ENPM808X Midterm (Phase 2)

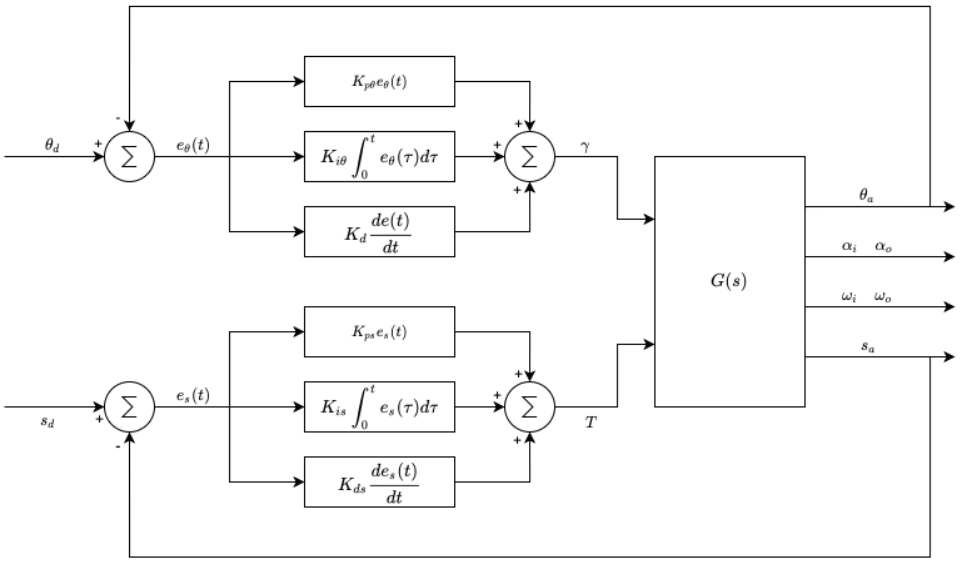
Justin Albrecht (111576951), Govind Kumar (116699488), Pradeep Gopal (116885027)

November 3 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 assume that a four wheeled vehicle travels around an instantaneous center of curvature and can compute the kinematics for given turning angles for both the inner and outer wheels. 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. We are assuming that the controller is for a four wheeled robot with front-wheel steering and rear-wheel drive.

2 Control System



θ_d	Desired heading
s_d	Desired speed
$e_\theta(t)$	Heading error function
$e_s(t)$	Speed error function
γ	Steering angle
T	Throttle
θ_a	Actual heading
$\alpha_i \quad \alpha_o$	Inner/outer wheel angles
$\omega_i \quad \omega_o$	Inner/outer wheel velocities
s_a	Actual speed

Figure 1: Control Diagram

The output of the first controller is γ. This value needs to be in a range:

$$\gamma \in [-45, 45] \tag{2.1}$$

The throttle value is directly related to how quickly the wheels of the vehicle are able to accelerate/decelerate. This value α_{max} will depend on the robot but we will assume a reasonable value.

$$\frac{d\omega_w}{dt} = \alpha_w \in [-\alpha_{max}, \alpha_{max}] \tag{2.2}$$

We can extrapolate the max and min throttle values using the controller tick rate dt

$$T \in [-\alpha_{max}dt, \alpha_{max}dt] \tag{2.3}$$

The program will ensure the output of γ and T are within these bounds.

3 Design and Development

This project is going to be implemented by using the pair programming procedure. To maintain quality of 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.

4 Equations

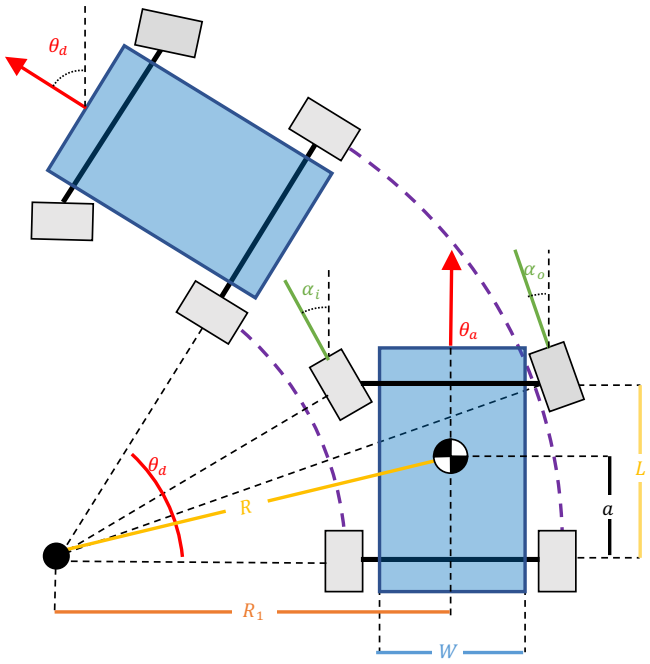


Figure 2: Robot turning with Ackermann steering

Terms

- θ_d - Desired heading of the robot
- s_d - Desired speed of the robot
- W - Track length (distance between wheels on an axle)
- L - Wheel base (distance between two axles)
- a - offset of COM from wheel axle
- 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 γ we can compute the turning radius of the vehicle:

$$R = \sqrt{a^2 + L^2 \cot^2 \gamma} \tag{4.1}$$

We can also compute the individual wheel angles which are the cotangent average of γ.

$$\alpha_i = \arctan\left(\frac{L}{R_1 - W/2}\right) \tag{4.2}$$

$$\alpha_o = \arctan\left(\frac{L}{R_1 + W/2}\right) \tag{4.3}$$

where:

$$R_1 = \sqrt{R^2 - a^2} \tag{4.4}$$

The output of the controller T is directly related to the change in the wheel's angular velocity.

$$d\omega = T \tag{4.5}$$

This $d\omega$ will be added to the outer wheel of the turn since that wheel moves faster.

$$\omega_{o_{new}} = \omega_{o_{old}} + d\omega \tag{4.6}$$

We are also assuming the car cannot go in reverse so ω_i and ω_o can never be less than zero. We also cannot accelerate forever there will be max values for ω_i and ω_o as well. We will never exceed those values.
After we know the wheel velocity for the outer wheel and the turning radius we can compute the change in heading and velocity for a given dt. To start we will compute the length of the arc that the outer wheel will follow. This is again assuming 0 slip. We want to multiply the circumference of the wheel times the number of rotations that the wheel will undergo in dt.

$$\text{circumference} = 2\pi r_w \tag{4.7}$$

$$\text{rotations} = \frac{\omega_o}{2\pi} dt \tag{4.8}$$

The 2π cancels

$$\text{arc length} = r_w \omega_o dt \tag{4.9}$$

We can then get the change in heading $d\theta$

$$d\theta = \frac{\text{arc length}}{R_1 + \frac{T}{2}} \tag{4.10}$$

Because the wheel velocities are related we can also compute the inner wheel velocity from our new $d\theta$

$$\omega_i = \frac{d\theta \left(R_1 - \frac{T}{2}\right)}{r_w dt} \tag{4.11}$$

For the case where the steering angle is zero:

$$\alpha_i, \alpha_o = 0 \tag{4.12}$$

$$\omega_{o_{new}} = \omega_{i_{new}} = \texttt{max}(\omega_{o_{old}}, \omega_{i_{old}}) + T \tag{4.13}$$

The heading of the robot does not change and the speed can be calculated using ω and r_w

$$\text{speed} = \frac{\text{distance}}{\text{time}} \tag{4.14}$$

$$\text{distance} = \text{rotations} \times \text{circumference} \tag{4.15}$$

$$\text{circumference} = 2\pi r_w \tag{4.16}$$

$$\text{rotations} = \frac{\omega_o}{2\pi} dt \tag{4.17}$$

$$\text{distance} = r_w \omega dt \tag{4.18}$$

$$\text{speed} = r_w \omega \tag{4.19}$$

5 Class Structure

To build out the program we are going to use the below class structure. The `Controller` class will be a child class of the `Robot` class.

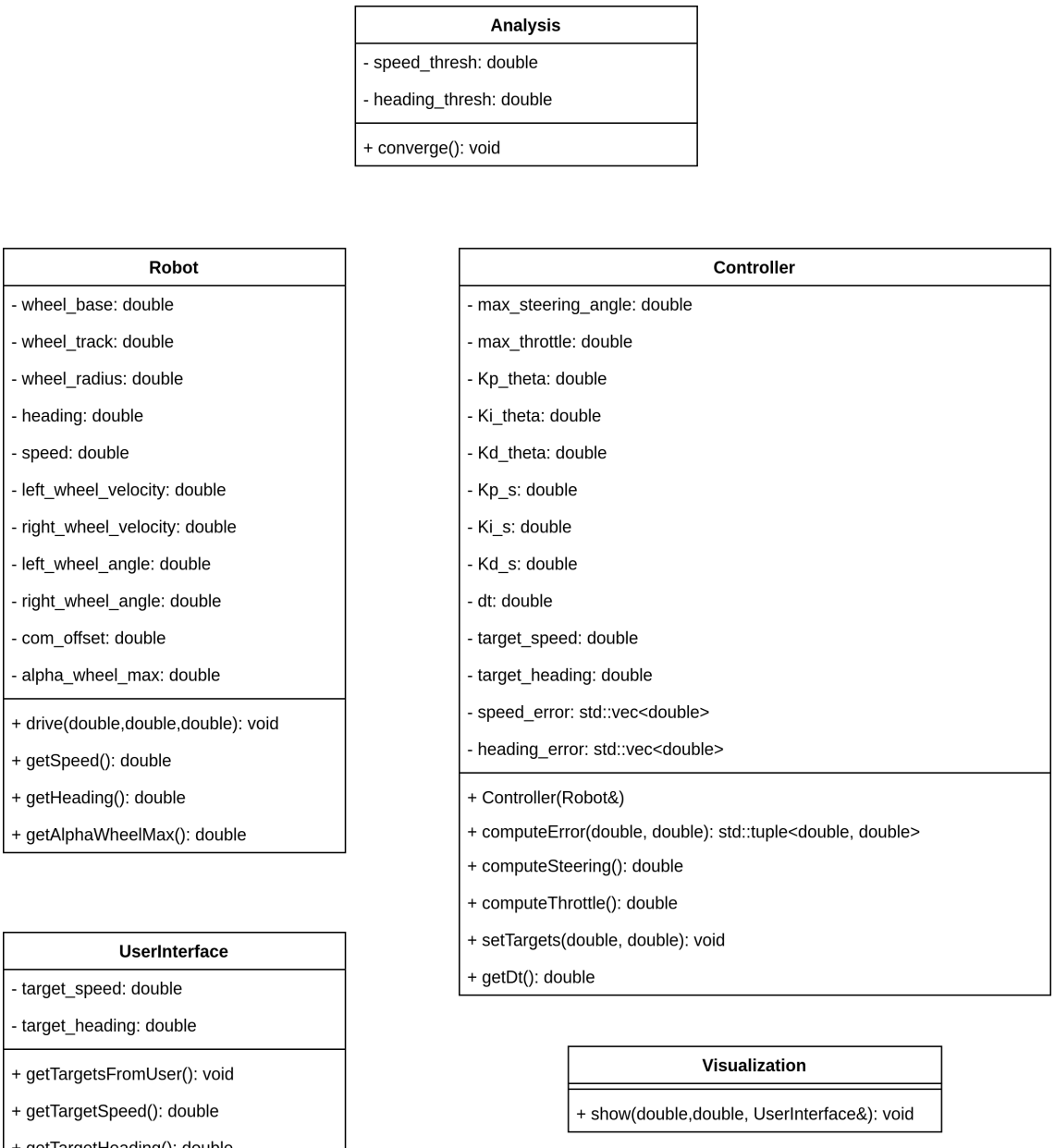


Figure 3: UML Diagram

- The `Robot` class will have several attributes that are intrinsic to how the robot is built such as `wheel_base` or `wheel_radius`. It also as attributes for the robot's current heading and velocity as well as the wheel velocities. All attributes have been classified as private attributes so if then need to be used outside of the class we will also need to create getters or setters. The class has one method `drive()` that takes in a throttle value and a steering angle and modifies the robot's heading, velocity and wheel velocities.
- The `Controller` class contains the two PID controllers that will be used to update the throttle and steering angle. The control constants (K) are attributes as well as two vectors for the error for both speed and heading. These vectors will be updated as the robot attempts to converge with the set points. There are three methods in this class. The first method computeError() will update the error vectors according to the actual speed and heading from the `drive` method in the robot class. The `computeSteering()` method outputs a steering angle using the heading error vector and the heading control constants. The `computeThrottle` method outputs a throttle value using the speed error and the speed control constants.
- The `UserInterface` class will be used to interact with the user and gather the target velocity and heading.
- The `Visualization` class will output a display that shows the heading and velocity converging with the set points from the user. Our current plan is to just print out to the console but if we have time we would like to include a more in depth visualization, possibly using something like OpenCV.
- The last class `Analysis` is essentially the `main()` function that will contain all the code necessary to integrate the various other classes and create a working system. When the program is run from terminal it will call the `converge()` method and the user will input the set values and the visualization will be shown.