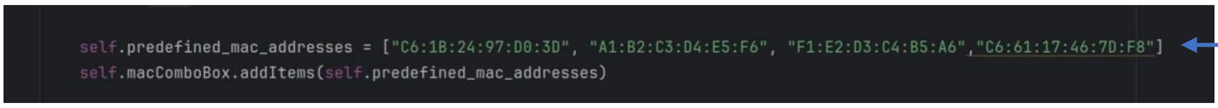


This is the primary page:
All information and settings for the car are displayed here.

- To connect the PC with the car, you must click on the dropdown box and choose the car.

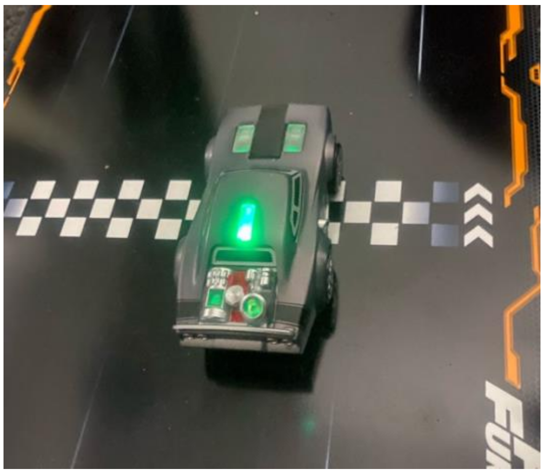


The dropdown box contains most of the available cars.
To add a new car, you can do so in the SetConnectionWindow class within the code.

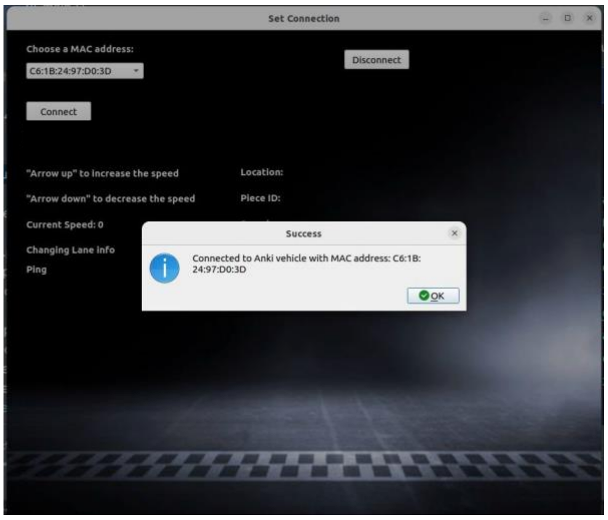


```
self.predefined_mac_addresses = ["C6:1B:24:97:D0:3D", "A1:B2:C3:D4:E5:F6", "F1:E2:D3:C4:B5:A6","C6:61:17:46:7D:F8"]
self.macComboBox.addItems(self.predefined_mac_addresses)
```

After selecting your desired car, you can establish a connection by clicking the "Connect" button.
Before the connection is established, the light will blink green, indicating that it is ready to connect.



After connecting successfully, a notification will confirm it and you are ready to drive the car.


Set Connection

Choose a MAC address: C6:1B:24:97:D0:3D

"Arrow up" to increase the speed
"Arrow down" to decrease the speed
Current Speed: 0
Changing Lane info
Ping

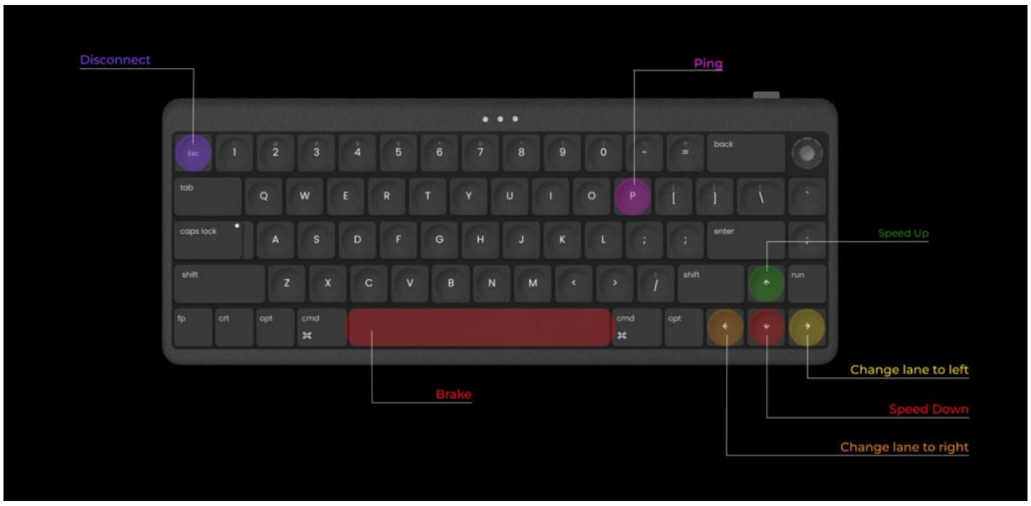
Location:
Piece ID:

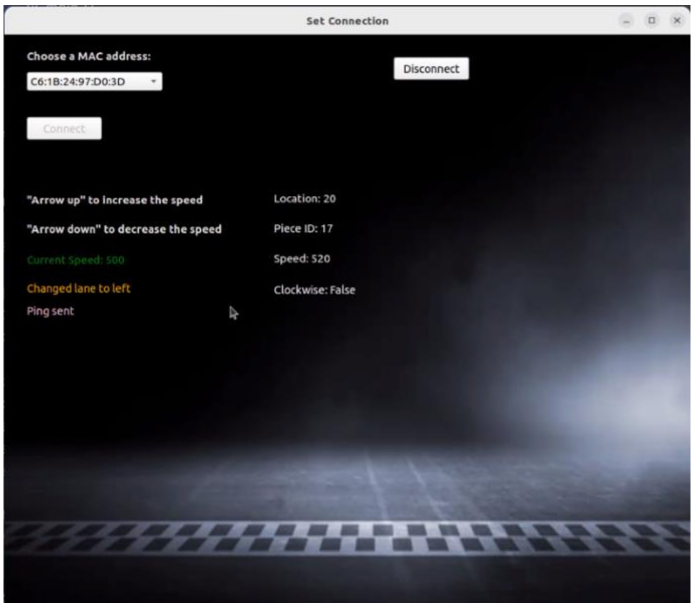
Success

Connected to Anki vehicle with MAC address: C6:1B:24:97:D0:3D

From now on, the car light will blink blue, indicating a successful connection.

The instructions are listed on this page, but essentially, pressing the "arrow up" key increases speed, while pressing the "arrow down" key decreases speed. Pressing the "arrow right" or "arrow left" keys changes lanes. Pressing the space key applies the brake, bringing the speed to 0.




Set Connection

Choose a MAC address:

C6:1B:24:97:D0:3D

Disconnect

Connect

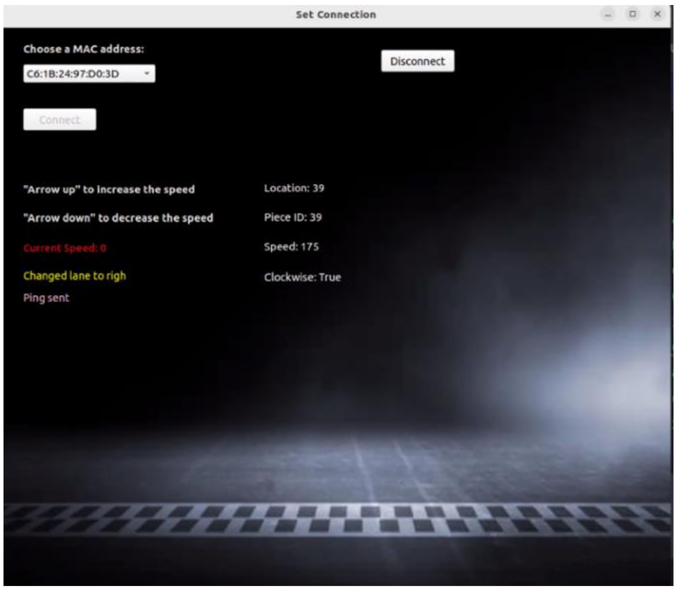
"Arrow up" to increase the speed — Location: 20

"Arrow down" to decrease the speed — Piece ID: 17

Current Speed: 500 — Speed: 520

Changed lane to left — Clockwise: False

Ping sent


Set Connection

Choose a MAC address:

C6:1B:24:97:D0:3D

Disconnect

Connect

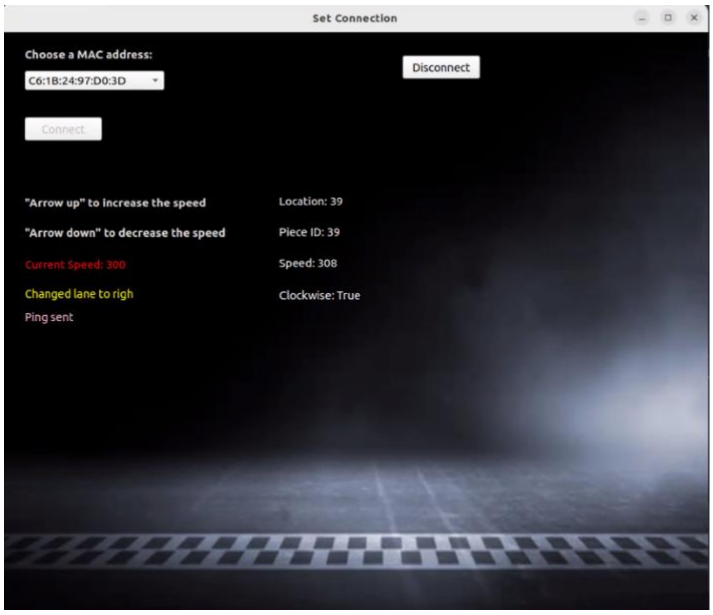
"Arrow up" to increase the speed — Location: 39

"Arrow down" to decrease the speed — Piece ID: 39

Current Speed: 0 — Speed: 175

Changed lane to righ — Clockwise: True

Ping sent


Set Connection

Choose a MAC address:

C6:1B:24:97:D0:3D

Disconnect

Connect

"Arrow up" to increase the speed — Location: 39

"Arrow down" to decrease the speed — Piece ID: 39

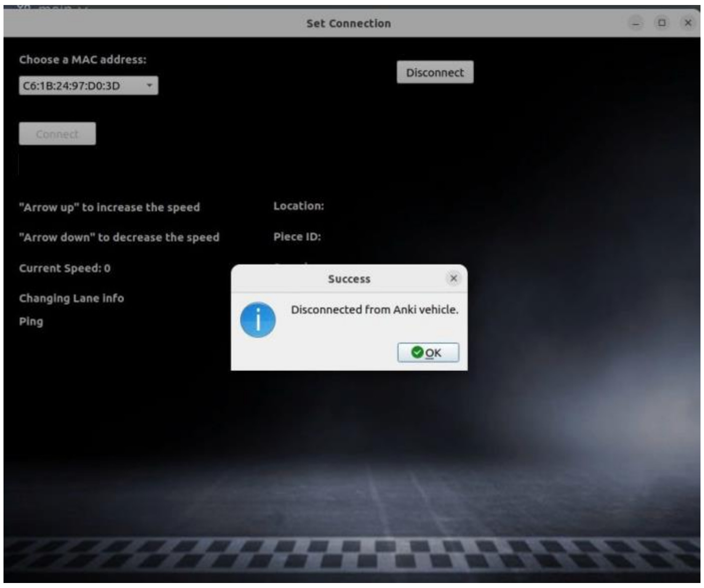
Current Speed: 300 — Speed: 308

Changed lane to righ — Clockwise: True

Ping sent

1. The labels "Current Speed" and "Changing Lane info" display the current speed in increments of 100 (100 indicates very slow, while 1000 indicates very fast) and indicate if the car has changed lanes to the left or right.
2. Other Labels include:
    - Location: GPS coordinates indicating the car's current position.
    - Piece ID: Identifies the track piece the car is on.
    - Speed: Indicates the car's actual speed.
    - Clockwise: Indicates whether the car is driving clockwise.

Once you've finished driving, you can click the "Disconnect" button to disconnect the car from the PC.


Set Connection

Choose a MAC address:

C6:1B:24:97:D0:3D

"Arrow up" to increase the speed

"Arrow down" to decrease the speed

Current Speed: 0

Changing Lane info

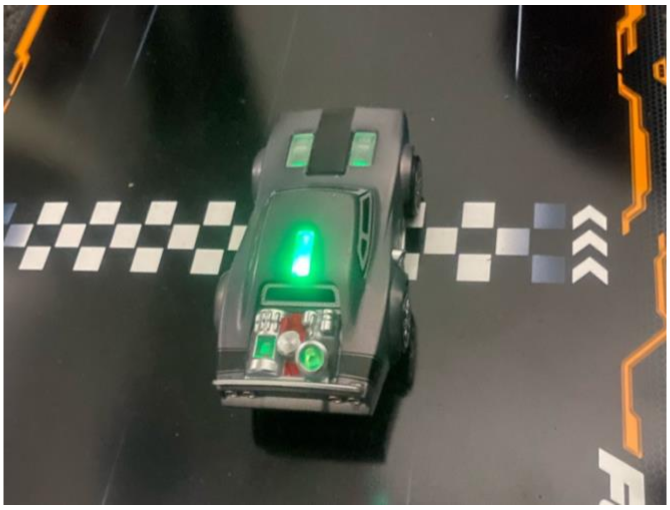
Ping

Location:

Piece ID:

Success

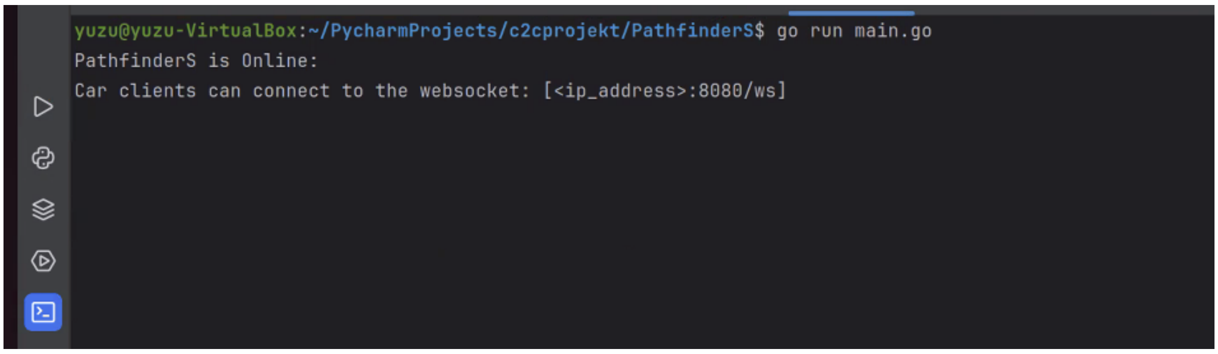
Disconnected from Anki vehicle.



The car will blink green again, signaling that it is ready to connect.

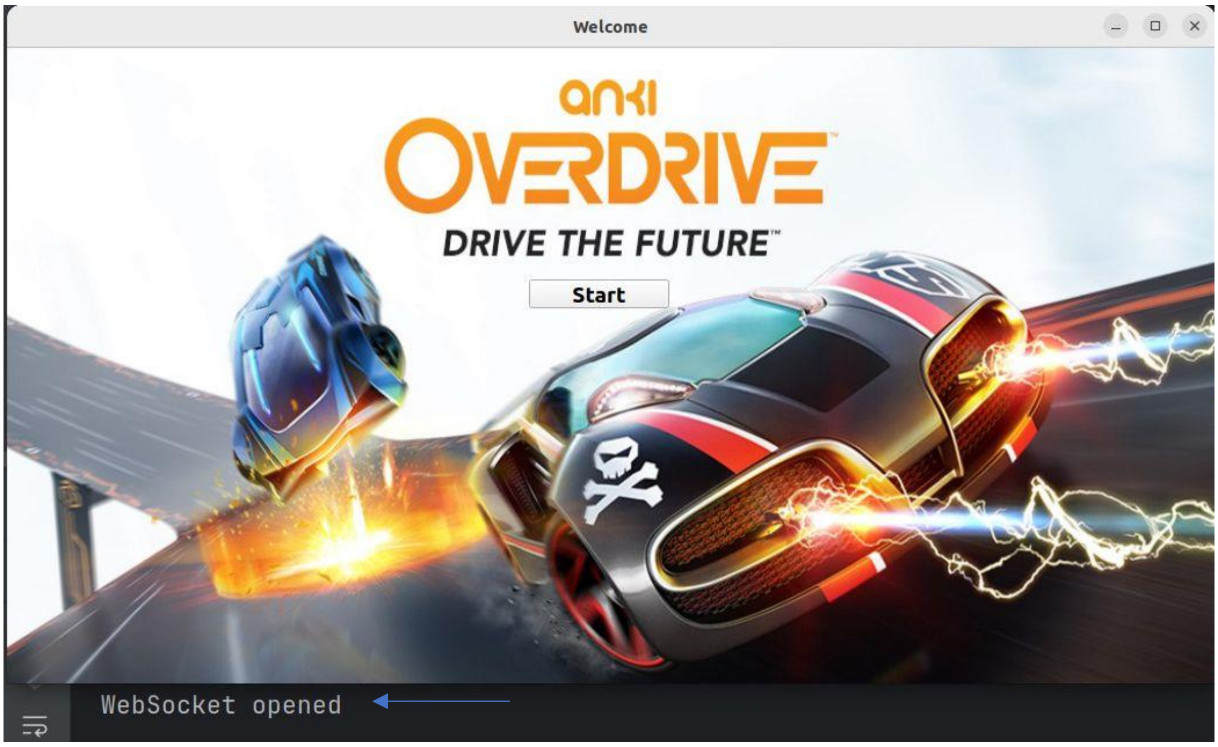
How to communicate

Before launching the application, ensure that the server main.go is running. Navigate to the PathfinderS directory from the terminal and execute the following command: go run main.go.



```
yuzu@yuzu-VirtualBox:~/PycharmProjects/c2cprojekt/PathfinderS$ go run main.go
PathfinderS is Online:
Car clients can connect to the websocket: [<ip_address>:8080/ws]
```

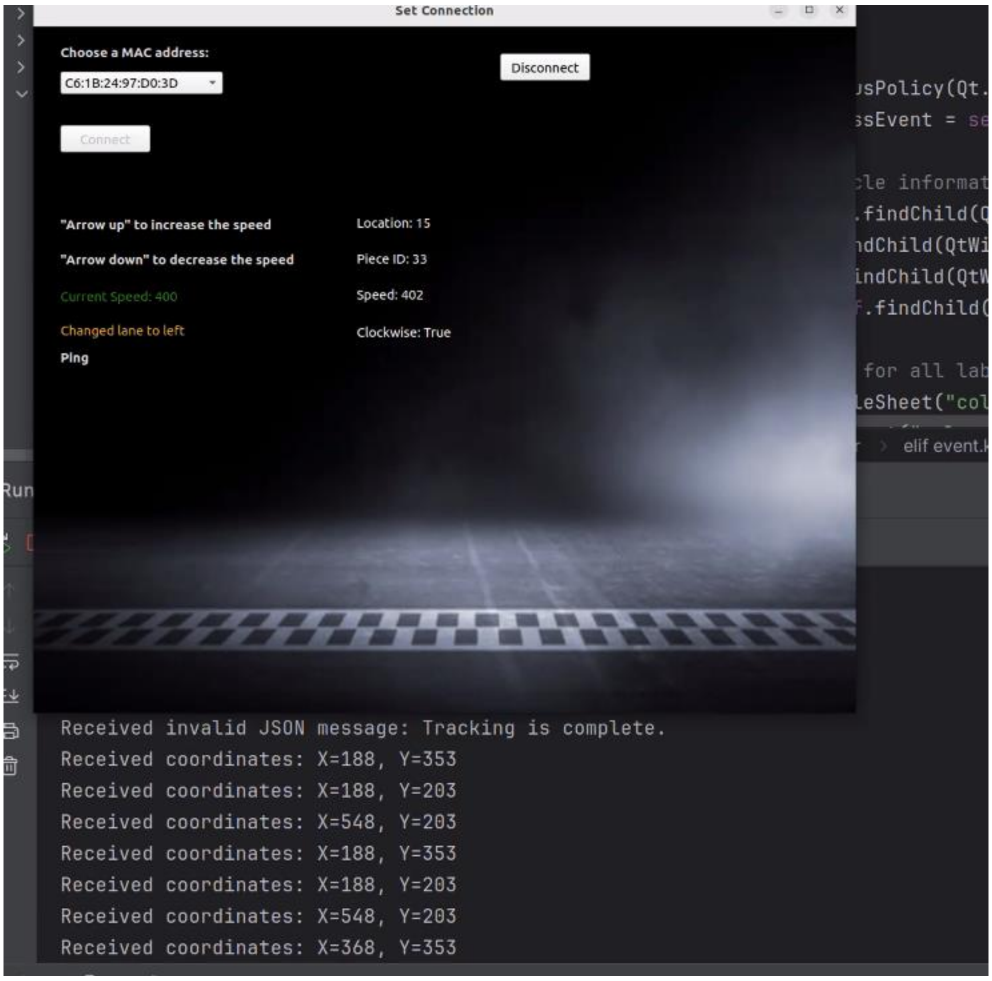
Once the server is successfully connected, you can start the application. A confirmation message will appear, indicating the establishment of the websocket connection.



```
WebSocket opened
```

First, you may receive invalid json messages, which indicates that the car is not receiving any messages from the server, likely because the car has not completed a lap yet.

The server can only process and send location data once it receives messages from the car.



"Arrow up" to increase the speed
"Arrow down" to decrease the speed
Current Speed: 400
Changed lane to left
Ping

Location: 15
Piece ID: 33
Speed: 402
Clockwise: True

```
Received invalid JSON message: Tracking is complete.
Received coordinates: X=188, Y=353
Received coordinates: X=188, Y=203
Received coordinates: X=548, Y=203
Received coordinates: X=188, Y=353
Received coordinates: X=188, Y=203
Received coordinates: X=548, Y=203
Received coordinates: X=368, Y=353
```

Once the car finishes a lap, you will receive the coordinates of the car in the console.

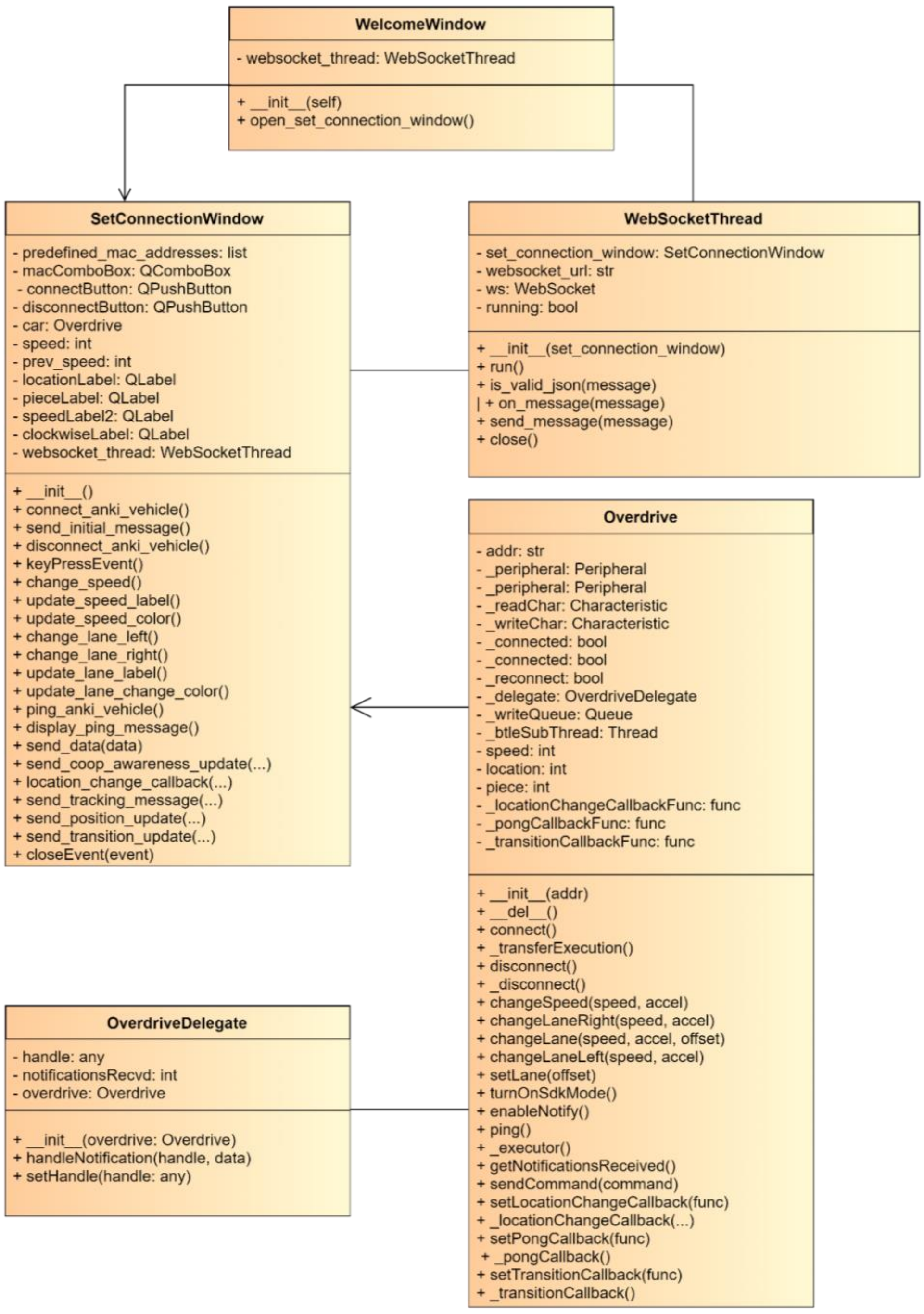
Other minor information are commented in the code.

Application Overview:

The application is designed using Qt5 for creating intuitive and user-friendly interfaces. Qt-Designer is utilized for designing and modifying UI files, enabling easy adjustments to the application's visual components.

Python Code Explanation:

Below is a class diagram illustrating the relationships between all classes within the application, along with explanations for the methods utilized in each class.



Class WelcomeWindow Methodes

1. **__init__(self):**
   Constructor method for the WelcomeWindow class. It initializes the window by loading the UI from the "welcome_window.ui" file, connects the startButton click signal to the open_set_connection_window slot, and starts the WebSocket thread.

2. **open_set_connection_window(self):**
   Method triggered when the "Start" button is clicked. It creates an instance of the SetConnectionWindow class, shows it to the user, and closes the current window (`WelcomeWindow`).

Class SetConnectionWindow Methodes

1. **_init__(self)** in **SetConnectionWindow** class**:
   This method serves as the constructor for the `SetConnectionWindow` class. It initializes the UI by loading the `.ui` file, sets up predefined MAC addresses, connects button signals to respective slots, initializes variables, and starts the WebSocket thread.

2. **connect_anki_vehicle(self)**:
   This method is triggered when the "Connect" button is clicked. It attempts to establish a connection with the Anki vehicle using the selected MAC address. If successful, it sets up callbacks, sends an initial message, and disables the Connect button.

3. **disconnect_anki_vehicle(self)**:
   This method is called when the "Disconnect" button is clicked. It disconnects the Anki vehicle, updates the UI, and enables the Connect button.

4. **keyPressEvent(self, event)**:
   This method handles key events. It allows controlling the vehicle using arrow keys for speed control, lane changes, stopping the vehicle, sending ping commands, making U-turns, and disconnecting the vehicle.

5. **change_speed(self)**:
   This method sends a command to change the speed of the Anki vehicle based on the current speed set in the UI.

6. **change_lane_left(self)** and **change_lane_right(self)**:
   These methods send commands to the Anki vehicle to change lanes to the left or right, respectively.

7. **ping_anki_vehicle(self):**
   This method sends a ping command to the Anki vehicle to check its responsiveness.

8. **location_change_callback(self, addr, location, piece, speed, clockwise)**:
   This callback method is invoked when the location of the Anki vehicle changes. It updates the UI with the new location, piece ID, speed, and clockwise direction. Additionally, it sends position and transition updates to the server and CoopAwarenessUpdate messages.

9. **send_position_update(self, location_id, road_piece_id, offset_from_road_center_mm, speed_mm_per_sec)**:
   This method sends a position update message to the server with information about the vehicle's location, road piece ID, offset from the road center, and speed.

10. **send_transition_update(self, road_piece_id, previous_road_piece_id, offset_from_road_center_mm, driving_direction):**
    This method sends a transition update message to the server when the vehicle transitions between road pieces. It includes information about the current and previous road piece IDs, offset, and driving direction.

11. **closeEvent(self, event)**:This method is overridden to handle the window close event. It stops the WebSocket thread and disconnects from the Anki vehicle before closing the window.

class WebSocketThread Methods:

1. **__init__(self, set_connection_window):**
   This method serves as the constructor for the WebSocketThread class. It initializes the WebSocketThread object with the URL of the WebSocket server, initializes the WebSocket connection, and sets up the running flag to control the thread execution.

2. **run(self):**
   This method is the main thread function. It continuously tries to establish a WebSocket connection with the server and listens for incoming messages. Upon receiving a valid JSON message, it calls the on_message method to handle it.

3. **is_valid_json(self, message):**
   This method checks if the received message is a valid JSON format by attempting to parse it. If parsing succeeds, it returns True; otherwise, it returns False.

4. **on_message(self, message):**
   This method handles incoming WebSocket messages. It decodes the JSON message, extracts coordinates if present, and prints them to the console. This method can be extended to perform additional actions based on the received message.

5. **send_message(self, message):**
   This method sends a JSON-encoded message to the WebSocket server if the connection is established. It introduces a delay before sending the message to avoid flooding the server.

6. **close(self):**
   This method closes the WebSocket connection gracefully. It is called upon encountering an exception during WebSocket operations or when the WebSocketThread object is terminated.

class Overdrive Methodes:

1. **__init__(self, addr):**
   This method initializes an Anki Overdrive connection object and attempts to connect to the specified Bluetooth MAC address. It sets up various attributes and starts the connection process.

2. **__del__(self):**
   This is the destructor method for the Overdrive object, responsible for cleaning up resources when the object is deleted.

3. **connect(self):**
   Initiates a connection to the Anki Overdrive. It sets up communication characteristics and enables notifications.

4. **disconnect(self):**
   Disconnects from the Anki Overdrive.

5. **changeSpeed(self, speed, accel):**
   Changes the speed of the Anki Overdrive car.

6. **changeLaneRight(self, speed, accel):**
   Switches the car to the adjacent right lane.

7. **changeLaneLeft(self, speed, accel):**
   Switches the car to the adjacent left lane.

8. **changeLane(self, speed, accel, offset):**
   Changes the lane of the car by providing an offset.

9. **setLane(self, offset):**
   Sets the internal lane offset (unused).

10. **turnOnSdkMode(self):**
    Turns on SDK mode for the Anki Overdrive.

11. **enableNotify(self):**
    Enables notification for receiving updates from the car.

12. **ping(self):**
    Sends a ping command to the Anki Overdrive.

13. **_executor(self):**
    Notification thread responsible for handling communication with the car.

14. **getNotificationsReceived(self):**
    Returns the count of notifications received.

15. **sendCommand(self, command):**
    Sends a raw command to the Anki Overdrive.

16. **setLocationChangeCallback(self, func):**
    Sets the callback function for location change events.

17. **_locationChangeCallback(self, location, piece, speed, clockwise):**
    Wrapper function for the location change callback.

18. **setPongCallback(self, func):**
    Sets the callback function for pong events.

19. **_pongCallback(self):**
    Wrapper function for the pong callback.

20. **setTransitionCallback(self, func):**
    Sets the callback function for piece transition events.

21. **_transitionCallback(self):**
    Wrapper function for the piece transition callback.

class OverdriveDelegate:

1. **__init__(self, overdrive):**
   initializes the OverdriveDelegate object with a reference to the Overdrive object.

2. **handleNotification(self, handle, data):**
   Callback method invoked when a notification is received. It processes different types of notifications, such as location updates, transitions, and pongs.

3. **setHandle(self, handle):**
   Sets the handle for the delegate to handle notifications.