ACKERMANN STEERING CONTROL

Pratik Sunil Bhujbal
UID: 117555295

Maitreya Kulkarni
UID: 117506075

Mohammed Maaruf Vazifdar
UID: 117509717

Project Overview: Ackermann steering condition arises when axes of all wheels in a vehicle intersect at a single turning point. In the case of Ackermann steering the wheels of the vehicle do not skid while turning as opposed to parallel steering. This reduces the tyre wear and increases the energy efficiency. Ackermann steering can be used in various autonomous systems such as self-driving cars, delivery robots which can increase the efficiency of the robot.

Ackermann steering assumes that the steering angle is calculated with respect to a common centre. In this case, rear wheels are the driving wheels without any steerage. The steering centre is assumed to be on the line extended from the rear axle.

During the development the following assumptions are made:

- All robot parameters are known.
- Initially the robot is on the origin of the world frame facing the x axis.
- Robot's desired heading and velocity are given by the user as input.
- Maximum steering angle is 45 degrees.
- Friction and wheel slippage are considered negligible.

Project Technologies:

- Ubuntu 18.04
- Build System: cmake
- Version control: GitHub
- Software Tools: VS Code
- Build Check: Travis
- Code Coverage: Coveralls
- Gazebo API: To interface with the robot model in the Gazebo environment.

Development Process: This project involves three programmers, following the Agile Iterative Process and programming in-pairs. To ensure product quality roles of driver, navigator and design-keeper will be switched amongst the team regularly. To prevent occurrence of bugs and failing of sub- modules the team will adapt to Test Driven Development philosophy. All programming will be done using modern C++, following Google Style Sheets and abiding to the practices of OOPS.

Algorithm: As the vehicle travels along a curved path, all four tyres follow unique trajectories around a shared turn centre, as defined by the blue arcs in Figure 1.

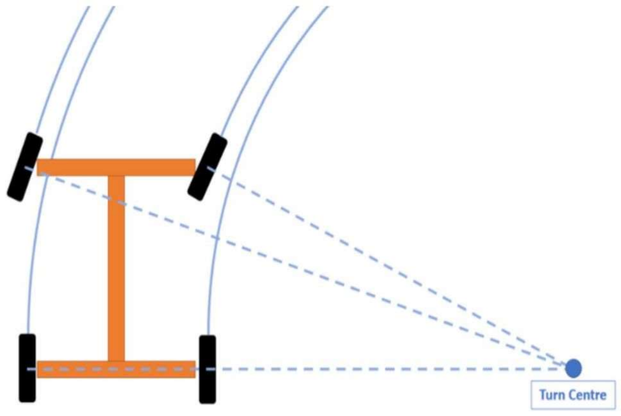


Fig.1: Wheel Trajectories

For a given turn radius R, wheelbase L, and track width T, the required front steering angles (δ_(f,in) and δ_(f,out)) with the following expressions:

$$\delta_{f,in} = \tan^{-1}\left(\frac{L}{R-\frac{T}{2}}\right) \quad \delta_{f,out} = \tan^{-1}\left(\frac{L}{R+\frac{T}{2}}\right)$$

Similarly, we can calculate the wheel speeds using the following equations, where desired drive speed is denoted by $\boldsymbol{VD}$ while $\boldsymbol{VRF}$, $\boldsymbol{VLF}$ are front right and left wheel speed., $\boldsymbol{l}$ denotes the length of vehicle, $\boldsymbol{d}$ denotes the distance between wheel and kinematic centre and $\boldsymbol{\varphi}$ is steer angle.

$$v_{LF} = \frac{\sqrt{(R-d)^2 + l^2} \cdot |\tan\varphi|}{l} \cdot v_D$$

$$v_{RF} = \frac{\sqrt{(R+d)^2 + l^2} \cdot |\tan\varphi|}{l} \cdot v_D$$

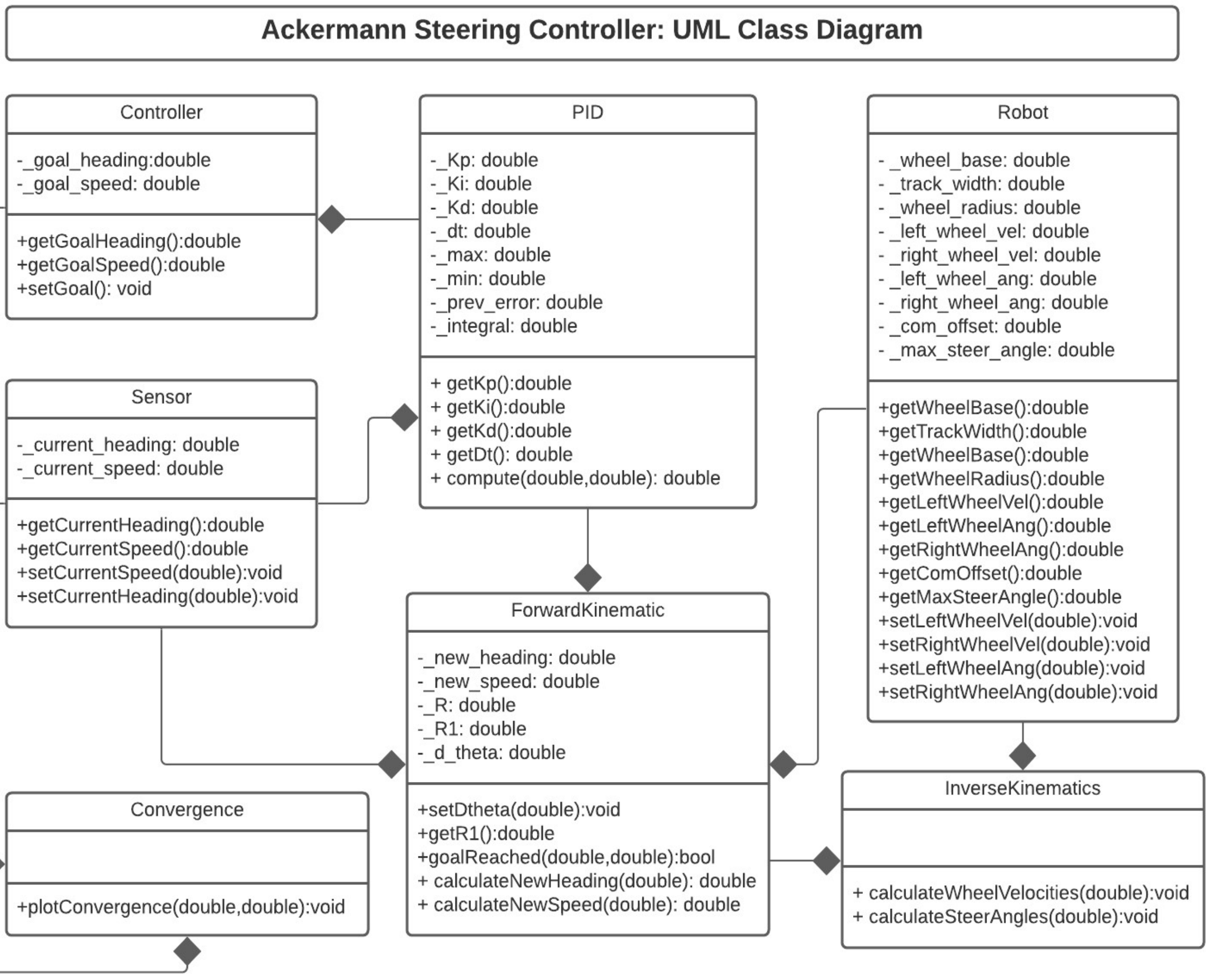
Class Diagram

As per feedback changed the development approach as shown below in **Class diagram** section.

Deliverables: A robust Ackermann Steering Controller that once integrated will help steer the robot to desired setpoints.

References:

- Kelly, Alonzo & Seegmiller, Neal. (2014). A Vector Algebra Formulation of Mobile Robot Velocity Kinematics. Springer Tracts in Advanced Robotics. 92. 613-627. 10.1007/978- 3-642-40686-7_41.



Ackermann Steering Controller: UML Class Diagram
Controller
-_goal_heading:double
-_goal_speed: double
+getGoalHeading():double
+getGoalSpeed():double
+setGoal(): void
PID
-_Kp: double
-_Ki: double
-_Kd: double
-_dt: double
-_max: double
-_min: double
-_prev_error: double
-_integral: double
+ getKp():double
+ getKi():double
+ getKd():double
+ getDt(): double
+ compute(double,double): double
Robot
- _wheel_base: double
- _track_width: double
- _wheel_radius: double
- _left_wheel_vel: double
- _right_wheel_vel: double
- _left_wheel_ang: double
- _right_wheel_ang: double
- _com_offset: double
- _max_steer_angle: double
+getWheelBase():double
+getTrackWidth():double
+getWheelBase():double
+getWheelRadius():double
+getLeftWheelVel():double
+getLeftWheelAng():double
+getRightWheelAng():double
+getComOffset():double
+getMaxSteerAngle():double
+setLeftWheelVel(double):void
+setRightWheelVel(double):void
+setLeftWheelAng(double):void
+setRightWheelAng(double):void
Sensor
-_current_heading: double
-_current_speed: double
+getCurrentHeading():double
+getCurrentSpeed():double
+setCurrentSpeed(double):void
+setCurrentHeading(double):void
ForwardKinematic
-_new_heading: double
-_new_speed: double
-_R: double
-_R1: double
-_d_theta: double
+setDtheta(double):void
+getR1():double
+goalReached(double,double):bool
+ calculateNewHeading(double): double
+ calculateNewSpeed(double): double
Convergence
+plotConvergence(double,double):void
InverseKinematics
+ calculateWheelVelocities(double):void
+ calculateSteerAngles(double):void

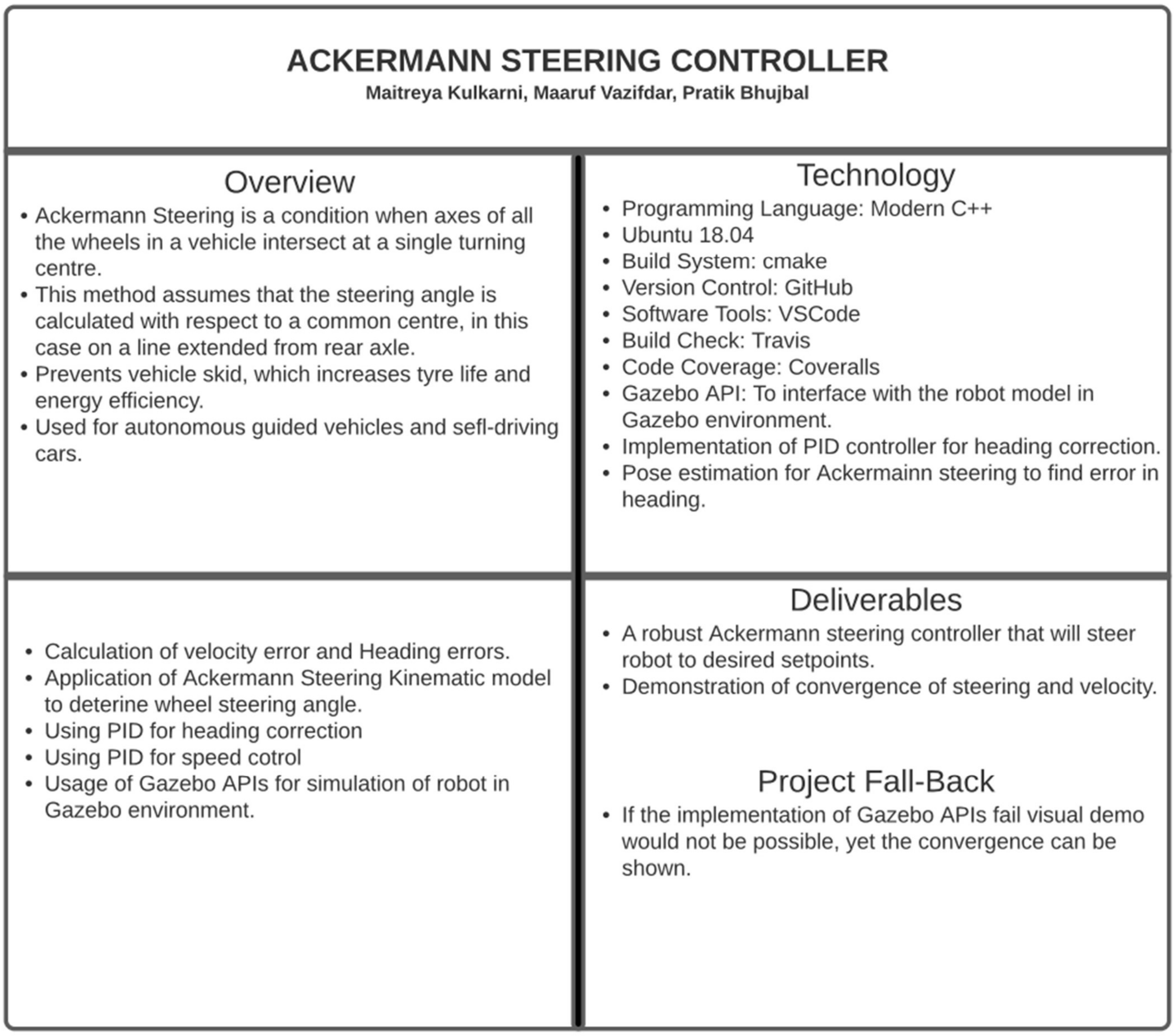
Fig.2 Class Diagram

Fallback Plan:

Implementation aims to demonstrate convergence to the setpoints. The plan is to simulate the project in Gazebo simulator using Gazebo API, but the fallback plan is to demonstrate the convergence without the simulation.

- A. J. Weinstein and K. L. Moore, "Pose estimation of Ackerman steering vehicles for outdoors autonomous navigation," 2010 IEEE International Conference on Industrial Technology, 2010, pp. 579-584, doi: 10.1109/ICIT.2010.5472738.
- https://www.xarg.org/book/kinematics/ackerman-steering/

QUADCHART



ACKERMANN STEERING CONTROLLER

Maitreya Kulkarni, Maaruf Vazifdar, Pratik Bhujbal

Overview

- Ackermann Steering is a condition when axes of all the wheels in a vehicle intersect at a single turning centre.
- This method assumes that the steering angle is calculated with respect to a common centre, in this case on a line extended from rear axle.
- Prevents vehicle skid, which increases tyre life and energy efficiency.
- Used for autonomous guided vehicles and sefl-driving cars.

Technology

- Programming Language: Modern C++
- Ubuntu 18.04
- Build System: cmake
- Version Control: GitHub
- Software Tools: VSCode
- Build Check: Travis
- Code Coverage: Coveralls
- Gazebo API: To interface with the robot model in Gazebo environment.
- Implementation of PID controller for heading correction.
- Pose estimation for Ackermainn steering to find error in heading.

- Calculation of velocity error and Heading errors.
- Application of Ackermann Steering Kinematic model to deterine wheel steering angle.
- Using PID for heading correction
- Using PID for speed cotrol
- Usage of Gazebo APIs for simulation of robot in Gazebo environment.

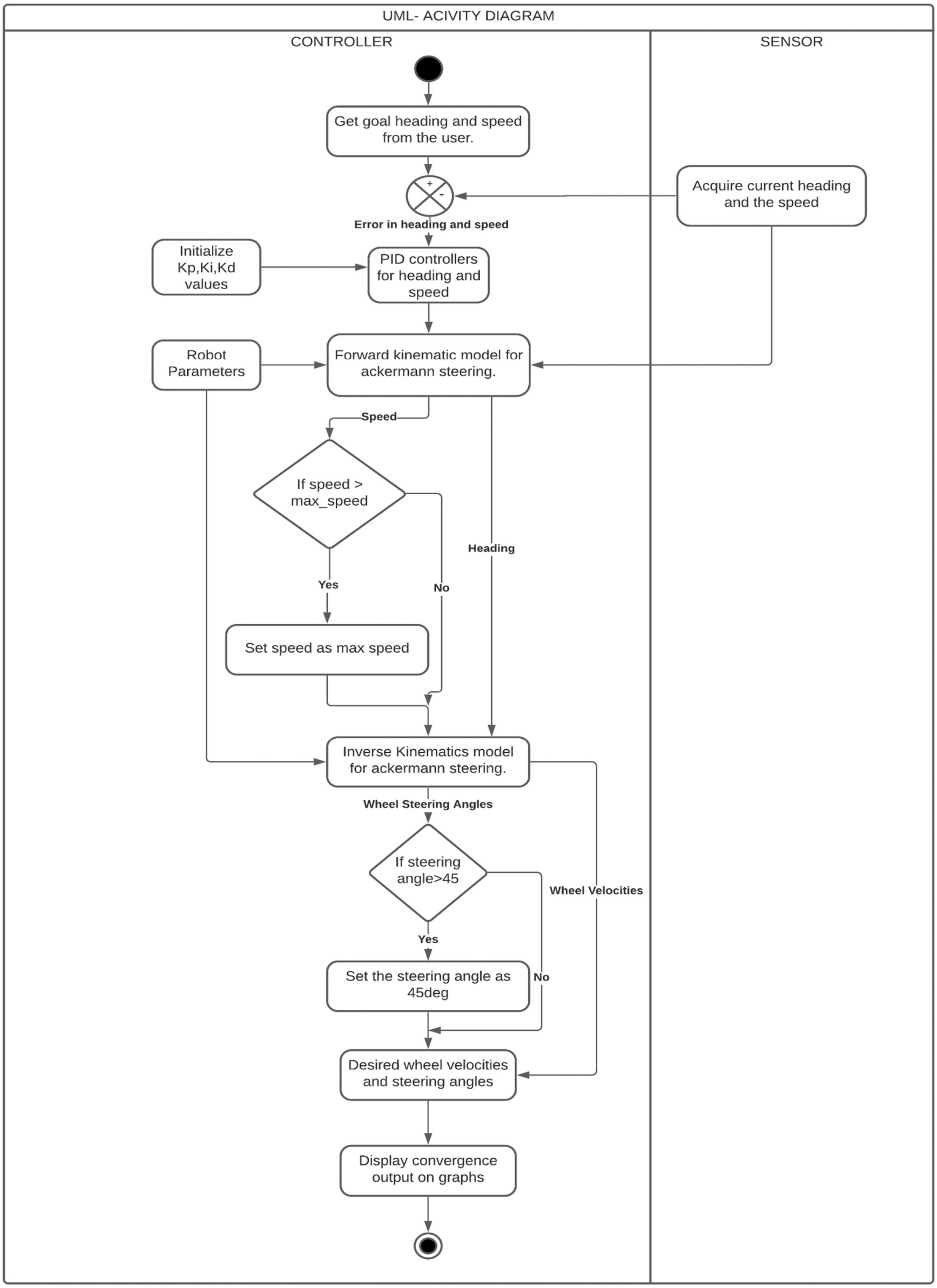
Deliverables

- A robust Ackermann steering controller that will steer robot to desired setpoints.
- Demonstration of convergence of steering and velocity.

Project Fall-Back

- If the implementation of Gazebo APIs fail visual demo would not be possible, yet the convergence can be shown.

Activity Diagram


UML- ACIVITY DIAGRAM
CONTROLLER
SENSOR
Get goal heading and speed from the user.
+
-
Acquire current heading and the speed
Error in heading and speed
Initialize Kp,Ki,Kd values
PID controllers for heading and speed
Robot Parameters
Forward kinematic model for ackermann steering.
Speed
If speed > max_speed
Heading
Yes
No
Set speed as max speed
Inverse Kinematics model for ackermann steering.
Wheel Steering Angles
If steering angle>45
Wheel Velocities
Yes
Set the steering angle as 45deg
No
Desired wheel velocities and steering angles
Display convergence output on graphs