Tanmay Goyal

Pure Pursuit controller of an Autonomous Gokart

Take-me-home functionality

Semester Project

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology Zurich

Supervision

Alessandro Zanardi
Prof. Dr. Emilio Frazzoli

April 2021

IDSC-proj-sp-goyal

IDSC

Abstract

The purpose of this technical report is to explain the implementation of the Adaptive Pure Pursuit control algorithm on autonomous gokart for path tracking. This algorithm is further extended to make the gokart move in reverse mode, as explained in the report. This extension is used to develop the Take-me-home functionality for the gokart, using which it can return to the origin from any deadlock or crashed position and perform Inverse-parking manoeuvre by which it can self dock itself. This report also provides the geometric derivations of the concept, implementation details, and the comparative results of using different parameters in pure pursuit controller, as well as a performance comparison with the existing Model Predictive Controller.

Keywords: Adaptive Pure Pursuit, Take-me-home functionality, Inverse-parking manoeuvre.

Contents

Chapter 1

Introduction

1.1 Motivation

The autonomous gokart platform is developed by the group of Prof. Dr. Frazzoli at the Institute for Dynamic Systems and Control at ETH Zurich. The primary goal of this semester thesis is to develop a robust adaptive pure pursuit controller for tracking path points and compare different path tracking algorithms implemented for the gokart to understand the pros and cons of all the controllers. With the knowledge of these results, the user can always decide the controller to be used in different situations.
The MPC controller is computatinally expensive and difficult to implement. On the other hand, Pure Pursuit Controller is simple to implement and computationally inexpensive as is a Model free Geometric control algorithm. Owing to robustness of the pure pursuit controller, it can also be used as baseline for Path tracking and AIDO competitions.
The secondary goal of the thesis is to develop the Take-me Home functionality for the gokart that is essential for driving completely driverless. This can further help in performing the inverse parking manoeuvre for the gokarts to dock the gokarts autonomously using the reverse mode.

1.2 Related Works

In [1], a model predictive contouring controller for the gokart has been developed. For this, a novel track formulation based on quadratic B-splines was defined, and a tricycle model was identified. This is used for extracting the centreline to be tracked. The controller developed in [1] was extended in [2] with an assistive torque controller. To achieve this, a second-order model to describe the steering dynamics was identified. This is used for converting the steering reference angle into the torque, which can be given as input to the gokart's steering wheel. A double-track vehicle model of the gokart was derived in [3], and a first simulator was developed. Extensive work has been done in developing the Pure Pursuit Controller [4] as it performs well in tracking the path at lower speeds. Pure Pursuit Controller was introduced in 1985 and was originally used to calculate arc necessary to get a robot back onto a path. It was started being used widely for path tracking since 1990s because of robustness, simple to implement computationally inexpensive, purely geometric with model free dynamics, small tracking error for non-zero curvature and can be used to implement different modes.

1.3 Contribution

Previously only the Model Predictive Controller was available for the gokart to track path and move autonomously. In this thesis, a robust adaptive pure pursuit controller was developed, giving

the users an alternate and simpler to implement choice for tracking paths at lower speeds. The implementation of the pure pursuit controller is based on the available literature and was further improved by using an adaptive pure pursuit path tracking algorithm in which the lookahead distance is dynamically calculated according to the velocity of the gokart. This implementation was integrated with the available software stack and pipeline of the autonomous gokarts. To ensure the algorithm's robustness, it was tested extensively at different speeds, parameters, and weights.
Once the robustness of the controller was ensured, and ideal parameters were realised, the work further extended to enable gokart to track the path and move autonomously in the reverse mode also. This extension allowed the gokart to perform inverse parking manoeuvres. Integrating the different modes of the adaptive pure pursuit controller enabled the development of Take-me Home functionality which was the end target of the thesis.

1.3.1 Manuscript Organization

- **Pure Pursuit Controller** [2], in this chapter, the general idea behind the Adaptive Pure Pursuit Controller, proof of the concepts for different modes of the runs, and their implementation details are explained in depth.
- **Take-me Home Functionality** [3], in this chapter, an explanation of the functionality and its implementation details are given.
- **Results and Conclusions** [4], in this chapter, the detailed results of all the test runs with different parameters, the conclusion of the test runs, and a comparison of different controllers are reported.

1.3.2 Abbreviations, Notations, and Mathematical symbols

- (X, Y) - Coordinates of the lookahead or goal point
- L - Lookahead distance
- R - Radius of Curvature on which gokart is supposed to move
- D - Difference of R and X
- P - Path points gokart has to follow
- γ - Curvature on which gokart is supposed to move
- δ - Steering angle
- θ - Angle between X axis and front wheel direction
- L_{car} - Car length between axles
- ICR - Instantaneous center of rotation of the gokart while turning
- X_{cur} - Current position of gokart
- V_{cur} - Current velocity of gokart
- L_p - Lookahead point
- C_p - Center Path point
- P_{record} - Path recorded that the gokart has to follow from deadlock position till it reaches the realignment point
- *start_controller* - Boolean variable telling if the gokart has crashed
- *realigned* - Boolean variable telling if the gokart has realigned itself with centerline
- *origin* - Starting point of the gokart

Chapter 2

Pure Pursuit Controller

2.1 Overview

Pure pursuit [4], [5] is a path tracking algorithm that enables a moving vehicle to go from its current position to a goal position by moving it on a calculated curvature. It takes input a set of path points that it has to travel and the current state. The algorithm then calculates the goal position [6] which is at a fixed distance (1 lookahead distance) ahead on the path. The implementation was further improved by using the Adaptive Pure Pursuit algorithm [7] in which the lookahead distance is calculated based on the current velocity of the gokart. It has to reach this goal position from the current position using the calculated curvatures and steering angles. This method is called pure pursuit as in the entire algorithm, the vehicle is seen chasing the goal point which keeps on changing. So in a way, it is pursuing that moving goal point by moving towards it and hence it is called so. In real-world also, humans drive in the same i.e they look some distance ahead from their current position, decide a goal position that they have to reach and then move towards it. This lookahead distance changes as the speed of the vehicle, curvatures of the road and vision occlusion changes. The detailed analysis of the results of these parameter changes on the algorithm can be seen in the Chapter 4.

2.2 Theoretical Derivation

2.2.1 Terminologies

- **Center Path Point** - It is the point on the path to be followed, which is closest to the center of the rear axle of the vehicle (closest assumed position of the IMU on the gokart).
- **Lookahead Distance** - It is the euclidean distance between center path point and lookahead point. The goal position is found to be 1 lookahead distance from the Center Path Point.
- **Lookahead Point** - It is the point on the path to be followed, which is closest to the 1 lookahead distance from the Center Path Point. It is also referred to as Goal Point.
- **Center of curvature** - It is the center of a circular arc joining the Center Path Point and Lookahead Point. The cord length of this circular arc between the 2 points is the approximate lookahead distance.

2.2.2 Proof of Concept

Consider Figure 2.1, the vehicle is shown with its rear axle's center as the origin and the X-Y axes. The point (X,Y) is the lookahead point shown to be at 1 lookahead distance (L) from the origin.

The goal is to calculate the curvature of the arc joining the Center Path Point to the Lookahead Point.

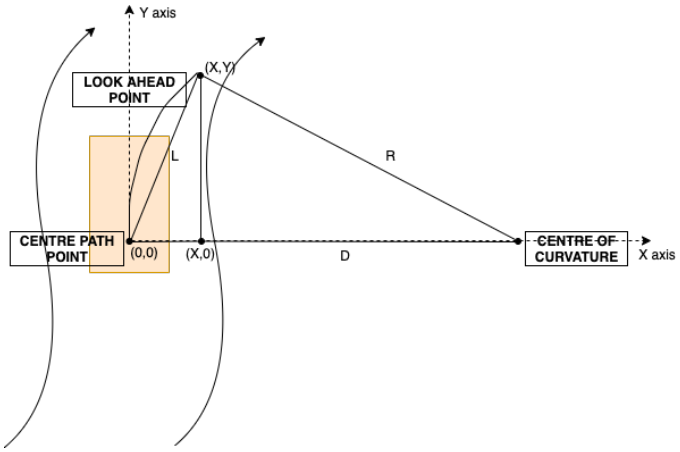


Figure 2.1: Geometry for Pure Pursuit Derivation

$$X^2 + Y^2 = L^2 \tag{2.1}$$

Equation 2.1 describes a circle around the origin with a set of points at 1 lookahead distance from the origin.

$$X + D = R \tag{2.2}$$

Equation 2.2 describes the relationship between the radius of the curvature of the arc and the X-offset of the lookahead point.

$$D^2 + Y^2 = R^2 \tag{2.3}$$

Equation 2.3 describes a circle around the center of curvature with a set of points at 1 radius of curvature from the center of curvature.
The intersection of the Equation 2.1 and Equation 2.3 gives the lookahead point.

$$(R - X)^2 + Y^2 = R^2 \tag{2.4}$$

Equation 2.4 is obtained by substituting D from Equation 2.2

$$R^2 - 2 \times R \times X + X^2 + Y^2 = R^2 \tag{2.5}$$

Equation 2.5 is obtained from expanding the Equation 2.4

$$2 \times R \times X = L^2 \tag{2.6}$$

Equation 2.6 is the final result of solving the Equation 2.5

$$\gamma = 1/R = (2 \times X)/L^2 \tag{2.7}$$

Equation 2.7 gives the final expression for the curvature of the required arc.

Consider Figure 2.2, showing the ICR, arc and the front and rear of the vehicle. In the Figure 2.2, δ is shown as the angle through which the wheel is required to turn based on the curvature of the arc. The objective is to calculate this δ in the term of γ. This is derived using the Kinematic Bicycle Model of the gokart.

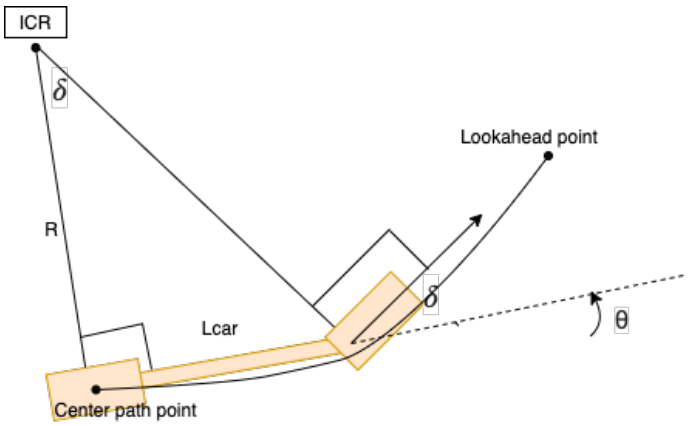


Figure 2.2: Geometry for Bicycle Model Derivation

$$R = Lcar/\tan\delta \tag{2.8}$$

Equation 2.8 gives the basic tangent relation between the Lcar, Radius and the δ.

$$\delta = \arctan\left(2 \times X \times Lcar\right)/L^2 \tag{2.9}$$

Equation 2.9 gives the steering angle by expanding the Equation 2.8 and Equation 2.7.

$$\delta = \arctan\left(\gamma \times Lcar\right) \tag{2.10}$$

Equation 2.10 gives the relation between the steering angle and the curvature of the arc.

2.3 Assumptions

In this thesis, the following assumptions are considered for developing the controller and the tests with the gokart [8].

- This algorithm in this implementation is only for path following and tracking rather than adaptive cruise control or lane-keeping based on white line recognition.

- Paths are defined as a series of discrete path-points. A path point (x, y, tangent at the point) represents a position and a direction on a two-dimensional space.
- Localization is performed using scan matching by 3D LIDAR and a three-dimensional map. The location is assumed to be highly accurate.
- Curvature is calculated such that Center of Curvature is on the axis perpendicular gokart's heading direction and passing through Center of Gravity of the gokart.

2.4 Algorithm

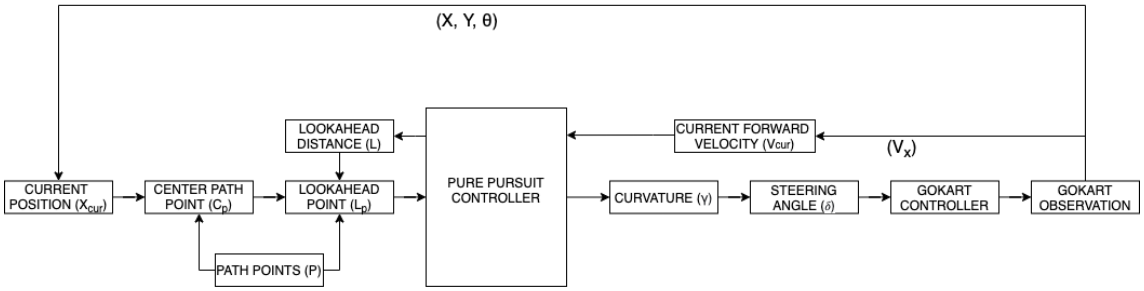
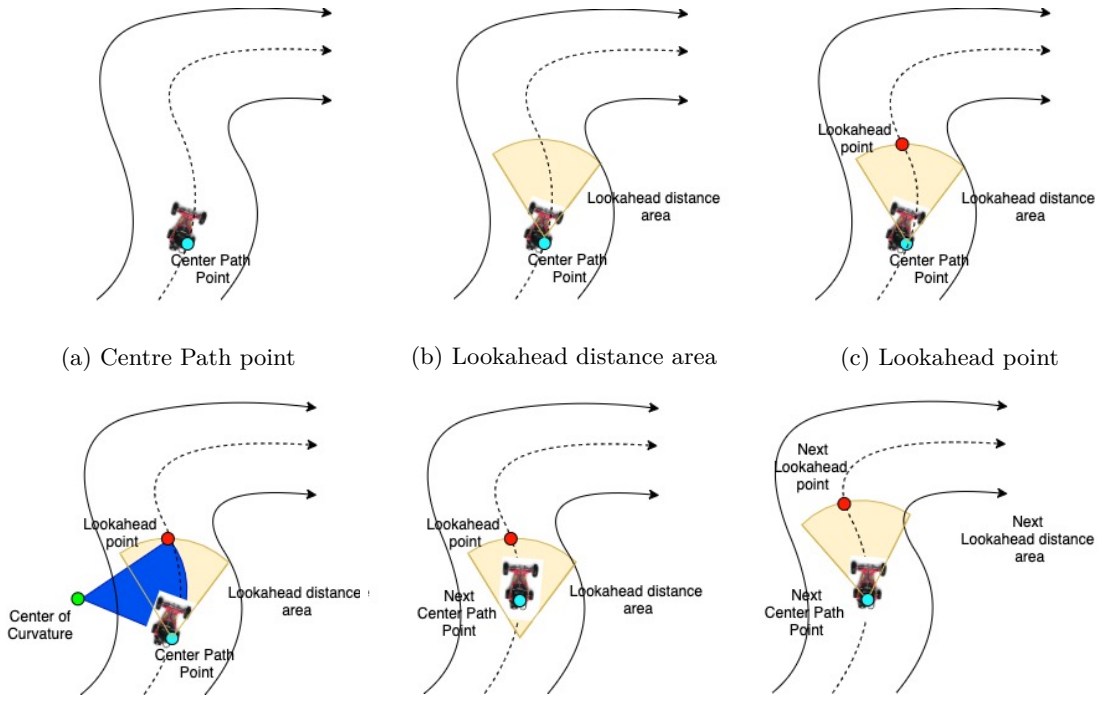


Figure 2.3: Algorithm Flowchart

- Gokart's current position and path points are used to calculate the center path point.
- Center path point, path points and lookahead distance are used to calculate the lookahead point. The lookahead distance is calculated by the Equation 4.1 by the adaptive pure pursuit controller as can be seen in the Figure 2.3.
- The lookahead point is given as input to the pure pursuit controller which calculates the radius of the curvature and the curvature to be followed using the Equation 2.7.
- From the curvature, the steering angle is calculated using the Equation 2.10 and given as an input to the gokart's steering controller.
- Once the gokart reaches close to the lookahead point, it starts looking for a new lookahead point and fixes the new lookahead point based on the new position and velocity.
- Then the gokart calculates the new radius of curvature and steering angle which is then given as input to the steering controller at the new instance. The algorithm can be visualised in Figure 2.4

Algorithm 1: Pure Pursuit Algorithm

Input: Path points P, Gokart's position X_{cur}, Gokart's velocity V_{cur}, Gokart's Length L_{car}
Output: Steering Angle δ
initialization;
while *P, X_{cur} and V_{cur} is available* **do**
    $C_p \leftarrow$ find_near_point(P, X_{cur});
    if $C_p = L_p$ **then**
        $L \leftarrow$ get_lookahead_distance(V_{cur});
        $L_p \leftarrow$ find_goal_point(P, X_{cur}, L);
    end
    $\gamma \leftarrow$ get_curvature(L_p, X_{cur});
    $\delta \leftarrow$ compute_steering_angle(γ, L_{car});
end



(a) Centre Path point (b) Lookahead distance area (c) Lookahead point

(d) Curvatures of movement for gokart (e) Gokart reaches near lookahead point (f) New lookahead distance and lookahead area

Figure 2.4: Pure Pursuit algorithm

2.5 Implementation

2.5.1 Functions

- **update_path** - This function is the callback function of the rostopic "/kitt/path/path_array" that stores the path points to be followed.
- **update_location** - This function is the callback function of the rostopic "/kitt/se2_localization /gokart_state" that updates the current location of the gokart.
- **update_velocity** - This function is the callback function of the rostopic "/kitt/se2_localization /gokart_state" that updates the current velocity of the gokart.
- **find_nearest_point_index_on_path** - This function returns the index of the Centre Path Point on the path based on the gokart's current position.
- **find_goal_point** - This function returns the Lookahead Point on the path based on the gokart's current position and the lookahead distance.
- **get_curvature** - This function returns the curvature the gokart has to follow based on the Centre Path Point and Lookahead Point.
- **compute_steering_angle** - This function returns the steering angle the gokart has to be given based on the radius of curvature to be followed by gokart.
- **compute_motors_cmd** - This function returns the output of the PID velocity controller that has to be published to the gokart motor command controller based on the gokart's current and desired velocities.

- **update_parameters** - This function updates the pure pursuit controller parameters - *"desired velocity, minimum approach distance, lookahead distance"* for the gokart based on the mode in which gokart has to move in.

2.5.2 Forward Mode

Parameters

- **v_desired** - This is the desired velocity to be followed in forward mode. It has been tested to perform well in between 2-4 m/s.
- **look_ahead_distance** - This is the lookahead distance based on the desired velocity of the gokart. By observation and runs as explained in Section 4.3.2, it was calculated to be -

$$look\_ahead\_distance = v\_desired + 1 \tag{2.11}$$

- **min_distance** - This is the minimum approach distance reaching which the gokart starts to look for the next lookahead point. By observation and runs, it was calculated to be -

$$min\_distance = look\_ahead\_distance \times 0.55 + 0.76 \tag{2.12}$$

2.5.3 Reverse Mode

Theoretical Derivation

Consider Figure 2.5, the vehicle is shown with its center as the origin and the X-Y axes. The point (X,Y) is the reverse lookahead point shown to be at 1 lookahead distance from the origin in the reverse direction. The goal is to calculate the curvature of the arc joining the Center Path Point to the Reverse Lookahead Point. The only difference between the forward and the reverse modes are the different parameters of the controller used.

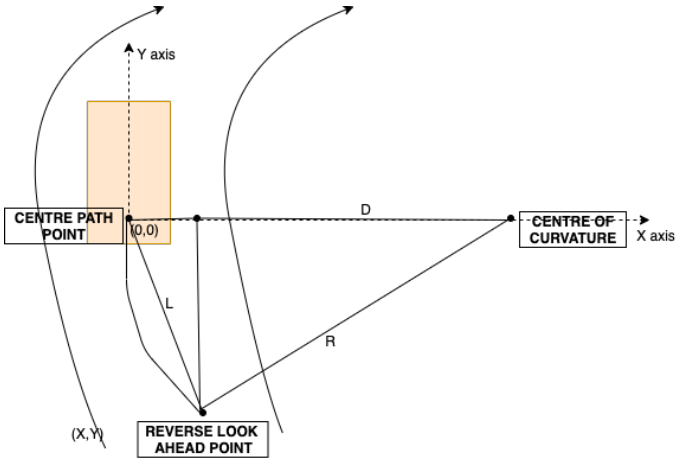


Figure 2.5: Geometry for Pure Pursuit Derivation in Reverse Mode

Consider Figure 2.6, showing the ICR, arc and the front and rear of the vehicle. In the Figure 2.6, δ is shown as the angle through which the wheel is required to turn based on the curvature of the arc. The objective is to calculate this δ in the term of γ. The only difference between the forward and the reverse mode in the steering angle calculation is that in reverse mode, the wheels don't directly point towards the lookahead point, but actually points along an arc that joins the Center Path Point and the Reverse Lookahead Point.

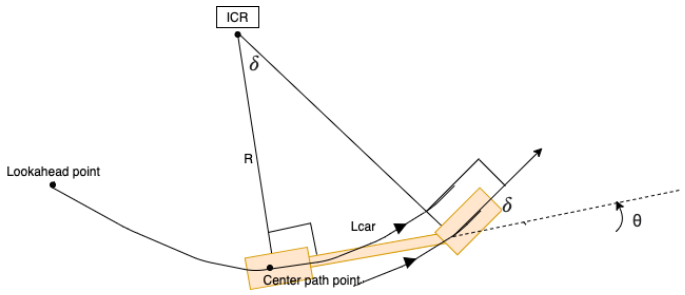


Figure 2.6: Geometry for Bicycle Model Derivation in Reverse Mode

Parameters

- **v_desired** - This is the desired velocity to be followed in reverse mode. It has been tested t perform well in 1.5-2.5 m/s.
- **look_ahead_distance** - This is the lookahead distance based on the desired velocity of the gokart. By observation and runs as explained in Section 4.4.2, it was calculated to be -

$$\text{look\_ahead\_distance} = 2 \times \text{v\_desired - 1} \tag{2.13}$$

- **min_distance** - This is the minimum approach distance reaching which the gokart starts to look for the next lookahead point. By observation and runs, it was calculated to be -

$$min\_distance = look\_ahead\_distance \times 0.8 \tag{2.14}$$

2.6 Properties of the Algorithm

2.6.1 Effect of Lookahead Distance

Longer lookahead distances tend to converge to the path more gradually and with less oscillation. The response of the pure pursuit tracker looks similar to the step response of a second-order dynamic system and the value of L tends to act as a damping factor.

- If the lookahead distance is very large, then the gokart takes a lot of time to converge resulting in corner-cutting as can be seen by the green line in Figure 2.7.
- If the lookahead distance is very small, then the gokart takes very less time to converge resulting in overshooting from the path causing a curvy path as can be seen by the red line in Figure 2.7. The algorithm is calculating a curvature so that the vehicle can drive along an arc. If the path between the vehicle and the goal point is sufficiently curvy then there is no single arc that joins the two points causing failure of the algorithm.
- Therefore it is essential to find the ideal adaptive lookahead distance [7] [9] which is based on the velocity of the vehicle [10] that shows the ideal convergence as can be seen by the blue line in Figure 2.7.

2.7 Limitations

- The controller cannot exactly follow direct paths between the given path points. That's why if the path points given to the controller are not close or if the lookahead distance is very large, then the pure pursuit controller doesn't track the path well [4]. This was considered

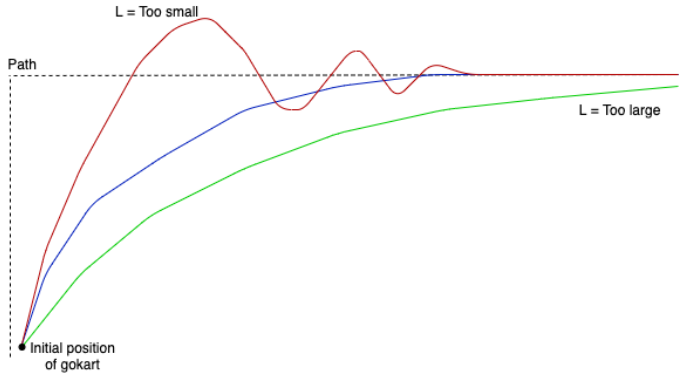


Figure 2.7: Geometry for Bicycle Model Derivation in Reverse Mode

in the implementation and the parameters were tuned to optimize the performance and to converge to the path over time. Also, adaptive lookahead distance was used.

- The pure pursuit algorithm does not stabilize the robot at a point [4]. It is not able to exactly align itself with the goal/lookahead point. Hence, a distance threshold for the lookahead point was applied to make the robot start looking for the next lookahead point once it is near the lookahead point even if not exactly aligned.
- At high speeds, there can be a drastic change in the curvature calculated leading to a drastic change in the steering angle [4]. This can conclude in the vehicle's rear end to skid and unstable shaky movement as shown by several test runs conducted at high speeds in Chapter 4. This was prevented in the final implementation by introducing ramping while changing the steering angle.

Chapter 3

Take-me Home Functionality

3.1 Overview

This functionality is developed to bring the gokart back to the start position from any deadlock or crashed position the gokart may run into during the test runs. This functionality is coded such that it can be integrated with any controller and can produce desired results. Currently, it is integrated with the pure pursuit controller. This functionality can work in two modes -

- **Reverse-Forward mode** - In this implementation, the gokart moves in a reverse mode until it aligns its position and orientation with the centerline points and the tangent to the centerline at the point. Once aligned, it changes its mode from reverse mode to forward mode and then continues to follow the centerline till it reaches the start point.
- **Reverse mode** - In this implementation, the gokart moves in a reverse mode from the crashed or deadlock position until it reaches the start point for the entire path.

Ideally, the mode to be used should be chosen based on the part of the path the gokart has travelled and the remaining part. If gokart has already covered more part than left, it should move in Reverse-Forward mode; else, it should move in Reverse mode. But based on several test runs and observing the results from forward and reverse mode runs as shown in the Chapter 4, it was concluded that forward mode was more reliable, stable and robust to traverse than the reverse mode. Also, higher speed can be achieved in the forward mode. Hence, in the final implementation, the Reverse-Forward mode was adopted.

3.2 Implementation

3.2.1 Terminologies

- **Origin** - This is the start position of the gokart.
- **Deadlock position** - This is the off-track position at which the gokart stops or crashes and has to be sent back to the origin.
- **Realignment point** - This is the point at which the gokart is assumed to have realigned its position and orientation with the centerline. At this point, the gokart's mode changes from reverse mode to forward mode.
- **New path** - This is the path created by storing the gokart's state while it is moving using the test controller. The gokart follows this path in reverse order in reverse mode until it reaches the realignment point or the start point.

- **Centreline path** - This is the path created by extracting the points from the track extractor. This corresponds to the center points of the track, which the gokart tracks while moving to the origin from the realignment point.

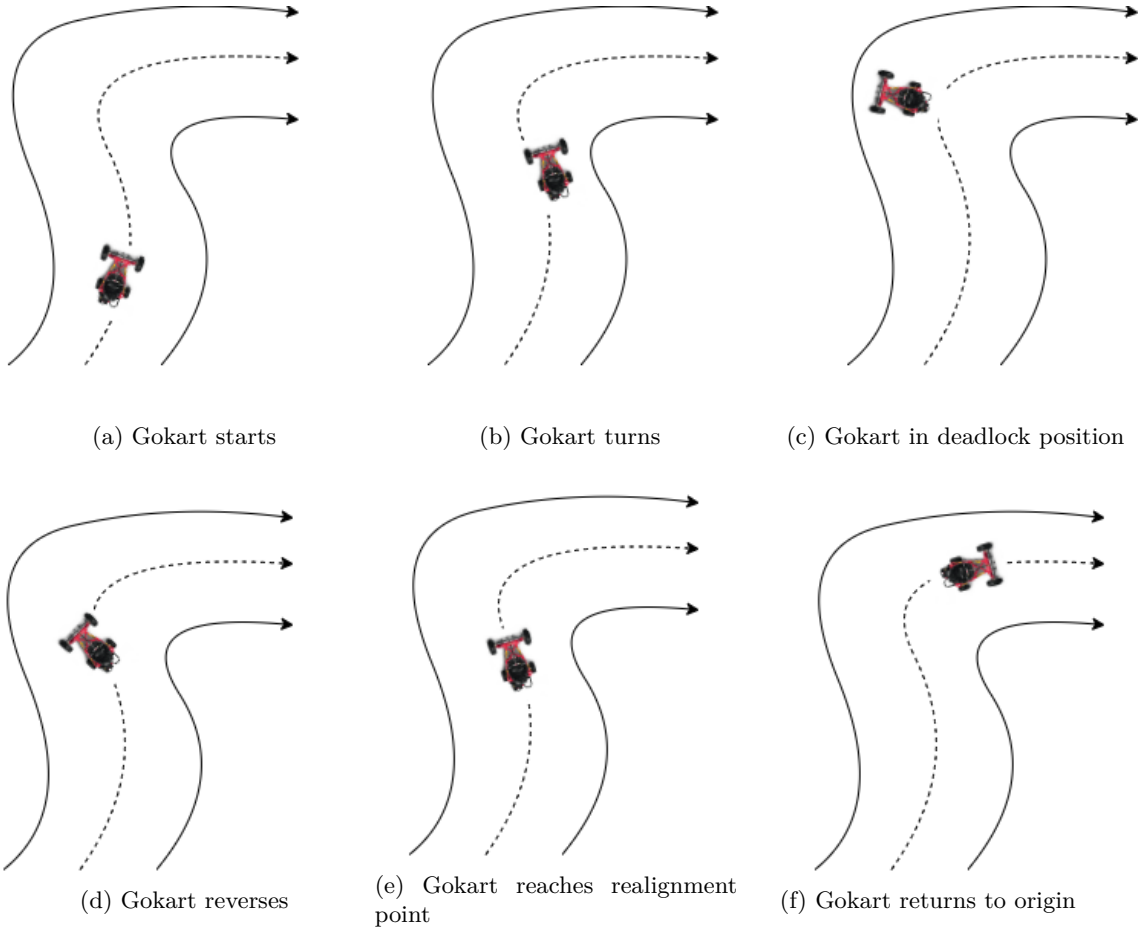
3.2.2 Algorithm

- To run the Reverse-Forward mode, 2 separate paths are needed for the algorithm to work. The first path is the path covered by gokart in reverse mode from deadlock position to the point where it gets realigned. The second path is the centerline that the gokart must follow from the realignment point to reach the origin point. These are the path points that gokart can use to travel in reverse mode till it gets realigned.
- The centerline path from the track extractor is already extracted. But there is still no path that the gokart can track while reversing to reach the realignment point. So the pure pursuit controller, when running any testing controller, keeps on storing the gokart's state until the gokart reaches a deadlock situation.
- So now, once gokart is in a deadlock situation, the controller can be started. It first moves on the reverse path on the stored path till it realigns itself. Once realigned, it starts moving in the forward mode tracking the centerline to reach the origin. The algorithm can be visualised in Figure 3.1.

Algorithm 2: Take-me Home Functionality Algorithm

Input: Centerline Path points P, Gokart's position X_{cur}, Gokart's velocity V_{cur}, Gokart's Length L_{car}
Output: Steering Angle δ

```
initialization;
while P, X_cur and V_cur is available do
    if !start_controller then
        P_record.append(X_cur);
    else
        if !realigned then
            C_p ← find_near_point(P_record, X_cur);
            if C_p = L_p then
                L ← get_lookahead_distance(V_cur);
                L_p ← find_goal_point(P_record, X_cur, L);
            end
            γ ← get_curvature(L_p, X_cur);
            δ ← compute_steering_angle(γ, L_car);
            realigned ← is_aligned(X_cur, P)
        else
            C_p ← find_near_point(P, X_cur);
            if C_p = L_p then
                L ← get_lookahead_distance(V_cur);
                L_p ← find_goal_point(P, X_cur, L);
            end
            γ ← get_curvature(L_p, X_cur);
            δ ← compute_steering_angle(γ, L_car);
            if X_cur = origin then
                stop;
            end
        end
    end
end
```



(a) Gokart starts (b) Gokart turns (c) Gokart in deadlock position

(d) Gokart reverses (e) Gokart reaches realignment point (f) Gokart returns to origin

Figure 3.1: Take-me Home functionality algorithm

3.2.3 Functions

- **is_aligned** - This function is called during the gokart's reverse mode till it reaches the realignment point. This function calculates the geodesic distance [11] between the gokart's current state and the centerline path points. Based on the geodesic distance observed, it can be realised if the gokart has realigned itself or not. From observations, if the geodesic distance is less than 0.8, then it was considered that the gokart has realigned itself, and it returns true. Then it also switches the gokart's mode from reverse to forward.

$$dist = \{(gk\_state.x - path\_point.x)^2 + (gk\_state.y - path\_point.y)^2 + (gk\_state.theta - path\_point.theta)^2\}^{1/2} \tag{3.1}$$

- **update_path** - This function is the callback function of the rostopic "/kitt/path/path_array" that stores the path points to be followed. In this functionality, it always stores the centerline path that the gokart has to follow to reach the origin from the realignment point.
- **update_location** - This function is the callback function of the rostopic "/kitt/se2_localization /gokart_state" that updates the current location of the gokart. In this functionality, this function also keeps on storing the gokart's state in an array to be used by the gokart to reach to realignment point from the deadlock position.
- **find_nearest_point_index_on_path** - This function returns the Centre Path Point on the path based on the gokart's current position. In this functionality, it returns the Centre

Path Point on the new path recorded while testing the test controller during the reverse mode. In the forward mode, it returns the Centre Path Point on the centerline path that the gokart uses to reach to origin.

- **find_goal_point** - This function returns the Lookahead Point on the path based on the gokart's current position and the lookahead distance. In this functionality, it returns the Lookahead Point on the new path recorded while testing the test controller during the reverse mode. In the forward mode, it returns the Lookahead Point on the centerline path that the gokart uses to reach to origin.
- **update_parameters** - This function updates the pure pursuit controller parameters - *"desired velocity, minimum approach distance, lookahead distance"* for the gokart based on the mode in which gokart has to move in.

3.2.4 Parameters

- In forward mode, it uses the parameters used in pure pursuit controller's forward mode as explained in Section 2.5.2.
- In reverse mode, it uses the parameters used in pure pursuit controller's reverse mode as explained in Section 2.5.3.

Chapter 4

Results

4.1 Hyper-parameters Tuned

This section reports all the hyper-parameter of the pure pursuit controller which are varied and tuned to increase the robustness of the algorithm.

- **Speed**: Speed was varied to find the limits in which the controller can work robustly in different modes. It is observed that at very low speed, the gokart is not able to overcome the friction and threshold power needed to move. At moderate speeds, the gokart is able to track the paths well. At very high speeds, the gokart generally overshoots while aligning itself, skids and is not able to track the path well.
- **Lookahead Distance**: Lookahead distance was varied at different speeds to find the ideal lookahead distance at each speed at which the controller can work robustly. As explained in the Section 2.6.1, ideal lookahead distances are desired to have ideal convergence.
- **Weight**: Weight was varied to test the stability of the controller and other hyper-parameters. Higher weights provide more friction to the car causing higher restoring torque while steering and moving. Thus steering controller was tuned to compensate for this and making the controller robust in the driver's weight range 50 kg - 90 kg.

4.2 Metrics

Several test runs were conducted to find the speed limits within which the controller works robustly in different weight ranges of the rider. Also, several runs were conducted to find the ideal lookahead distance at different speeds. And once all the hyper-parameters of the pure pursuit controller were tuned, the performance of the controller was compared with the MPC controller. To quantify the results of the test runs to compare different hyper-parameters and controllers, the following metrics [12] were defined -

4.2.1 Quantitative Analysis

- **Lateral Error**: It is the lateral euclidean distance between gokart's position and the Center Path Point.
- **Lateral Mean - Error**: This is measured as the average of all the lateral errors during the test runs. This tells us that during a test run, how much the gokart deviates from the centerline.
  - A lower value of Mean - error signifies that the gokart deviates less from the centerline and thus the hyper-parameters and the controller performs better.

- **Lateral Standard Deviation - Error**: This is measured as the standard deviation of all the lateral errors during the test runs. This tells us the spread of the data. Thus this tells us that during a test run, how much volatile is the gokart's deviation from the centerline.
  - A lower value of Standard Deviation - error signifies that the gokart's deviation from the centerline is more consistent and less volatile and thus the hyper-parameters and the controller performs better concluding in a smoother ride.

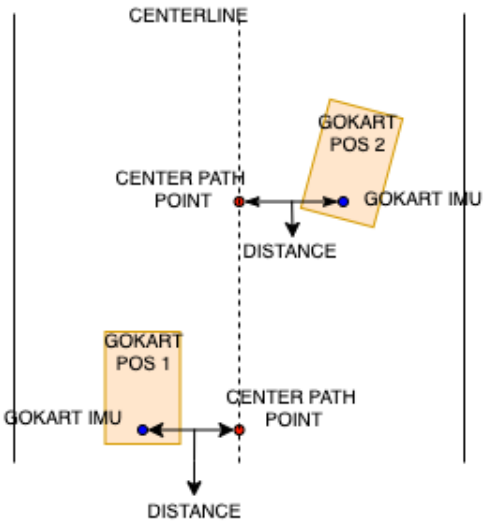


Figure 4.1: Plot showing distance used for calculating metrics

4.2.2 Qualitative Analysis

- **Graph Analysis**: Plots of observed steering angles and given reference steering angles at different instances are compared to qualitatively understand the stability and the smoothness of the movement in the test runs.
  - If the plots overlap well, then it can be concluded that the gokart's steering wheel is able to achieve the desired reference angles which result in better path tracking. Thus higher overlap is desired.
  - If the frequency of drastic changes is less in the values of observed steering angles, then it can be concluded that the steering wheel doesn't overshoot much resulting in smooth, less shaky and more stable movement. Thus less drastic changes in observed steering angles are desired.

Comparing the two quantitative metrics for any hyper-parameter and controller provides the quantitative evaluation of the performance. But they don't take into account the shaking of the steering wheel and the jerky movement of the gokart which can only be evaluated qualitatively using the graphs and experiences of the rider. So the final comparative results of the parameters and controllers in different modes are reported by taking into account both the quantitative and qualitative metrics and the rider's comfort during the test runs.

4.3 Forward Mode

This section shows the metric values for several test runs with different hyper-parameters in Forward mode.

4.3.1 Varying Lookahead Distance and Weights at different at Speeds

Speed: 2 m/s

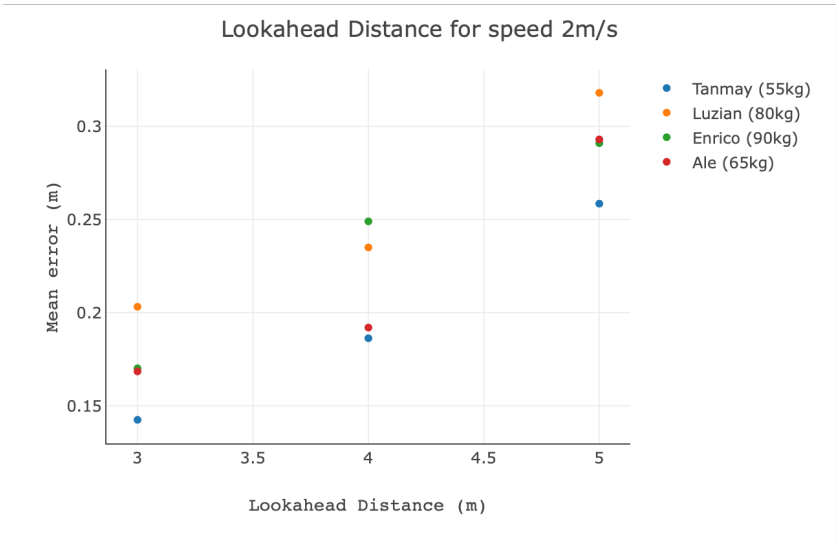


Figure 4.2: Plots showing mean-error for different lookahead distances at 2 m/s

Speed (m/s)	Lookahead Distance (m)	Weight (kg)	Mean - error (m)	Standard Deviation - Error (m)
		Tanmay (55 kg)	0.142	0.0804
	3	Ale (65 kg)	0.168	0.105
		Luzian (80 kg)	0.203	0.875
		Enrico (90 kg)	0.170	0.102
		Tanmay (55 kg)	0.186	0.124
2	**4**	Ale (65 kg)	0.192	0.186
		Luzian (80 kg)	0.235	0.427
		Enrico (90 kg)	0.249	0.163
		Tanmay (55 kg)	0.258	0.195
	5	Ale (65 kg)	0.293	0.206
		Luzian (80 kg)	0.318	0.852
		Enrico (90 kg)	0.291	0.105

Table 4.1: Table showing metric values for test runs at 2 m/s

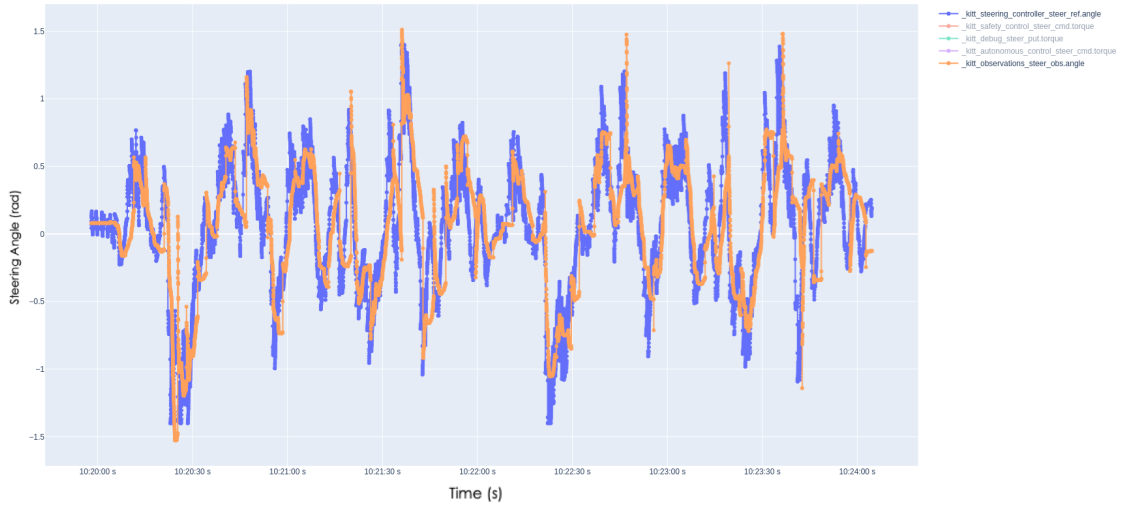


Figure 4.3: Plots showing steering observation and command angles in forward mode at speed 2.0 m/s and lookahead distance 3 m

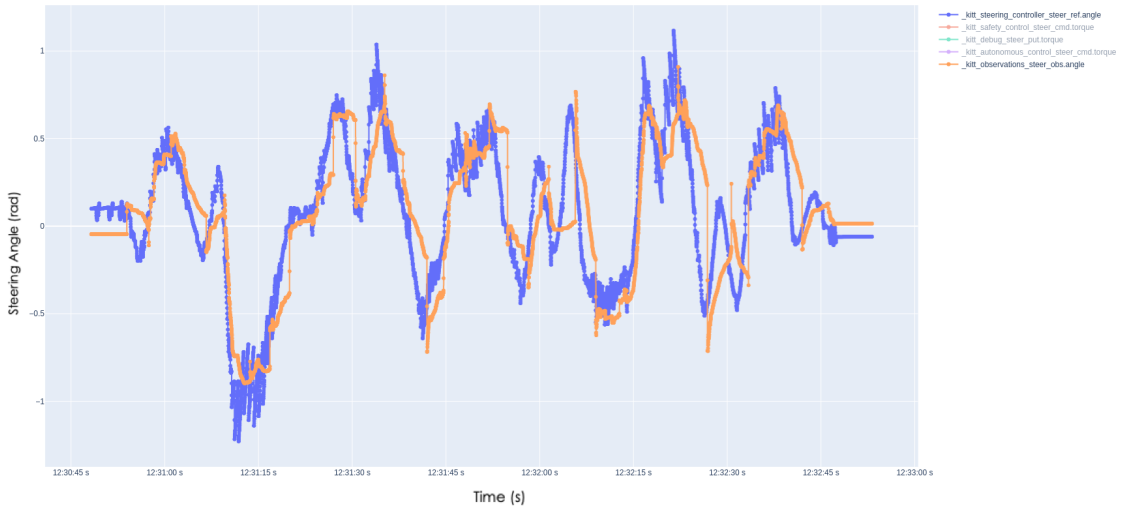


Figure 4.4: Plots showing steering observation and command angles in forward mode at speed 2.0 m/s and lookahead distance 4 m

Table 4.1 and Figure 4.2 show the comparison of various runs with varying Lookahead distance and Weights on the gokart at speed 2 m/s. Figure 4.3, Figure 4.4 and Figure 4.5 shows graphs depicting the reference steering angle predicted by the controller which the gokart has to achieve at different time instances during the run and the gokart's steering at that time instance. So a comparison made by plotting the 2 values on graph can tell us about how well the steering controller is tracking the desired steering angles. It was observed that when the lookahead distance is set at 3 m, the mean - error is minimum compared to higher lookahead distances. Also at 2 m/s, the movement of gokart is less bumpy and jerky as felt by the rider and as observed by looking at the steering value graphs as in Figure 4.3 it can be seen that the plots overlap well without sudden changes in

the observed angle values. So it was concluded that the ideal lookahead distance for the speed 2 m/s is 3 m.

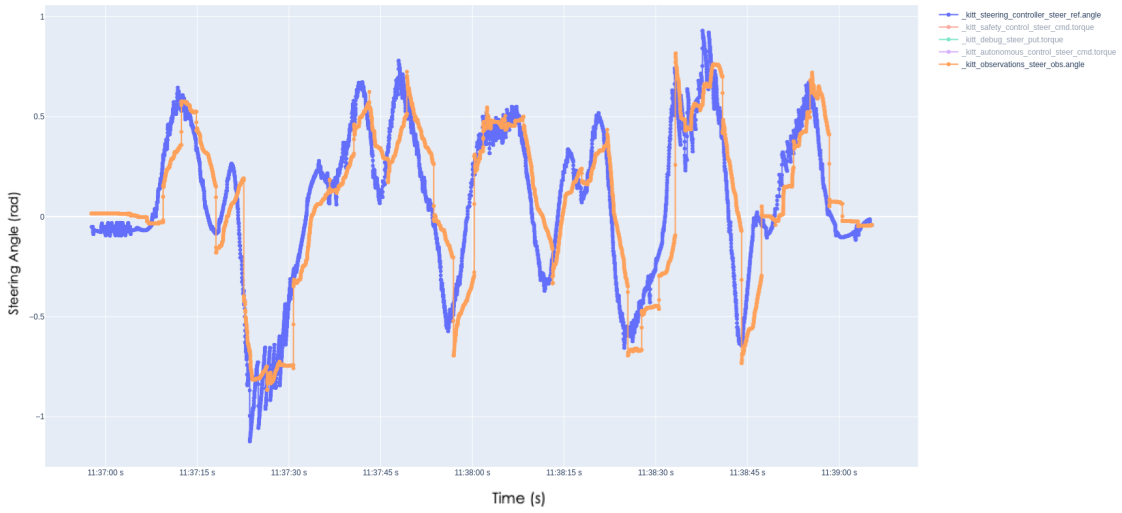


Figure 4.5: Plots showing steering observation and command angles in forward mode at speed 2.0 m/s and lookahead distance 5 m

Speed: 3 m/s

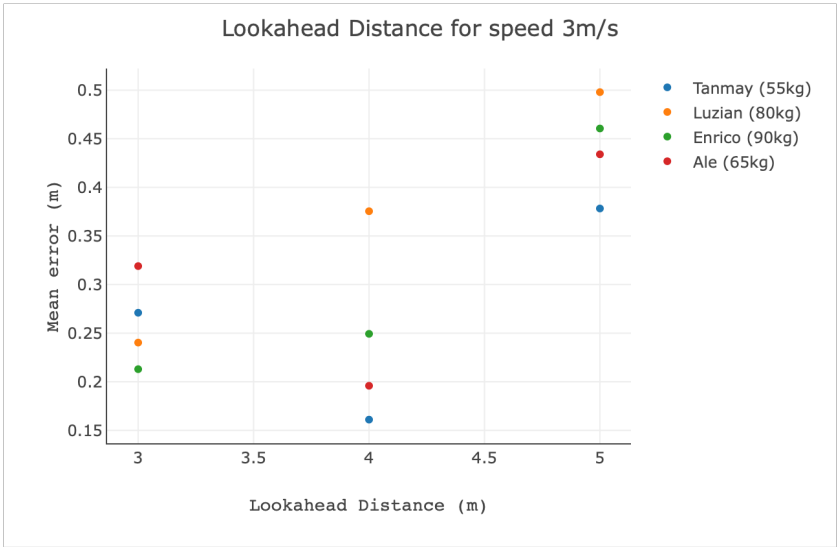


Figure 4.6: Plots showing mean-error for different lookahead distances at 3 m/s

Speed (m/s)	Lookahead Distance (m)	Weight (kg)	Mean - error (m)	Standard Deviation - Error (m)
		Tanmay (55 kg)	0.271	0.213
	3	Ale (65 kg)	0.319	1.265
		Luzian (80 kg)	0.240	1.485
		Enrico (90 kg)	0.212	0.166
		Tanmay (55 kg)	0.161	0.124
3	**4**	Ale (65 kg)	0.195	0.156
		Luzian (80 kg)	0.375	1.061
		Enrico (90 kg)	0.249	0.161
		Tanmay (55 kg)	0.378	0.280
	5	Ale (65 kg)	0.434	0.303
		Luzian (80 kg)	0.497	0.355
		Enrico (90 kg)	0.460	0.335

Table 4.2: Table showing metric values for test runs at 3 m/s

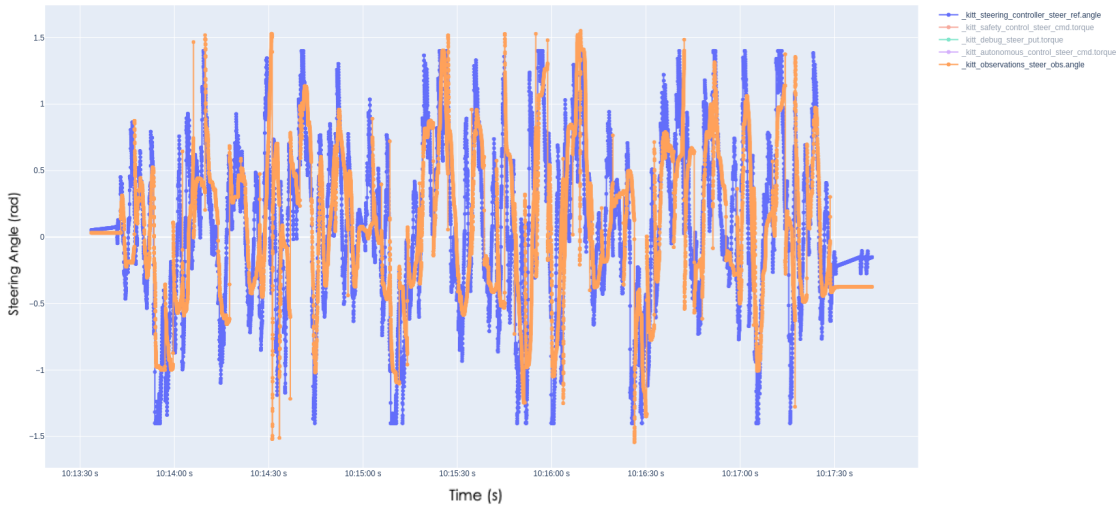


Figure 4.7: Plots showing steering observation and command angles in forward mode at speed 3.0 m/s and lookahead distance 3 m

Table 4.2 and Figure 4.6 show the comparison of various runs with varying the Lookahead distance and Weights on the gokart at speed 3 m/s. Figure 4.7, Figure 4.8 and Figure 4.9 show graphs depicting the reference steering angle predicted by the controller which the gokart has to achieve at different time instances during the run and the gokart's steering at that time instance. It was

observed that when the lookahead distance is set at 4 m, the mean - error is minimum compared to higher and lower lookahead distances for lower weights and also the movement was smooth. For higher weights, it was observed that the mean - error was a little more for lookahead distance 4 m. But looking at the graphs of steering angles at different lookahead distances, it can be seen in Figure 4.7 corresponding to lookahead distance 3m that the observed angles change a lot on most time instances concluding a shaky and bumpy run. But for 4m lookahead distance, the plots are smoother with less drastic change in values. So it was concluded that the ideal lookahead distance for the speed 3 m/s is 4 m.

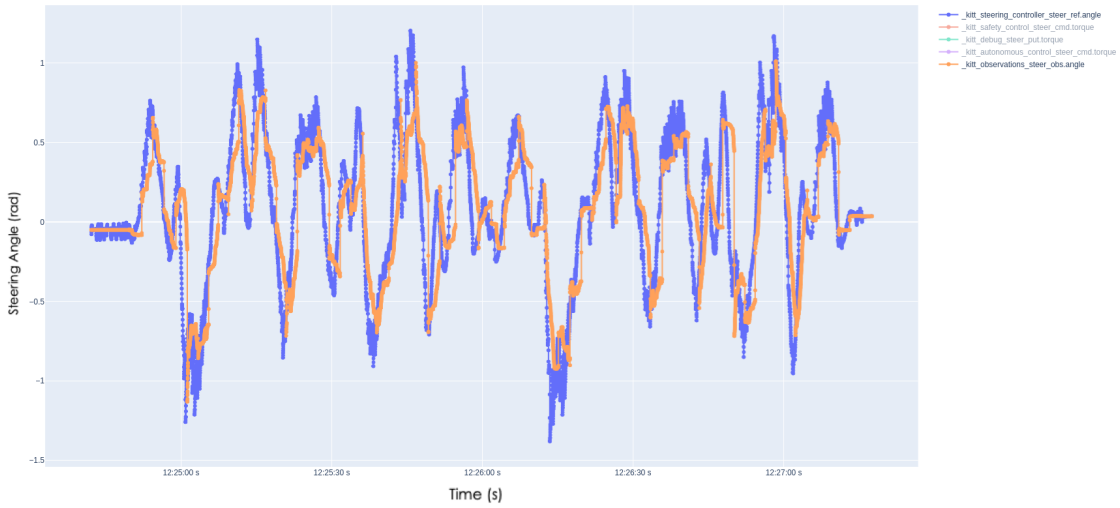


Figure 4.8: Plots showing steering observation and command angles in forward mode at speed 3.0 m/s and lookahead distance 4 m

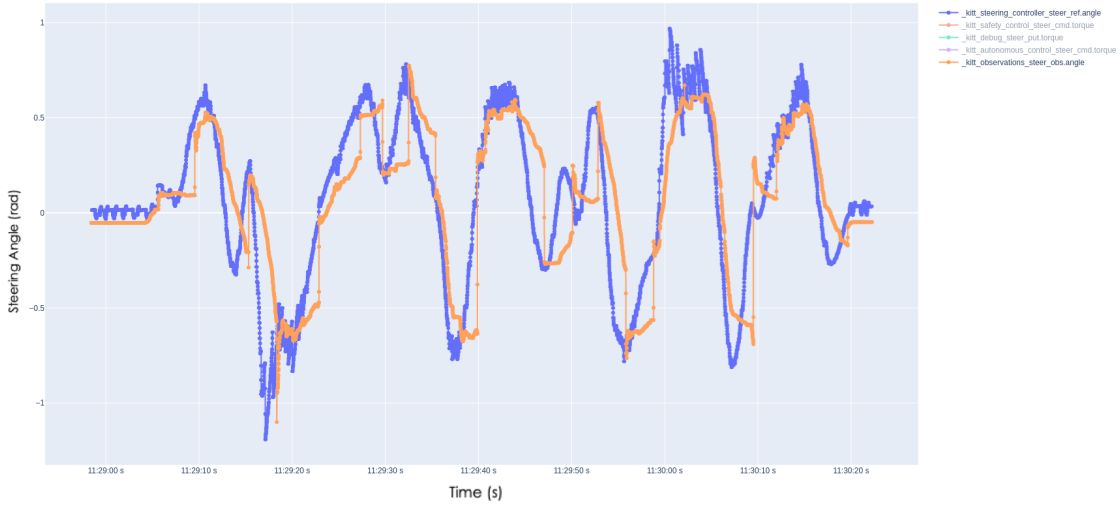


Figure 4.9: Plots showing steering observation and command angles in forward mode at speed 3.0 m/s and lookahead distance 5 m

Speed: 4 m/s

Figure 4.10: Plots showing mean-error for different lookahead distances at 4 m/s

Speed (m/s)	Lookahead Distance (m)	Weight (kg)	Mean - error (m)	Standard Deviation - Error (m)
		Tanmay (55 kg)	0.343	0.293
	3	Ale (65 kg)	0.298	0.386
		Luzian (80 kg)	0.483	1.25
		Enrico (90 kg)	0.461	0.423
		Tanmay (55 kg)	0.294	0.231
4	**4**	Ale (65 kg)	0.331	0.258
		Luzian (80 kg)	0.416	1.324
		Enrico (90 kg)	0.366	0.272
		Tanmay (55 kg)	0.601	0.401
	5	Ale (65 kg)	0.516	0.355
		Luzian (80 kg)	0.727	0.532
		Enrico (90 kg)	0.598	0.399

Table 4.3: Table showing metric values for test runs at 4 m/s

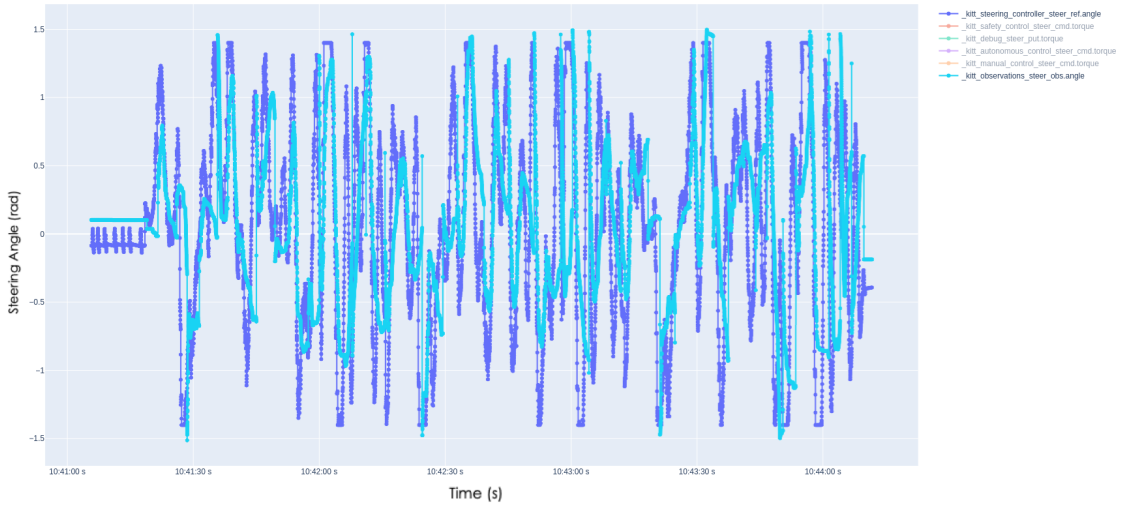


Figure 4.11: Plots showing steering observation and command angles in forward mode at speed 4.0 m/s and lookahead distance 3 m

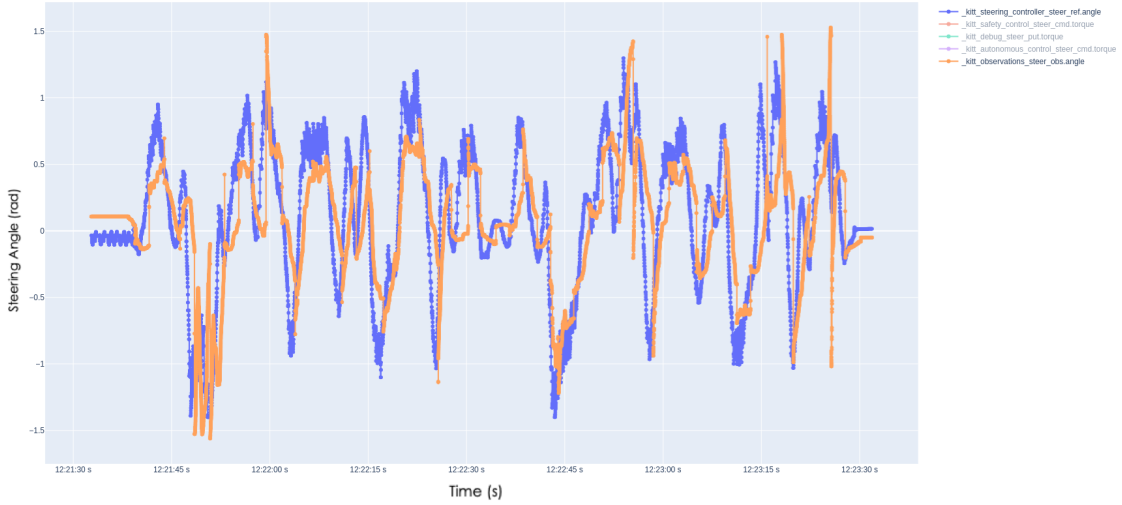


Figure 4.12: Plots showing steering observation and command angles in forward mode at speed 4.0 m/s and lookahead distance 4 m

Table 4.3 and Figure 4.10 show the comparison of various runs with varying the Lookahead distance and Weights on the gokart at speed 4 m/s. Figure 4.11, Figure 4.12 and Figure 4.13 show graphs depicting the reference steering angle predicted by the controller which the gokart has to achieve at different time instances during the run and the gokart's steering at that time instance. It was observed that when the lookahead distance is set 5 m, the mean - error is maximum compared to the lower lookahead distances but the movement of gokart was smoothest. For smaller lookahead distances, it was observed that although the mean - error was less, the movement was a lot more jerky and unstable. By seeing Figure 4.11 and Figure 4.12, it can be concluded that for lookahead distance of 3m and 4m, the observed steering angles changes drastically on most time instances

which concluded a shaky and bumpy ride. But in Figure 4.13 of lookahead distance 5m, the observed angles don't change drastically as much concluding a smoother ride. So it was concluded that the ideal lookahead distance for the speed 4 m/s is 5 m because at this lookahead distance the movement was much smoother and stable and at the same time the mean-error was only a little more.

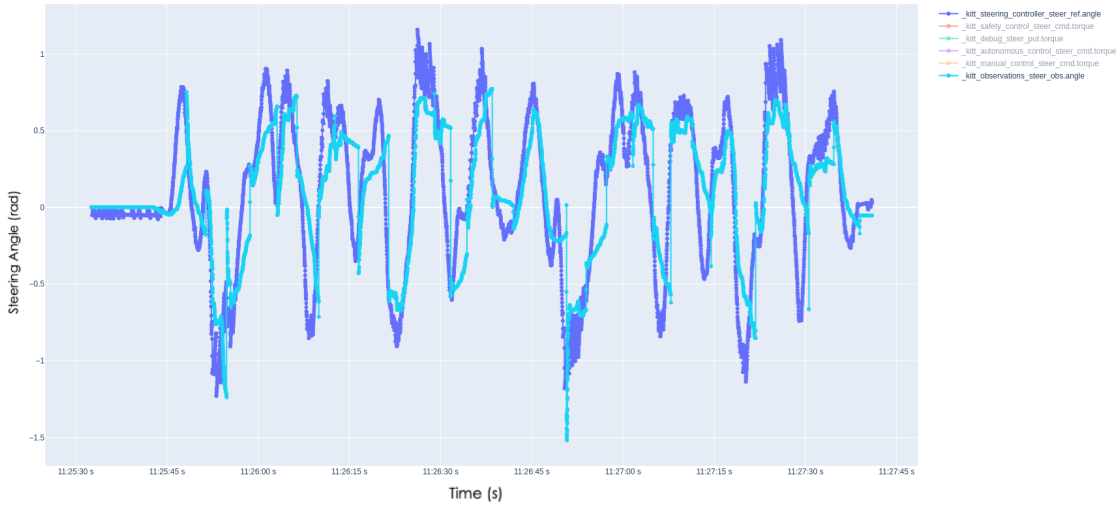


Figure 4.13: Plots showing steering observation and command angles in forward mode at speed 4.0 m/s and lookahead distance 5 m

4.3.2 Deducing Lookahead distance - Speed relation

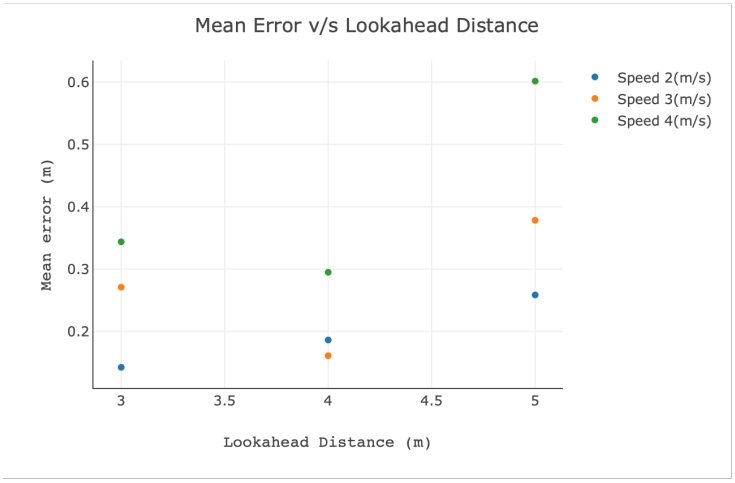


Figure 4.14: Plots showing mean-error for different lookahead distances at different speeds

Figure 4.14 shows the mean - error of various runs at different lookahead distances and speeds. At 2 m/s, it can be seen that the mean - error is minimum for lookahead distance 3 m. Also, the movement was stable and less jerky as can be seen from the results of Section 4.3.1, making it an

ideal value. At 3 m/s, it can be seen that the mean - error is minimum for lookahead distance 4 m. Also, the movement was stable and less jerky as can be seen from the results of Section 4.3.1, making it an ideal value. At 4 m/s, it can be seen that the mean - error is minimum for lookahead distance 4 m. But the movement is very jerky and unstable as can be seen from the results of Section 4.3.1. Hence, the ideal value of lookahead distance for 4 m/s is 5 m.

S.No.	Speed (m/s)	Lookahead Distance (m)
1	2.0	3
2	3.0	4
3	4.0	5

Table 4.4: Table showing ideal lookahead distances for different speeds in forward mode

Therefore by performing linear regression on the 3 sets of values as shown in Table 4.4, the equation can be deduced as -

$$lookahead\_distance = v\_desired + 1. \tag{4.1}$$

4.3.3 MPC and Pure Pursuit Controller comparison

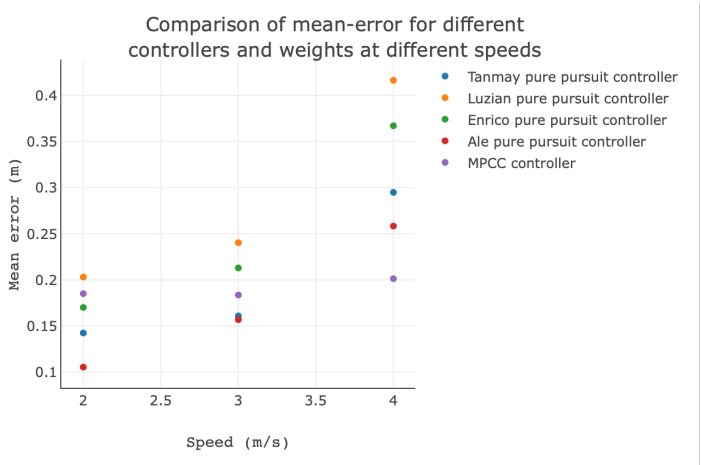


Figure 4.15: Plots showing a comparison of mean-error for different controllers and weights at different speeds

The Figure 4.15 depicts the mean - error of the test runs at different speeds for the pure pursuit controller at different weights and MPC controller. By observing the graph, we can see that for lower speeds, 2 m/s and 3 m/s, the mean error was at par for the pure pursuit controller in comparison to the MPC controller [1]. Thus pure pursuit controller is able to track the path at par with the MPC controller for lower speeds. But as we increase the speed to 4 m/s, it can be seen that the mean error for the pure pursuit controller increases drastically but for the MPC controller the mean error doesn't increase much. The mean error for the pure pursuit controller increase because at higher speeds, the gokart tends to move on a larger radius of curvature while turning according to the algorithm. This causes the gokart to take some time in retracing the

centerline while turning, causing larger deviations from the centerline. Hence it can be concluded that for smaller speeds, pure pursuit controller tracks path at par with MPC controller, but for larger speeds, Pure Pursuit controller performance reduces significantly when compared with MPC controller.

4.4 Reverse Mode

This section shows the metric values for several test runs with different hyper-parameters in Reverse mode.

4.4.1 Varying Lookahead Distance at different at Speeds and Weights

Speed: 1.5 m/s

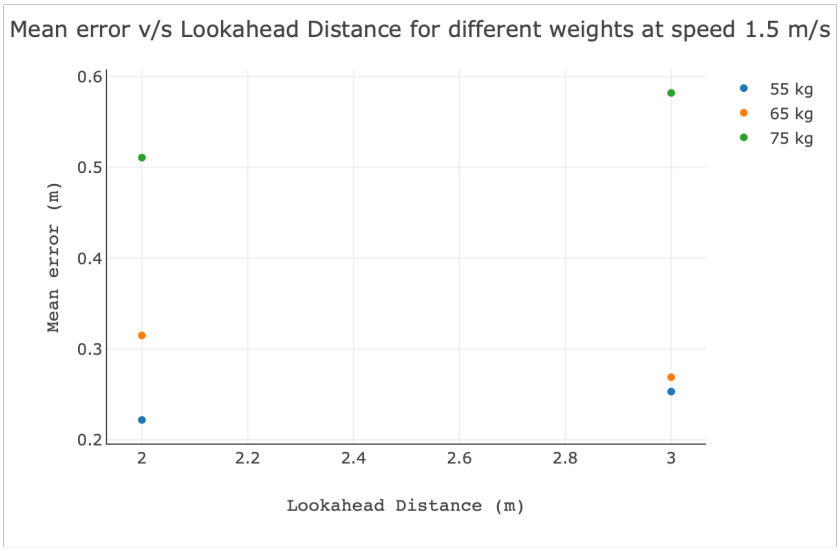


Figure 4.16: Plots showing mean-error for different lookahead distances at 1.5 m/s

Speed (m/s)	Lookahead Distance (m)	Weight (kg)	Mean - error (m)	Standard Deviation - Error (m)
1.5	**2**	55 kg	0.221	0.171
		65 kg	0.314	1.166
		75 kg	0.510	1.371
	3	55 kg	0.253	0.192
		65 kg	0.268	0.148
		75 kg	0.582	1.364

Table 4.5: Table showing metric values for test runs at 1.5 m/s

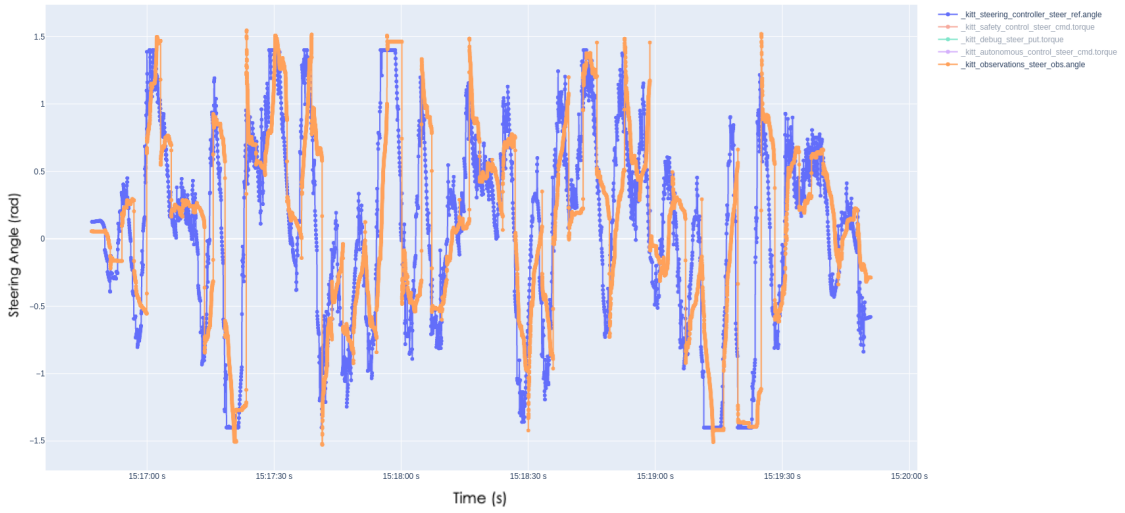


Figure 4.17: Plots showing steering observation and command angles in reverse mode at speed 1.5 m/s and lookahead distance 2 m

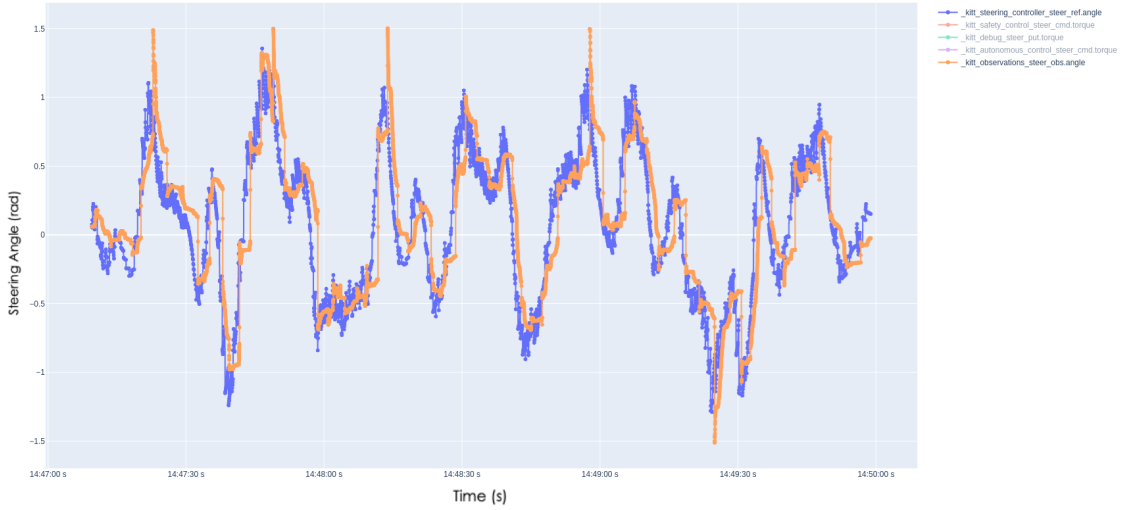


Figure 4.18: Plots showing steering observation and command angles in reverse mode at speed 1.5 m/s and lookahead distance 3 m

Table 4.5 and Figure 4.16 show the comparison of various runs with varying the Lookahead distances and Weights on the gokart at speed 1.5 m/s in reverse mode. It was observed it follows the path most accurately. Figure 4.17 and Figure 4.18 show graphs depicting the reference steering angle predicted by the controller which the gokart has to achieve at different time instances during the run and the gokart's steering angle at that time instance. By seeing the graphs, it can be concluded that for lookahead distance 2 m, the plots overlap well and also the observed angles don't change drastically. So it was concluded that the ideal lookahead distance for the speed 1.5 m/s is 2 m as it showed accurate tracking with minimal jerks and bumps.

Speed: 2 m/s

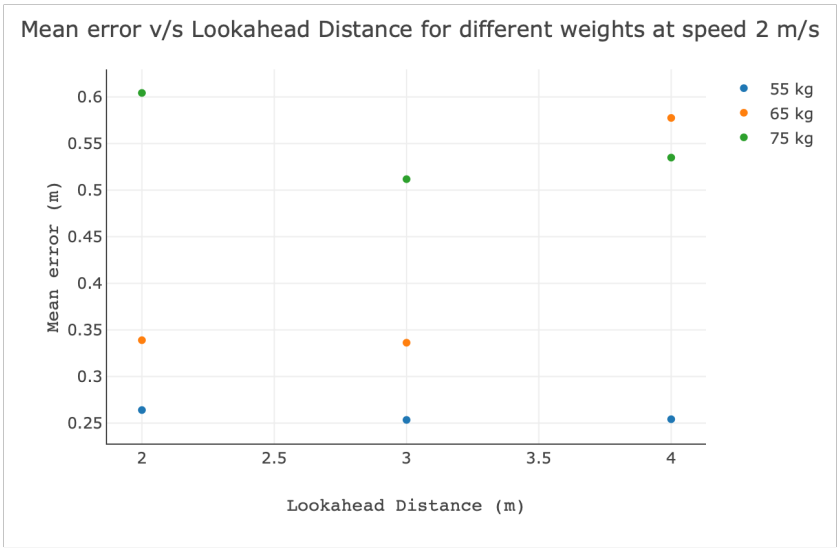


Figure 4.19: Plots showing mean-error for different lookahead distances at 2 m/s

Speed (m/s)	Lookahead Distance (m)	Weight (kg)	Mean - error (m)	Standard Deviation - Error (m)
		55 kg	0.264	0.529
	2	65 kg	0.338	0.259
		75 kg	0.604	2.348
		55 kg	0.253	0.678
2	**3**	65 kg	0.336	1.641
		75 kg	0.511	1.641
		55 kg	0.254	0.845
	4	65 kg	0.577	1.637
		75 kg	0.534	0.343

Table 4.6: Table showing metric values for test runs at 2 m/s

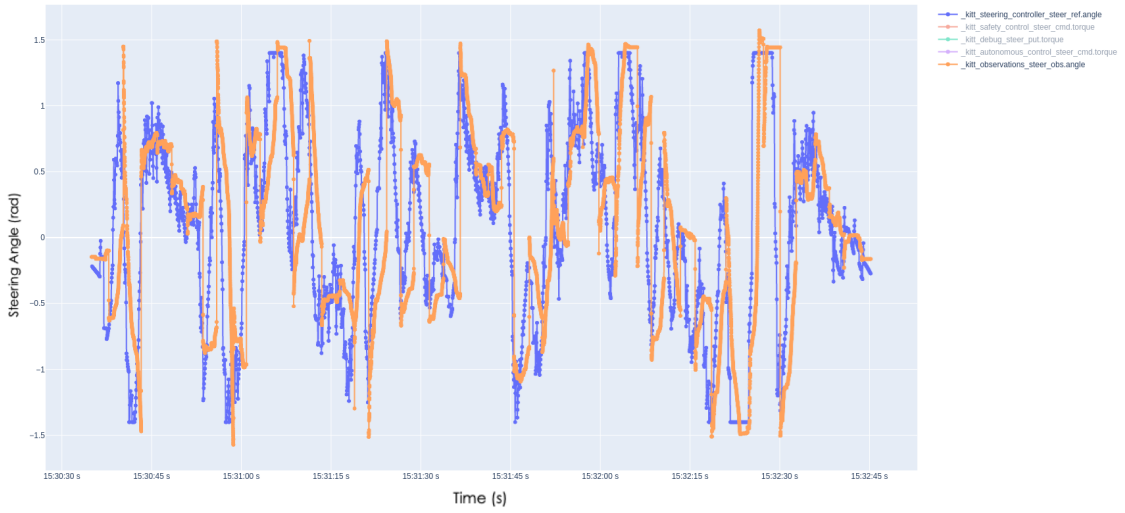


Figure 4.20: Plots showing steering observation and command angles in reverse mode at speed 2.0 m/s and lookahead distance 2 m

Figure 4.21: Plots showing steering observation and command angles in reverse mode at speed 2.0 m/s and lookahead distance 3 m

Table 4.6 and Figure 4.19 show the comparison of various runs with varying the Lookahead distance and Weights on the gokart at speed 2 m/s. It was observed that when the lookahead distance is set at 3 m, the mean - error is more compared to the other lookahead distances for lower weights. But at higher weights, it was minimum. Figure 4.20, Figure 4.21 and Figure 4.22 show graphs depicting the reference steering angle predicted by the controller which the gokart has to achieve at different time instances during the run and the gokart's steering angle at that time instance. For lookahead distance 2 m, it can be seen that the observed angles changed drastically on most time instances resulting in a bumpy and jerky ride. But at a lookahead distance of 3 m, it was

observed that the plots overlap well and also the observed steering angles don't change drastically concluding that the movement of gokart was smoothest and most stable. So it was concluded that the ideal lookahead distance for the speed 2 m/s is 3 m because at this lookahead distance the movement was stable and at the same time the mean - error was also optimal.

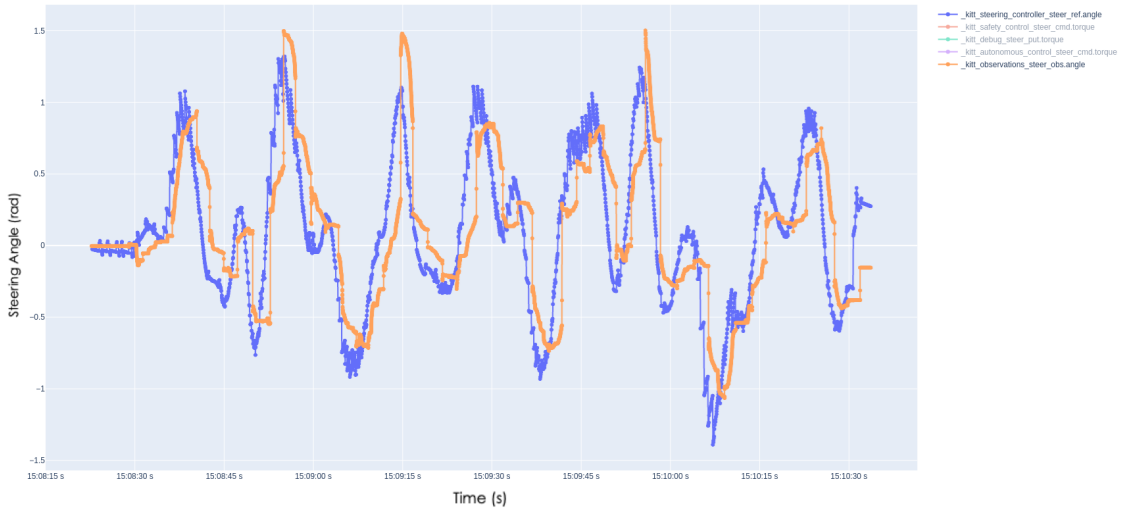


Figure 4.22: Plots showing steering observation and command angles in reverse mode at speed 2.0 m/s and lookahead distance 4 m

Speed: 2.5 m/s

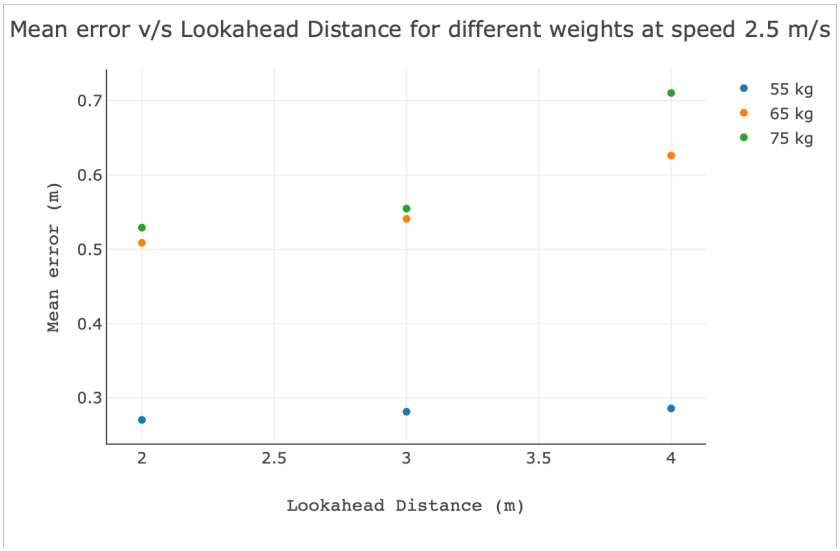


Figure 4.23: Plots showing mean-error for different lookahead distances at 2.5 m/s

Speed (m/s)	Lookahead Distance (m)	Weight (kg)	Mean - error (m)	Standard Deviation - Error (m)
		55 kg	0.270	0.242
	2	65 kg	0.508	1.301
		75 kg	0.529	1.163
		55 kg	0.281	0.226
2.5	**3**	65 kg	0.540	1.375
		75 kg	0.554	1.657
		55 kg	0.285	0.188
	4	65 kg	0.626	1.683
		75 kg	0.710	2.517

Table 4.7: Table showing metric values for test runs at 2.5 m/s

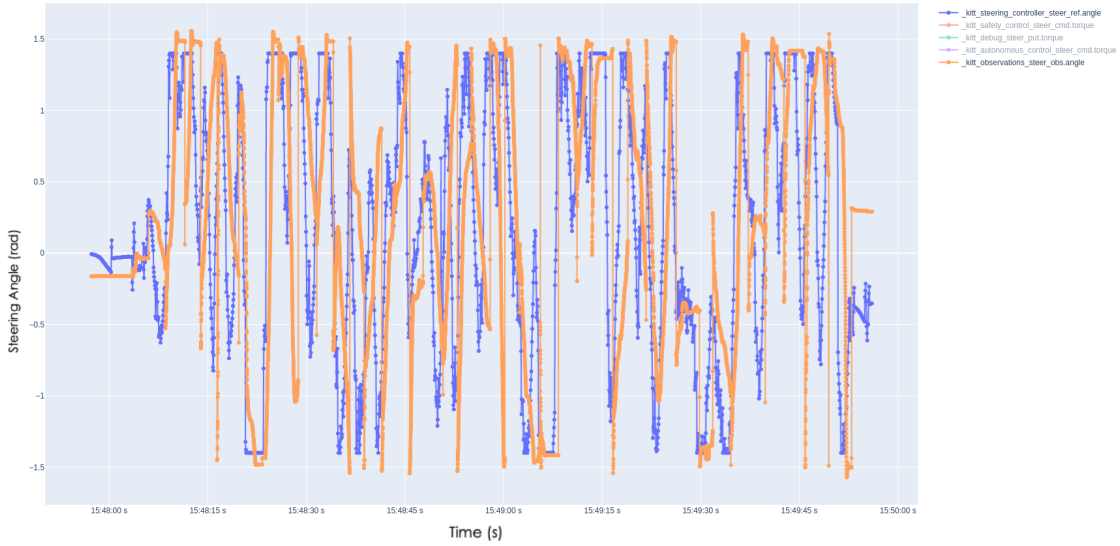


Figure 4.24: Plots showing steering observation and command angles in reverse mode at speed 2.5 m/s and lookahead distance 2 m

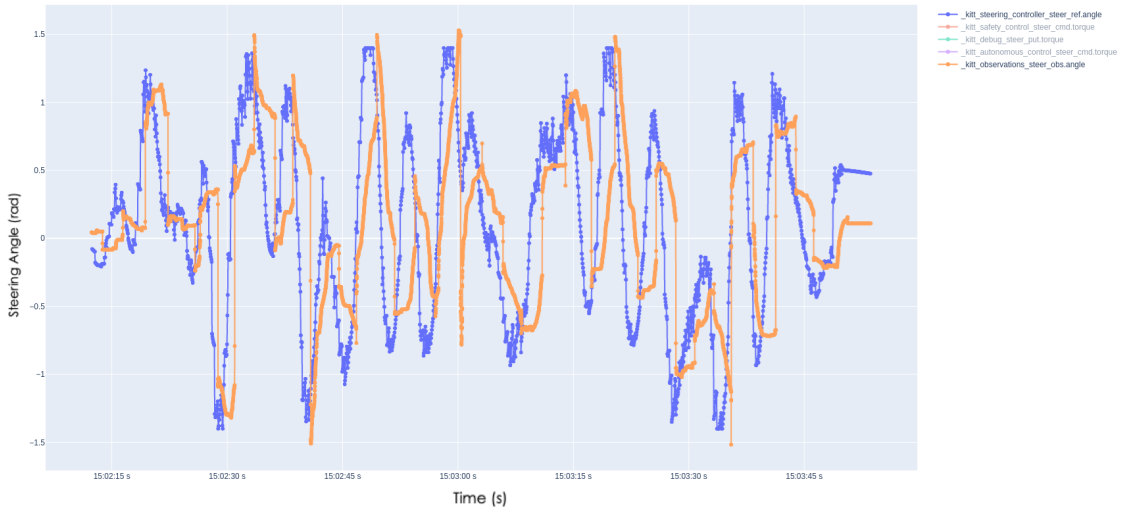


Figure 4.25: Plots showing steering observation and command angles in reverse mode at speed 2.5 m/s and lookahead distance 3 m

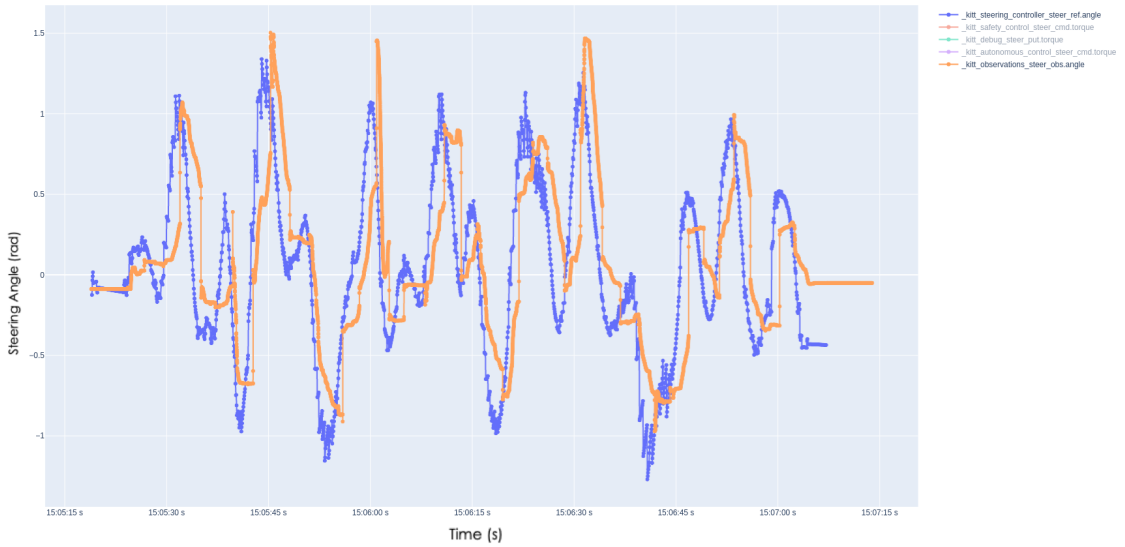


Figure 4.26: Plots showing steering observation and command angles in reverse mode at speed 2.5 m/s and lookahead distance 4 m

Table 4.7 and Figure 4.23 show the comparison of various runs with varying the Lookahead distance and Weights on the gokart at speed 2.5 m/s. It was observed that when the lookahead distance is set at 4 m, the mean - error is slightly more compared to the other lookahead distances. Figure 4.24, Figure 4.25 and Figure 4.26 show graphs depicting the reference steering angle predicted by the controller which the gokart has to achieve at different time instances during the run and the gokart's steering angle at that time instance. It can be seen that for lookahead distance 2 m and 3 m, the plots don't overlap much. Also, the observed steering angles changed drastically on most

time instances. This meant that there are a lot of jerks during the runs resulting in instability. But for lookahead distance 4 m, the plots overlap well with minimal drastic changes in observed steering angles concluding that the movement of gokart was a lot smoother and most stable. So it was concluded that the ideal lookahead distance for the speed 2.5 m/s is 4 m because at this lookahead distance the movement was most stable and at the same time the mean - error was also only a little more causing optimal tracking.

4.4.2 Deducing Lookahead distance - Speed relation

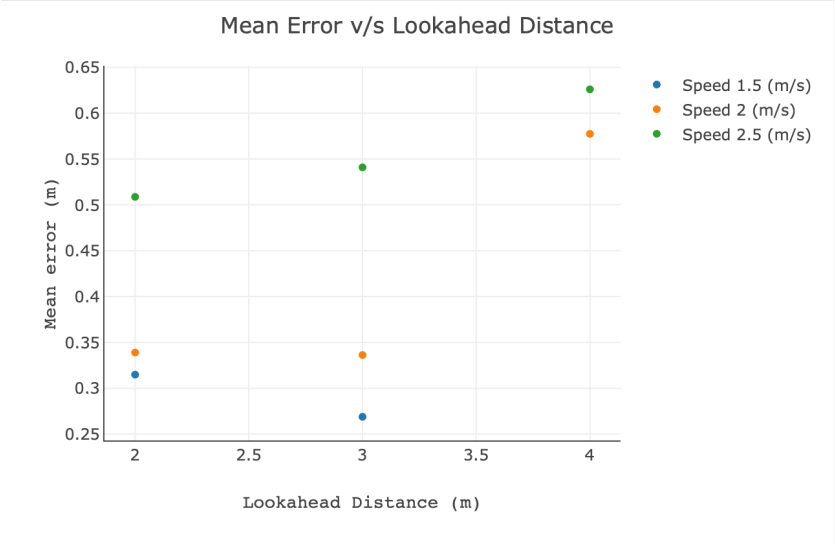


Figure 4.27: Plots showing a comparison of mean-error for different lookahead distances at different speeds

Figure 4.27 show the mean - error of various runs at different lookahead distances and speeds. At 1.5 m/s, it can be seen that the mean - error is less for lookahead distance 2 m. Also, the movement was stable and less jerky as shown by the results of Section 4.4.1, making it an ideal value. At 2 m/s, it can be seen that the mean - error is minimum for lookahead distance 3 m. Also, the movement was stable and less jerky as shown by the results of Section 4.4.1, making it an ideal value. At 2.5 m/s, it can be seen that the mean - error is minimum for lookahead distance 2 m. But the movement is very jerky and unstable as shown by the results of Section 4.4.1. Hence, the ideal value of lookahead distance for 2.5 m/s is 4 m.

S.No.	Speed (m/s)	Lookahead Distance (m)
1	1.5	2
2	2.0	3
3	2.5	4

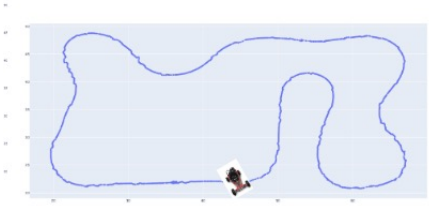
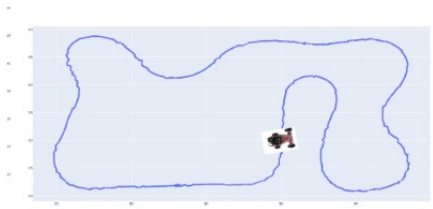
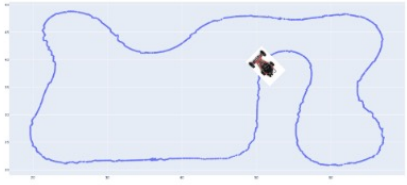
Table 4.8: Table showing ideal lookahead distances for different speeds

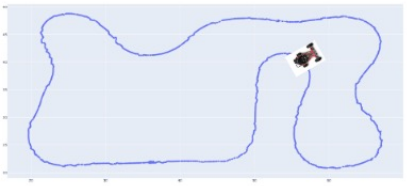
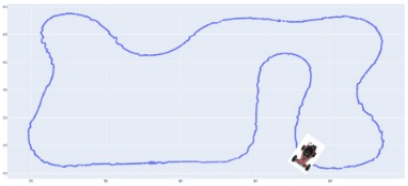
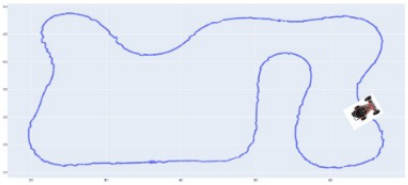
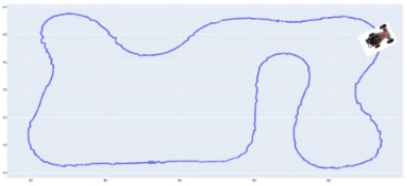
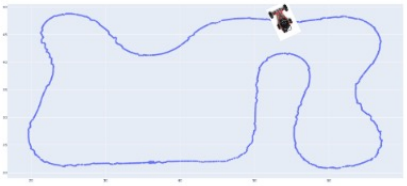
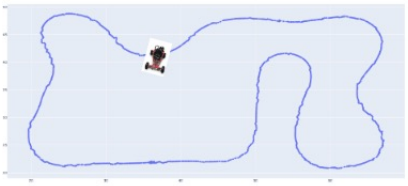
Therefore by performing linear regression on the 3 sets of values as shown in Table 4.8, the equation can be deduced as -

$$lookahead\_distance = 2 \times v\_desired - 1. \tag{4.2}$$

4.5 Take-me-Home Functionality Mode

This section shows the results of several test runs testing the Take-me Home functionality. In each test run, the gokart was crashed at different positions on the track at different angles to test if the gokart is able to recover from different deadlock positions. From several test runs and deadlock positions, it was concluded that the gokart was able to recover from any deadlock position provided that it does not turn more than 180° from heading direction in the deadlock position and enough turning space is there for the gokart. The results can be seen in Table 4.9. It can be seen that the gokart was able to recover from all the deadlock positions which are depicted in Table 4.9 that the gokart may encounter. To ensure the algorithm works, functionalities like guardian angel should be used which can detect that the gokart is about to collide and thus can stop gokart in an orientation close to the centerline. This can ensure that the gokart doesn't turn more than 180° before the crash and have enough turning room thus ensuring Take-me Home functionality to function without failure.

S.No.	Deadlock Position	Recovered
1		Yes
2		Yes
3		Yes

4		Yes
5		Yes
6		Yes
7		Yes
8		Yes
9		Yes

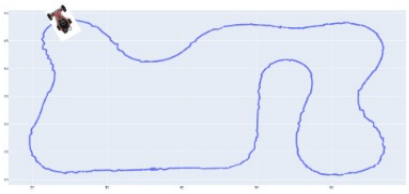
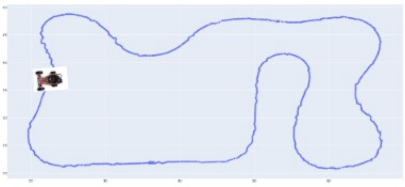
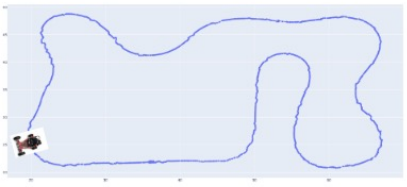
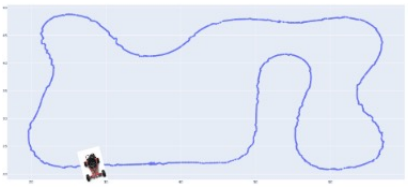
10		Yes
11		Yes
12		Yes
13		Yes

Table 4.9: Table showing the results of Take-me Home functionality for different deadlock positions
,

Chapter 5

Conclusion

In this thesis, a pure pursuit controller for gokart to track path was implemented. The implementation was further improved by introducing adaptive lookahead distance based on the velocity of the gokart. Using the adaptive pure pursuit controller improved the robustness of the controller significantly. The pure pursuit controller was integrated with the existing pipeline of the autonomous gokart and was tested extensively in the real world. This controller was developed because it produces a good path tracking algorithm and can be used as an alternate path tracking algorithm. The algorithm is developed in such a way that with minor adjustments in the hyper-parameters, it can be integrated into any gokart model.

The thesis work also evaluated the effects of changing the hyper-parameters such as velocity, lookahead distance and weights on the controller. The outcomes were measured by quantitative and qualitative metrics defined in the thesis. This helped in estimating the ideal hyper-parameters in which the pure pursuit controller can function without failure on the gokart. Several test runs were conducted to find the ideal range of velocity, lookahead distances corresponding to velocity and weight range in which the gokart can move.

This thesis also evaluated the comparison of the pure pursuit controller with the existing MPC controller in terms of path tracking and mean deviation from the path. It was concluded that the Pure Pursuit controller performs at par with the MPC controller for lower speeds. But at higher speeds, the MPC controller outperforms the Pure Pursuit controller because, at higher speeds, the Pure Pursuit controller uses longer lookahead distances which starts cutting corners and taking longer time to realign with the centreline.

Once the hyper-parameters were tuned, the controller was further extended to enable the gokart to move in reverse mode on the track. Using this, the gokart was able to track the path and complete the track runs in reverse mode. The parameters were finely tuned for the reverse mode. It was also observed that the controller was able to function at slower speeds in reverse mode. This functionality can also be used in Autonomous docking and Inverse Parking Manoeuvre but this lacked robustness. With some changes, these can be made more robust which can be future work in this direction.

As the forward and the reverse mode of the gokart were extensively tested to fine-tune to parameters and were made robust, the implementation was further extended to Take-me Home functionality. This functionality is developed to drive the gokart completely driverless. This functionality can make the gokart recover from any deadlock or crashed position and reach back to the starting position. This functionality was extensively tested by crashing gokart at random positions and in

random orientations and then seeing if the gokart is able to recover. It was concluded that the gokart was able to recover from any crashed position if sufficient turning room is available and the gokart doesn't rotate more than 180°. This functionality can be used to test any controller.

As a conclusion to the thesis, a robust Adaptive Pure Pursuit controller was successfully developed with forward, reverse, Take-me Home functionality modes. Extensive testing in each of the 3 modes was done to tune the hyper-parameters and test the robustness of the controller. This functionality can in future be integrated with the Guardian Angel functionality to enable testing of the controller. Guardian Angel can detect if the gokart is about to crash and can stop the gokart preventing the crash and also triggering the Take-me Home functionality of the pure pursuit controller. Then the gokart can recover from that position and reach the starting position. Another future work that can be done is to integrate it with the other existing functionalities to increase the maximum velocity at which the controller can function. The Take-me Home functionality can also be improved by using a better method of generating the path for reversing the gokart from crashed position to the realignment point. Once all these functionalities are developed and integrated, an autonomous controller testing method can be developed which doesn't need a continuous need of any human.

Appendix A

Instructions to Run

A.1 Common Steps

- Install the repository and checkout to dev-pure-pursuit-final branch. Also go to gokart-ros-base repository and checkout to dev-pure-pursuit.
- Go to gokart core package and run command make build.
- Go to /gokart-env-developer/gokart_ws/src/gokart-core/packages/motion/control/pure_pursuit_controller/param and open pure_pursuit_params.yaml. Change the mode in it to the desired mode -

  Forward - to follow a given path in forward direction

  Reverse - to follow a given path in reverse direction

  Deadlock - to implement take-me home functionality once a test controller fails

A.1.1 Forward Mode

- Change the mode to "forward" mode in pure_pursuit_params.yaml
- Generate a path which is an array of Pose2D by any algorithm. Publish this path via the rostopic - /kitt/path/path_array of type Path.msg. If no other algorithm is available, then publish the centerline by going to gokart-core directory and running make run-pure-pursuit-center-path-extractor
- Go to gokart-core directory and run make run-pure-pursuit-at-beginning.
- Go to gokart-core directory and run make run-pure-pursuit-at-deadlock.
- Press the boost button now to run gokart.

A.1.2 Reverse Mode

- Change the mode to "reverse" mode in pure_pursuit_params.yaml
- Generate a path which is an array of Pose2D by any algorithm. Publish this path via the rostopic - /kitt/path/path_array of type Path.msg. If no other algorithm is available, then publish the centerline by going to gokart-core directory and running make run-pure-pursuit-center-path-extractor
- Go to gokart-core directory and run make run-pure-pursuit-at-beginning.
- Go to gokart-core directory and run make run-pure-pursuit-at-deadlock.

- Press the boost button now to run gokart.

A.1.3 Take-me-Home Functionality Mode

- Install the repository and checkout to dev-pure-pursuit-final branch. Also go to gokart-ros-base repository and checkout to dev-pure-pursuit.
- Go to gokart core package and run command make build.
- Go to /gokart-env-developer/gokart_ws/src/gokart-core/packages/motion/control/pure_pursuit_controller/param and open pure_pursuit_params.yaml. Change the mode to "deadlock" mode in pure_pursuit_params.yaml
- Generate a path which is an array of Pose2D by any algorithm. Publish this path via the rostopic - /kitt/path/path_array of type Path.msg. If no other algorithm is available, then publish the centerline by going to gokart-core directory and running make run-pure-pursuit-center-path-extractor
- Go to gokart-core directory and run make run-pure-pursuit-at-beginning.
- Start the testing controller which is to be tested and allow it to run. If no testing controller available, use the manual mode to run gokart to a deadlock position.
- Once the gokart leaves the track, apply break to it and stop the testing controller. If no testing controller available, then stop the manual mode if running.
- Go to gokart-core directory and run make run-pure-pursuit-at-deadlock.
- Press the boost button now to run gokart.

Appendix B

Node Running

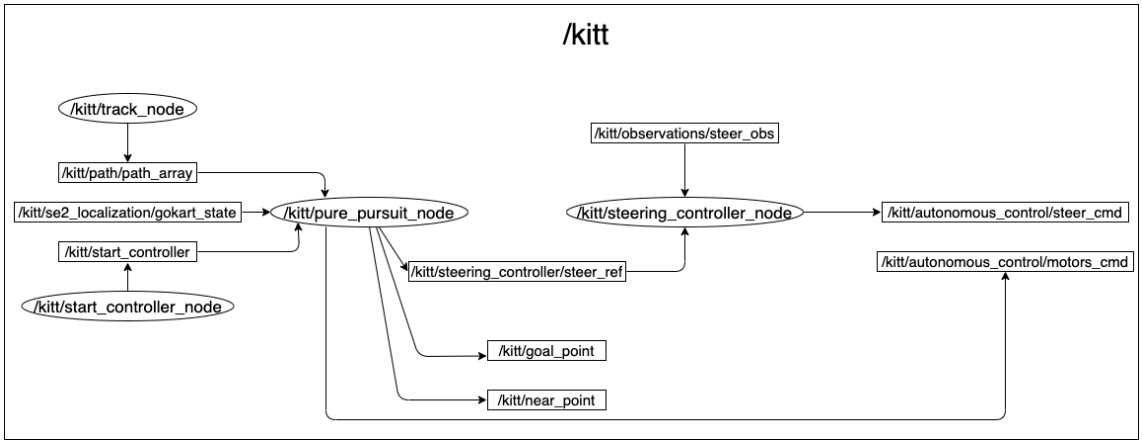


Figure B.1: Plot showing the nodes and rostopics running

B.1 pure_pursuit_node

This node runs the pure pursuit controller in the desired mode. It takes input - path to be tracked, gokart state and when to start the controller. It then processes the inputs to find the steering angle and the motors command. It publishes the steering angle to steering controller and publishes motor commands to motors.

B.1.1 Parameter File

The file with the parameters for the pure pursuit controller is - param/pure_pursuit_params.yaml

B.1.2 Subscribed Topics

- /kitt/path/path_array: gokart_msgs/msg/Path
- /kitt/se2_localization/gokart_state: gokart_msgs/msg/GokartState
- /kitt/start_controller: std_msgs/msg/Bool

B.1.3 Published Topics

- /kitt/steering_controller/steer_ref: gokart_msgs/msg/SteerRef
- /kitt/near_point: visualization_msgs/msg/Marker
- /kitt/goal_point: visualization_msgs/msg/Marker
- /kitt/autonomous_control/motors_cmd: gokart_cmd_msgs/msg/MotorsCmd

B.2 steering_controller_node

This node receives the steer ref angle from pure_pursuit_node, implements PID to it and calculates the torque to be published to steer_cmd.

B.2.1 Parameter File

The file with the parameters for the pure pursuit controller is - param/steering_controller_params.yaml

B.2.2 Subscribed Topics

- /kitt/steering_controller/steer_ref: gokart_msgs/msg/SteerRef
- /kitt/observations/steer_obs: gokart_obs_msgs/msg/SteerObs

B.2.3 Published Topics

- /kitt/autonomous_control/steer_cmd: gokart_cmd_msgs/msg/SteerCmd

B.3 track_node

This node generates the center line of default track and publishes it. It is used as default for pure pursuit controller.

B.3.1 Parameter File

The file with the parameters for the pure pursuit controller is - param/pure_pursuit_controller_param.yaml

B.3.2 Published Topics

- /kitt/path/path_array: gokart_msgs/msg/Path

B.4 start_controller_node

This node instructs the pure_pursuit_node to start running the gokart.

B.4.1 Published Topics

- /kitt/start_controller: std_msgs/msg/Bool

Bibliography

[1] M. Heim, *Autonomous Go-kart Racing*, Institute for Dynamic Systems and Control (IDSC), ETH Zürich, Switzerland, Mar. 2019.

[2] T. Andrew, *Assistive Go-kart Racing*, Institute for Dynamic Systems and Control (IDSC), ETH Zürich, Switzerland, Jan. 2020.

[3] F. Schmoll, *The Gokart Plugin*, Institute for Dynamic Systems and Control (IDSC), ETH Zürich, Switzerland, May 2020.

[4] R. C. Conlter, *Implementation of the Pure Pursuit Path Tracking Algorithm*, The Robotics Institute, Carnegie Mellon University, Jan. 1992.

[5] R. Wang, Y. Li, J. Fan, T. Wang, and X. Chen, "A novel pure pursuit algorithm for autonomous vehicles based on salp swarm algorithm and velocity controller," *IEEE Access*, vol. 8, pp. 166 525–166 540, 2020.

[6] H.-G. Park, K.-K. Ahn, M.-K. Park, and S.-H. Lee, "Study on robust lateral controller for differential gps-based autonomous vehicles," *International Journal of Precision Engineering and Manufacturing*, vol. 19, no. 3, pp. 367–376, Mar 2018. [Online]. Available: https://doi.org/10.1007/s12541-018-0044-9

[7] L. Kong, "Adaptive pure pursuit model for autonomous vehicle path tracking," 2017.

[8] H. Ohta, N. Akai, E. Takeuchi, S. Kato, and M. Edahiro, "Pure pursuit revisited: Field testing of autonomous vehicles in urban areas," 10 2016, pp. 7–12.

[9] S. Prabhakaran, "Evaluating the viability of adaptive pure pursuit control in low resource skid-steer drive mobile robots," 04 2019.

[10] K. Lee, S. Jeon, H. Kim, and D. Kum, "Optimal path tracking control of autonomous vehicle: Adaptive full-state linear quadratic gaussian (lqg) control," *IEEE Access*, vol. 7, pp. 109 120–109 133, 2019.

[11] A. R. Tawfeek, "An introduction to geodesics: the shortest distance between two points," 2020.

[12] G. Jahangirova, A. Stocco, and P. Tonella, "Quality metrics and oracles for autonomous vehicles testing," 04 2021.

ETH zürich

Institute for Dynamic Systems and Control

Prof. Dr. R. D'Andrea, Prof. Dr. E. Frazzoli, Prof. Dr. Lino Guzzella, Prof. Dr. C. Onder, Prof. Dr. M. Zeilinger

Title of work:

Pure Pursuit controller of an Autonomous Gokart

Take-me-home functionality

Thesis type and date:

Semester Project, April 2021

Supervision:

Alessandro Zanardi
Prof. Dr. Emilio Frazzoli

Student:

Name: Tanmay Goyal
E-mail: tgoyal@student.ethz.ch
Legi-Nr.: 20-949-137
Semester: FS 2021

Statement regarding plagiarism:

By signing this statement, I affirm that I have read and signed the Declaration of Originality, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Declaration of Originality:

`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/declaration-originality.pdf`

Zurich, 18.10.2022: _____