**1. プロジェクト概要**  
1/10スケールのアッカーマン型ラジコン車両を自律走行させるための経路（waypoint）作成・管理ツール。車両の物理特性（最小回転半径）や障害物を考慮した経路生成をサポート。最終的にJSON形式で保存し、実機制御に使用可能。

**2. 使用しているファイル構成**

project/

│

├─ waypoint\_editor.py # GUIツール、waypoint作成・編集用

├─ cource\_map.py # コース情報、グリッドマップ、障害物、パイロン情報など

├─ <scenario\_name>.json # 各シナリオのwaypointデータ（保存先）

└─ その他ライブラリ

├─ matplotlib

├─ numpy

└─ scikit-image（skimage） # line関数による障害物チェック

**cource\_map.py の内容**

* grid\_matrix : 2D numpy 配列でコースの通行可能/不可を定義
* world\_to\_grid(x, y) : メートル単位座標 → グリッド座標
* grid\_to\_world(x, y) : グリッド座標 → メートル単位座標
* obstacles : 障害物リスト
* start\_lines : スタートライン情報
* pylons : パイロン位置情報
* （以前は start\_pos / goal\_pos を描画に使用していたが現在は非表示）

**3. 実現できること（waypoint\_editor.py）**

1. **waypoint作成・編集**
   * マウスクリックでwaypointを追加
   * スライダーでwaypoint選択
   * 選択waypointの速度設定（-100～100、初期値100）
   * 選択waypointの削除
   * 全waypointのクリア
2. **障害物・壁・パイロン回避**
   * クリック時に障害物・壁・パイロン上のwaypoint追加を制限
   * 直前のwaypointとの間に障害物・パイロンをまたぐ場合も追加不可
3. **車両特性を考慮した制限**
   * 最小回転半径（デフォルト0.7 m）以下の曲線を描くwaypoint追加を制限
   * 3点から円を計算して曲率を評価
4. **経路の可視化**
   * グリッドマップ上にwaypointと経路を描画
   * 選択waypointは赤丸、他はマゼンタ丸
   * 背景に障害物（緑）、パイロン（オレンジ）、スタートライン（青）を描画
5. **データ保存**
   * waypointsをJSON形式で保存
   * 保存時に自動でyaw（向き）を計算して付与
   * JSONは将来的にラズパイでの自律制御に利用可能

**4. 操作マニュアル（簡易）**

1. ターミナルで起動：　python waypoint\_editor.py
2. シナリオ名を入力　ファイル名のことです。
3. GUI画面上でwaypointをクリックして追加
   * 赤いエラー表示で無効なクリックを通知
4. 左下のボタン操作
   * **Clear**：全waypoint削除
   * **Delete**：選択waypoint削除
   * **Save**：waypoint保存（yaw自動計算）
5. 右上のスライダー
   * **Waypoint Index**：waypoint選択
   * **Speed**：選択waypointの速度設定

**5. 将来的に追加できる機能案**

* 始点・終点を特別マーカーで表示して視覚効果
* ウェイポイント間の距離や最小回転半径に基づく自動補正・最適化
* JSONに加えてRVizやGazebo用のパス形式出力
* IMU初期オフセットの補正値管理
* 実機ラズパイでの自律走行連携（PCA9685経由ステアリング・スロットル制御）

◆制御フロー

waypoint\_editor.py ──> <scenario\_name>.json

│

▼

IMU/初期位置補正

│

▼

実機制御ループ

┌──────────┐

│ 位置取得 │

│ IMU │

└─────┬────┘

▼

┌──────────┐

│ 経路追従 │

│ Pure Pursuit │

└─────┬────┘

▼

┌──────────┐

│ ステアリング │

│ スロットル │

└─────┬────┘

▼

他、例外処置で考えられること

　・衝突スタック判定からの復帰

　・逆走判定からの復帰

　・540°の失敗判定からの復帰

　・障害物回避（他車両との関係）・・・できるのか？

次waypointに到達

│

…ループ…そして終了

C:\Users\DELL\python\

│

├─ temp完成後削除/ # 一時作業フォルダ（不要になったら削除）

├─ \_\_pycache\_\_/ # Python自動生成キャッシュ

├─ ト技会のサンプル/ # 参考サンプルコード集

├─ 作成したドキュメント/ # マニュアルや仕様書など

│

├─ adjust\_alignment.py # モータPWMアライメント調整用スクリプト

├─ alignment\_parameter.txt # アライメント補正パラメータ

├─ cource\_map.py # コース情報（壁・パイロン・スタートラインなど）

├─ imu\_sense\_thread.py # IMU（BNO055）取得用クラス・スレッド

├─ main\_control\_loop.py # メイン制御ループ（WAYPOINT走行、障害物回避など）

├─ motor\_drive.py # モータドライブ（PWM制御）クラス

├─ pc\_waypoint\_monitor.py # PC側Waypoint進捗モニタリングGUI

├─ quarify.json # シナリオ設定やクエリ情報

├─ ultrasonic\_array\_thread.py#超音波センサ5個まとめて取得するクラス・スレッド

├─ vehicle\_interface.py # 車両センサー・アクチュエータ統合インターフェイス

├─ waypoint\_editor.py # Waypoint作成・編集GUI

├─ \_\_init\_\_.py # パッケージ初期化

2. 各モジュールの概要

ファイル 役割 主な機能

cource\_map.py コース情報管理 壁・パイロン・スタートライン位置、グリッド変換関数

waypoint\_editor.py Waypoint作成GUI マウスクリックでWaypoint作成、速度・yawの設定、保存

vehicle\_interface.py 車両インターフェイス 超音波センサ・IMU・モータドライブを統合、バックグラウンド更新対応

ultrasonic\_array\_thread.py 超音波センサ制御 5個の超音波センサ値を順次取得、名前付きで提供

imu\_sense\_thread.py IMU取得 BNO055 9軸IMU値をシリアル通信で取得、スレッド更新

motor\_drive.py モータ制御 PWM制御でアクセル・ステアリング、アライメント補正反映

adjust\_alignment.py アライメント調整 PWMパラメータ調整用スクリプト

main\_control\_loop.py 車両制御 Waypoint走行、障害物回避、例外復帰処理、スタート/ストップ制御

pc\_waypoint\_monitor.py PCモニタリング MatplotlibでWaypoint進捗を可視化、進捗監視・復帰操作用GUI

quarify.json シナリオ設定 Waypoint走行や条件分岐の設定値格納

3. 制御フロー（概要）

3.1 起動・初期化

main\_control\_loop.py 起動

vehicle\_interface.py により以下を初期化：

超音波センサ（5個）

IMU (BNO055)

モータPWM（アクセル/ステアリング）

アライメント補正を adjust\_alignment.py からロード

Waypointファイルをロード（waypoint\_editor.pyで作成済み）

3.2 メインループ

Waypoint走行（Pure Pursuitなど）

現在位置（IMU+オドメトリ）からターゲットWaypointを追跡

モータドライブ命令を生成（速度・舵角）

障害物回避

超音波センサ値に基づき衝突回避行動

例外処理

スタック検出 → 後退・旋回で復帰

逆走検知 → 進行方向修正

パイロン回転失敗検出 → 復帰処理

データ記録

センサ値（超音波・IMU）

現在Waypoint進捗

モータPWM値

停止条件

最終Waypoint到達

遠隔コマンドによる停止

3.3 遠隔・モニタリング

PCで pc\_waypoint\_monitor.py を起動

Waypoint進捗をMatplotlib GUIで確認

必要に応じて車両中断・再開指示

将来的にはBluetoothなどで遠隔操作も可能

4. 操作マニュアル（簡易）

Waypoint作成

waypoint\_editor.py を起動

マウスでWaypoint追加、速度設定

保存すると <scenario\_name>.json に出力

車両起動

main\_control\_loop.py をラズパイ上で起動

「Enter」キーで走行開始

走行中の監視

PCで pc\_waypoint\_monitor.py を起動

Waypoint進捗を確認

緊急停止/復帰

PC GUI または将来のBluetoothコマンドで中断・再開

データ確認

record\_data.csv にセンサ・PWM・時間情報が保存

停止後

モータ停止、GPIOリセット

必要に応じて再走行

5. 注意事項

IMUのドリフトに注意：長時間走行では位置誤差が蓄積

超音波センサは同時発射不可（干渉防止）

モータアライメントは事前調整必須

PC側での進捗確認が必須：特に例外復帰処理時

◆課題

実車での超音波・IMUキャリブレーションと誤差確認

逆走、スタック、パイロン失敗の復帰アルゴリズム完成

遠隔スタート/ストップ、中断・再開の実装

実車でのWaypointパラメータチューニン・ルックアヘッド）