ENPM808X Midterm (Phase 0)

Justin Albrecht (111576951), Govind Kumar (116699488), Pradeep Gopal (116885027)

October 6 2020

1 Introduction

For this project we are going to implement a controller that uses the Ackermann kinematic steering equations for the Acme Robotics company. The Ackermann equations compute the ideal turning angle for both the inner and outer wheels to follow a specified turning radius. The basic idea behind Ackermann steering is that the inner wheel should steer for a bigger angle when compared to the outer wheel. This stops the wheels from slipping side ways when the vehicle follows a curved path. This is vital for any robot which uses Ackermann steering. This controller will be readily available to be integrated with the motor control functionalities of the robot. Based on the target heading and target velocity to the robot, this controller will compute the ideal turning angle for both wheels as well as the wheel velocities, which in turn will be fed as input to the next low level motor controllers which sends input to the hardware motor driver. We are assuming that the controller is for a four wheeled robot with front-wheel steering and rear-wheel drive.

2 Design and Development

This project is going to be implemented by using the pair programming procedure. To maintain more quality to the product, we have an additional design keeper who will overlook the operations and make sure the implementation is following the project design.

Phase 1 : Pradeep - Navigator, Govind - Driver, Justin - Design Keeper
Phase 2 : Pradeep - Design Keeper, Govind - Navigator, Justin - Driver

The C++ programming language will be used on a Linux environment with "make" build system. Quality to the code will be ensured by using sufficient unit testing to test every module. Regular commits with meaningful messages in GitHub will be followed. Tools such as cppcheck, Google styleguides with cpplint validation, Travis, Coverall.io coverage of (90+) and Valgrind will be used.

3 Equations

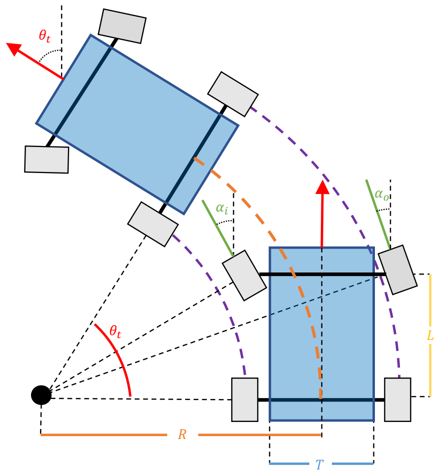

Figure 1: Robot turning with Ackermann steering

Terms

- θ_t - Target heading of the robot
- v_t - Target velocity of the robot
- T - Track length (distance between wheels on an axle)
- L - Wheel base (distance between two axles)
- r_w - Wheel radius
- R - Turning radius
- α_i - Turning angle for the inner wheel
- α_o - Turning angle for the outer wheel

For a given turning radius we can compute ideal turning angles for both the inner and outer wheels:

$$\alpha_i = \arctan(\frac{L}{R - T/2}) \tag{3.1}$$

$$\alpha_o = \arctan(\frac{L}{R + T/2}) \tag{3.2}$$

We can assume that after the turn has been completed we turn the wheels back to straight and convert all of our rotational velocity to linear velocity. Using the target velocity we then compute how long it will take to complete the turn.

$$v_t = \omega_{robot} R \tag{3.3}$$

$$\omega_{robot} = \frac{\theta_t}{t} \tag{3.4}$$

$$t = \frac{R\theta_t}{v_t} \tag{3.5}$$

The robot will drive in a circle around the turning radius defined from the center of the rear axle. The inner and outer wheels will therefore follow concentric arcs. The lengths of these arcs is:

$$s_i = (R - T/2)(\theta_t) \tag{3.6}$$

$$s_o = (R + T/2)(\theta_t) \tag{3.7}$$

We can compute the angular velocity for each wheel using the arc length, the wheel radius, and the time to complete the arc:

$$\omega_i = \frac{s_i}{r_w * t} \tag{3.8}$$

$$\omega_o = \frac{s_o}{r_w * t} \tag{3.9}$$

4 Class Structure

To build out the controller we propose using the below class structure. The `AckermannController` class will be a child class of the `Robot` class. The exact definition of the Robot class is unknown to us but we are assuming that it will have at least the specified attributes and methods shown.

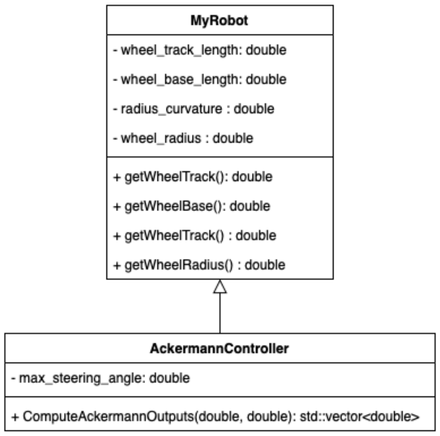
MyRobot
- wheel_track_length: double
- wheel_base_length: double
- radius_curvature : double
- wheel_radius : double
+ getWheelTrack(): double
+ getWheelBase(): double
+ getWheelTrack() : double
+ getWheelRadius() : double
AckermannController
- max_steering_angle: double
+ ComputeAckermannOutputs(double, double): std::vector<double>

Figure 2: UML Diagram

The `ComputeAckermannOutputs()` method will take in doubles for both the target heading and velocity and will output a vector of four doubles $(\alpha_i, \alpha_o, \omega_i, \omega_o)$.