# Pure Pursuit with PID

Haris Khan, Jonny Moreira

# Revisions Performed

## Switched Odometry Source

- Now subscribes to PoseArray on /pose_info and pulls the vehicle's pose from msg.poses[1] (per instructor's note)

## Replaced Velocity-PID with Position-PID

- PID now computes a velocity reference based on the position error between the current pose and the next waypoint

## Updated Slides & Code Organization

Labor Share:

Haris Khan: Most of the Coding for Part 1, Testing and debugging for part 2, Powerpoint Creation

Jonny Moreira: Most of the Coding for Part 2, testing and debugging for part 1,

# Goals

- Understand and apply the Pure Pursuit algorithm for lateral control.

- Gain experience integrating both longitudinal and lateral controllers for trajectory-tracking control.

- Implement path-following using real-time pose and waypoint tracking.

- Develop ROS2 nodes that interact through topics and process real-time data.

-Gazebo based simulation

# Phase 1

Objective: Implement a Pure Pursuit Steering Controller (Lateral Control) that allows the autonomous car to follow a given path by steering appropriately using the provided waypoints.

Success Criteria:

Vehicle follows the waypoint path with mean lateral error < 0.5 m.

Controller runs at ≥ 10 Hz, publishing only angular velocity (or steering angle) to /cmd_vel.

No dependency on Gazebo or full vehicle dynamics—pure geometry-based logic.

# Our Approach

**Waypoint Loading:**

**• Read 'sonoma_waypoints.txt' (CSV) into arrays of X and Y coordinates.**

**• Ignore Z coordinate for 2D control.**

**Odometry Subscription:**

• Subscribe to '/odom' (nav_msgs/Odometry) to get current position (x,y) and heading (yaw).

**Look-Ahead Point Selection:**

• For each control cycle, find the first waypoint at least a constant look-ahead distance ahead of the vehicle

• Use a rolling index to optimize searching through the waypoint lust

**Steering Computation:**

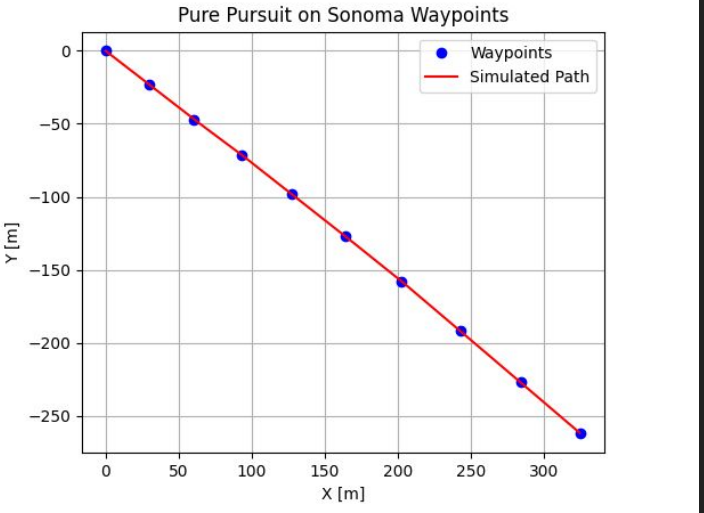
• Compute heading error between vehicle orientation and direction to look-ahead point.

• Calculate steering angle using the pure-pursuit formula: δ = arctan((2 * L * sin(α)) / L_d)

where L is wheelbase and L_d is look-ahead distance.

**Command Publication:**

• Publish a Twist message with constant forward speed and computed angular velocity to '/cmd_vel'.



# Phase 2

**Objective:** Involve the integration of the previous PID based velocity control with steering control. Both controllers will be simulated and tested in the Gazebo environment, first separately, and then, in integration. You are free to implement both controllers in the same file or split them into separate modules as you see fit.

**Success Criteria:** Successfully following the prescribed waypoint path while maintaining the target speed—i.e. tracking the trajectory with minimal lateral error ($\lesssim$0.5 m) and holding the commanded velocity within a tight tolerance.

# Our Approach

## PID Speed Controller:

• Read target speed parameter from configuration.

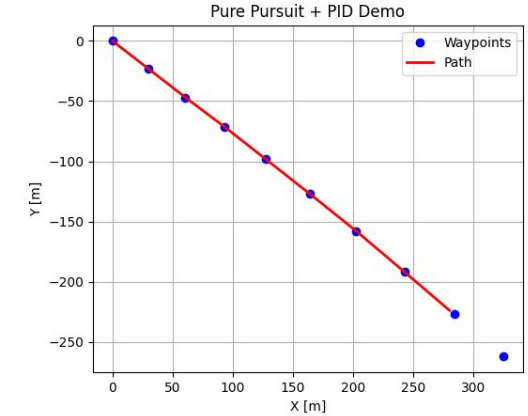
• Subscribe to '/odom' for current speed measurement.

• Compute speed error and update PID terms (proportional, integral with windup clamp, derivative).

• Output speed command 'u', clamped to vehicle limits.

## Speed Dynamics Model:

• Model forward speed as a first-order lag: v_dot = (u - v) / τ, with time constant τ




## Combined Control Loop:

• Each time step: run PID update, then pure-pursuit steering using same look-ahead logic.

• Update vehicle state for both

## Command Publication:

• Publish combined Twist: linear.x = u, angular.z = steering command.

## Validation:

• Offline demos produce speed_vs_time plots and animated path GIFs.

• In ROS 2/Gazebo, verify smooth trajectory following at target speed.