

**EG6021 DESIGN OF MECHATRONICS SYSTEMS**

**Title: Enhancing the Performance of OSOYOO V2.1 Robot Car Using PID Control**

**Level: 6**

**Professor: Paola Falugi**

**Submission date: May 16th, 2024**

**Student numbers: 2192290 / 2195325 / 2195323 / 2126569 / 2190288**

# **Abstract**

This report presents the PID (Proportional-Integral-Derivative) control system for the OSOYOO V2.1 Robot Car to enhance its tracking capabilities on a designated track. This report involves developing a mathematical model, simulating the control system, and analyzing the performance based on stability, robustness, and speed. Design improvements for enhanced performance are also discussed.

A small robot with wheels and wires

Description automatically generated

A small robot with wheels and wires

Description automatically generated

**Contents**

[**Abstract** 2](#_Toc166766935)

[**1.** **Introduction** 6](#_Toc166766936)

[**2. Controller Design and Modelling** 7](#_Toc166766937)

[**2.1 PID Controller Design** 7](#_Toc166766938)

[**2.2 Modelling Approach** 11](#_Toc166766939)

[**3. Controller Performance and Limitations** 13](#_Toc166766940)

[**3.1 Effects of Saturation** 13](#_Toc166766941)

[**3.2 Modelling Assumptions and Limitations** 15](#_Toc166766942)

[**4. Simulation and Results** 17](#_Toc166766943)

[**4.1 Simulation Setup** 17](#_Toc166766944)

[**4.2 Results** 20](#_Toc166766945)

[**4.3 Anti-Windup Schemes** 23](#_Toc166766946)

[**4.4 SOLIDWORKS Simulation:** 27](#_Toc166766947)

[**5. Performance Analysis** 30](#_Toc166766948)

[**5.1 Lap Time Comparison** 30](#_Toc166766949)

[**5.2 Stability and Robustness** 31](#_Toc166766950)

[**6. Calculations** 33](#_Toc166766951)

[**6.1 Weight distribution** 33](#_Toc166766952)

[**6.2 Center of Mass** 36](#_Toc166766953)

[**6.3 Moment of Inertia** 38](#_Toc166766954)

[**6.4 Friction** 41](#_Toc166766955)

[**6.5 Angle of friction** 44](#_Toc166766956)

[**6.6 Velocity time graph using Matlab** 45](#_Toc166766957)

[**7. Design Improvements** 50](#_Toc166766958)

[**7.1 Enhanced Sensory Inputs** 50](#_Toc166766959)

[**7.2 Mechanical Adjustments** 51](#_Toc166766960)

[**8. Conclusion** 52](#_Toc166766961)

[**9. References** 53](#_Toc166766962)

[**10. Appendix** 55](#_Toc166766963)

# **Introduction**

The OSOYOO V2.1 Robot Car, which utilizes Arduino technology, is an excellent tool for learning about robotics and control systems. This project focuses on implementing a PID controller to help the car navigate a loop track with speed and precision.

In educational robotics, mastering advanced control systems like PID controllers is vital. These controllers are widely used because they can maintain desired outcomes with great accuracy. By implementing a PID controller in the OSOYOO V2.1 Robot Car, students can gain practical experience in fine-tuning and optimizing control systems to meet specific performance targets.

A PID controller works by continuously measuring the error between a desired setpoint and the car’s current position. It then adjusts the steering based on three components: proportional, integral, and derivative. The proportional component addresses the current error, the integral component corrects accumulated past errors, and the derivative component predicts future errors to smooth out the control response.

This project not only showcases the practical application of PID control but also highlights its importance in achieving precise and responsive robotic systems. By efficiently navigating a loop track, the robot car demonstrates how advanced control techniques can enhance performance. This project helps learners understand the complexities of control systems and their role in modern technology.

# **2. Controller Design and Modelling**

## **2.1 PID Controller Design**

An advanced control mechanism frequently used in industrial control systems where accuracy and precision are essential is the PID (Proportional-Integral-Derivative) controller (Omega, 2019). By using a PID controller, the OSOYOO V2.1 Robot Car's steering may be dynamically adjusted based on real-time feedback with the goal of minimizing cross-track error, or the lateral distance between the vehicle and the desired track path.

**Proportional Component (P)**

The PID controller's basic operation depends on the proportional term. The output it produces is exactly proportionate to the cross-track fault (Samak et al., 2021) When the robot car deviates from the intended path, the proportional control responds by producing a steering correction that is proportionate to the deviation. This implies that the corrective steering movement will be stronger the further the automobile is from the track center. One tuning parameter that can be used to modify the proportional response is *KP*. The controller's response to an error is scaled by this parameter. (Villafuerte et al., 2012) A car may overshoot or oscillate around the track center if its *KP* value is higher. However, a higher *KP* value might also cause an unstable response.

**Integral Component (I)**

The Integral component of the PID controller deals with accumulated errors over time, addressing any continuous offset that might prevent the PID from reaching the track center. (Tan et al., 2012) This could stem from systematic biases such as a misalignment in the steering mechanism or uneven weight distribution. By integrating the error over time and reacting based on the cumulative sum, this component helps to eliminate the residual steady-state error that the Proportional component alone might leave behind. The strength of the integral action is controlled by *Ki*, the integral gain. Appropriate tuning of *Ki* ensures that the controller corrects these biases without causing instability due to excessive action from high integral values.

**Derivative Component (D)**

Predictive is the derivative component. It dampens the system response by providing output dependent on the cross-track error's rate of change. This part is essential for facilitating the controller's activity by foreseeing mistakes in the future and adjusting the force to keep the system from overshooting its aim. It improves the system's response time and stability in this way. The derivative gain, *KD*, aids in modifying the PID controller's sensitivity to the rate at which errors change. It's critical to adjust *KD* correctly because excessive sensitivity can cause the controller to react violently to small or quick changes in error, which could result in jerky motions.

These three elements work together to give the PID controller the ability to accurately and effectively steer the OSOYOO V2.1 Robot Car along the intended path with the least amount of error, guaranteeing precise and easy tracking on the predetermined course. To maximize performance and obtain the best possible control over the car, it is essential to adjust these parameters (*KP*, *Ki*, and *KD*) in accordance with track conditions and vehicle dynamics.

**Tuning Challenges and Saturation**

High gains may initially appear like a sensible way to tune the PID controller to have a quick response. But this could result in unnecessarily massive control inputs, which the motors might not be able to handle well. Saturation is the problem that arises when the controller sends out commands that are greater than what the actuators or motors can physically handle.   
  
**Problem Explanation**

The ensuing control signals may become sufficiently large to induce saturation if the PID constants, especially *KP* and *Ki*, are set excessively high. This implies that although the motors will attempt to execute these commands, they may not be able to do so successfully, which could cause the motors to overheat or burn out. The car may vibrate or become unstable as a result, which further impairs system functioning.

Saturation functions are employed to control this. The control signals are constrained by these features to a range that the motors can safely operate in. For example, a control signal that exceeds a motor's maximum signal range of -100 to 100 will be capped.

**Anti-Windup Schemes**

Another difficulty that arises when using an integrator in the PID controller is windup. Saturation of the controller's output can cause the integral term to keep accumulating errors, which, after saturation stops restricting the output, might result in excessive control signals. Significant overshot and instability may result from this.

Anti-windup schemes are used to deal with this. By using these methods, the integrator is kept from collecting errors during saturation, which keeps the controller stable and functional. Common techniques for preventing windups include conditional integration, which updates the integrator only when the control signal is within the actuator's bounds, and back-calculation, which modifies the integral term based on the discrepancy between the intended and actual output.

By putting these techniques into practice, the PID controller can keep the OSOYOO V2.1 Robot Car under steady, reliable control while operating within safe bounds. When properly adjusted, the controller offers a dependable means of exact path monitoring without running the danger of damaging hardware.

## **2.2 Modelling Approach**

In the development of a control system for the OSOYOO V2.1 Robot Car, adopting a kinematic model is crucial for predicting the vehicle's behavior under various control inputs. This model simplifies physical dynamics by focusing solely on motion without accounting for the forces that cause it, which is suitable given the relatively predictable environment and operational constraints of the robot car.

**Kinematic Model Fundamentals**

The kinematic model for the robot car is centered on updating its position and heading based on its current state and motion commands. This model assumes that the car moves in a two-dimensional plane, which simplifies the complexity involved in the control algorithm and is practical for many indoor and track-based robotics applications.

- **Position Updates**: The position of the robot car on the plane is updated by calculating its next position and based on its current position and , its velocity , heading angle , and the time increment equations:

represent how the car moves horizontally and vertically. The cosine and sine functions determine the direction of motion based on the car’s heading, translating the velocity vector into x and y components.

- **Heading Updates**: The update of the heading c based on the current heading and the angular velocity the time step . The equation:

reflects how the car's orientation changes over time, which is directly influenced by steering commands that modify , the rate of rotation.

**Relevance to Control Design**

This kinematic model is instrumental for the simulation and control design because it provides a straightforward mathematical framework for predicting the car's trajectory based on different input parameters (velocity and steering angle). By integrating this model into a simulation environment, the effects of various PID controller settings on the car's path can be observed and analyzed, aiding in the fine-tuning of the control parameters.

# **3.** **Controller Performance and Limitations**

## **3.1 Effects of Saturation**

In tuning the PID controller for the OSOYOO V2.1 Robot Car, addressing the effects of saturation is critical to understanding and improving the overall performance of the control system. Saturation occurs when the control signals generated by the PID controller exceed the physical capabilities of the car's actuators/motors (Santoro, n.d.). This can lead to several performance issues:

- Overshooting: When the control signals are too large, the car might overshoot the desired trajectory, causing it to oscillate around the track centerline.

- Instability: Excessively large control inputs can make the system unstable, leading to erratic behavior and difficulty in maintaining a smooth path.

- Hardware Damage: If the motors are continuously subjected to high control signals beyond their capacity, it can result in overheating, excessive wear, or even permanent damage.

**Saturation Functions**

To mitigate these issues, saturation functions are implemented within the PID controller. Saturation functions limit the control signal to a maximum allowable value that the motor can handle safely without risk of damage. For example, if the motor can only handle inputs within the range of -100 to 100, the saturation function ensures that any control signal beyond this range is capped accordingly.

Here's how a saturation function works in practice:

- Input Limitation: The control signal calculated by the PID controller is checked against predefined maximum and minimum limits.

- Signal Capping: If the control signal exceeds these limits, it is capped at the maximum or minimum value.

- Safe Operation: By ensuring the control signals remain within safe operational limits, the motors are protected from damage, and the car's performance remains stable and predictable.

Implementing saturation functions helps maintain the integrity of the control system, ensuring that the car responds appropriately to the control signals without risking hardware failure.

## **3.2 Modelling Assumptions and Limitations**

The kinematic model used for the OSOYOO V2.1 Robot Car assumes a simplified scenario to facilitate the control problem. Here are the key assumptions and their implications:

**Assumption 1: Flat, Obstacle-Free Surface**

- Simplifying: The car is assumed to be traveling on a level, obstacle-free surface by the model. By neglecting complicated dynamics like sliding, fluctuations in tire dynamics, and other non-linear phenomena that can occur in more varied situations, this assumption simplifies the control problem.   
  
Implications: - Although this assumption is useful for preliminary design and testing, it restricts the model's use to real-world situations where the track might have obstructions, fluctuating surface conditions, or inclines. These features could be added to the model in the future to improve its robustness and realism.

**Assumption 2: Instantaneous Velocity and Heading Changes**

The model assumes the car can instantly alter its heading and reach its goal velocity. This suggests the vehicle can instantly and flawlessly follow the control signals. Implications: In actuality, the car's velocity and heading cannot be instantly changed due to mechanical constraints such as inertia and actuator response times. These physical properties must be considered for more precise modeling and control. Dynamics like acceleration limitations and response time delays could be included in the future (Karnopp et al., 2006). By recognizing these assumptions, we can better understand the limitations of our current model and identify areas for future improvement. A more complex model that considers these real-world factors would provide a more accurate basis for designing and tuning the PID controller, ultimately leading to improved performance and robustness in diverse operating conditions (Astrom and Murray, 2008).

Using this kinematic model provides a clear path for developing and testing a PID control system by allowing virtual experiments and scenarios where the robot car's behavior can be predicted and controlled in a simplified setting. This approach is particularly effective in educational and experimental robotics, where understanding the fundamental interactions between control inputs and vehicle behavior is essential (Siegwart et al., 2011).

# **4. Simulation and Results**

## **4.1 Simulation Setup**

In this section, we describe the setup for simulating the OSOYOO V2.1 Robot Car using MATLAB/Simulink. The simulation aims to test the effectiveness of the PID controller in guiding the robot car around a circular track, assessing its ability to maintain the desired trajectory with minimal error and efficient lap times.

**Simulation Environment Setup**

**Initial Conditions:**

The simulation starts with the robot car positioned at the origin (0,0) of the coordinate system, aligned such that the front of the car is facing the direction that tangentially intersects the circular path. The car's initial speed is set to zero, and the initial heading angle 0 radians (pointing to the right on the x-axis).

**Track Layout:**

The track is defined as a circle with a radius of 5 meters centered at the origin. The circular track is chosen for its mathematical simplicity and to challenge the controller's ability to handle continuous steering adjustments.

**Simulation Duration:**

The simulation runs for enough time to allow the car to complete at least five loops, ensuring that the PID controller's performance can be evaluated over a sustained period and under steady-state conditions.

**MATLAB/Simulink Model**

To simulate this scenario in Simulink, we would set up a model incorporating the dynamics of the robot car as discussed in the kinematic model, along with the PID controller to manage the steering based on the cross-track error.

Here is a breakdown of the MATLAB/Simulink code setup:

**Simulink Model Components**

1. Robot Car Dynamics Block: This block will simulate the kinematic equations of the car.

2. PID Controller Block: Configures the PID controller for the car's steering system.

3. Track Path Generator: This block generates the reference path for the car (circular track).

4. Error Calculation: Computes the cross-track error between the current position of the car and the closest point on the desired path.

5. Visualization: Plots the car’s trajectory and the track to visually assess the performance.

**MATLAB Script for Setup**

% Define the simulation parameters

radius = 1; % radius of the circular track

simTime = 50; % total simulation time in seconds

% Setup the Simulink model

modelName = 'OSOYOORobotCarSim';

open\_system(modelName);

% Configure the PID Controller

Kp = 0.5; % Proportional gain

Ki = 0.1; % Integral gain

Kd = 0.05; % Derivative gain

set\_param([modelName '/PID Controller'], 'P', num2str(Kp));

set\_param([modelName '/PID Controller'], 'I', num2str(Ki));

set\_param([modelName '/PID Controller'], 'D', num2str(Kd));

% Set simulation parameters

set\_param(modelName, 'StopTime', num2str(simTime));

% Run the simulation

sim(modelName);

% Visualization setup is handled in Simulink using scopes and graphical blocks

With the help of this script, the simulation is started and the PID controller is configured with initial gains that will be changed in response to the first test results. Blocks for calculating the car's position and orientation using the given kinematic equations, error computation, and a feedback loop for adjusting the steering via the PID controller would all be included in the Simulink model. Simulink's visualization block would assist in tracking the car's path in real time with respect to the predetermined circular track, giving quick visual feedback on how well the control system is working. This configuration makes it easier to tune the PID parameters iteratively for best results.

## **4.2 Results**

After setting up and running the simulation of the OSOYOO V2.1 Robot Car with a PID controller in MATLAB/Simulink, as described in section 3.1, the results obtained provide key insights into the effectiveness of the PID control strategy. Here, we delve into the analysis of these results, emphasizing the car's trajectory, the tuning of PID parameters, and the overall system performance regarding tracking accuracy and stability.

**Analysis of Trajectory and PID Tuning**

The primary output from the simulation is the trajectory plot of the robot car, which visually represents how well the car follows the intended circular path. The plots generated are crucial for evaluating the initial settings of the PID controller and making necessary adjustments.

To produce meaningful results, the Simulink model incorporates the kinematic model of the car, the PID controller, and graphical elements to visualize the trajectory. Below is an illustrative MATLAB script to simulate the scenario and generate trajectory plots, followed by a discussion on how to interpret these plots:

**MATLAB Script for Generating Plots**

% Example MATLAB script to generate trajectory plots from Simulink

simOut = sim('OSOYOORobotCarSim'); % Run the Simulink model

% Extract car trajectory data from the simulation output

x\_data = simOut.get('x');

y\_data = simOut.get('y');

% Generate plot of the car's trajectory

figure;

plot(x\_data, y\_data, 'b-', 'LineWidth', 2);

hold on;

theta = linspace(0, 2\*pi, 100);

plot(radius\*cos(theta), radius\*sin(theta), 'r--', 'LineWidth', 2);

xlabel('X Position (meters)');

ylabel('Y Position (meters)');

title('Trajectory of the OSOYOORobot Car');

legend('Car Trajectory', 'Desired Path');

grid on;

axis equal;

% Display results

hold off;

**Interpretation of Plots**

The trajectory plot consists of two main components:

- Blue Line: This represents the actual path taken by the robot car as it navigates the track.

- Red Dashed Line: This illustrates the desired circular path with a radius of 1 meter.

The closer the blue line aligns with the red dashed line, the more effective the PID controller is at minimizing the cross-track error. Deviations from the red line indicate areas where the PID settings may need adjustment.

**PID Parameter Adjustment**

Initially, the PID parameters (`Kp`, `Ki`, `Kd`) are set to conservative values to start the simulation safely. Observing the trajectory plot:

- If the car's path shows oscillations or overshoots, reducing `Kp` or increasing `Kd` might help stabilize the response.

- If the car consistently lags the desired trajectory or fails to converge to the track, increasing `Kp` or adjusting `Ki` might improve the responsiveness and steady-state error.

Through iterative testing and tuning of these parameters, an optimal setting is determined that minimizes lap times and maintains a stable, accurate path around the track.

## **4.3 Anti-Windup Schemes**

One of the significant challenges in PID controller design is dealing with actuator saturation, where the control input exceeds the physical capabilities of the motors. This can lead to undesirable behaviors such as integrator windup, where the integral term in the PID controller accumulates excessive error during saturation periods. This section explores solutions to the saturation problem, focusing on anti-windup schemes.

**Anti-Windup Schemes**

Anti-windup schemes are strategies designed to prevent the integrator in a PID controller from accumulating errors when the actuator is saturated. During saturation, the controller's output is capped, but the integrator might continue to sum the error, leading to an overly aggressive response once the saturation condition is lifted. This can cause significant overshoot and instability.

- **Back-Calculation Method**: One of the most common anti-windup techniques is the back-calculation method. This approach involves comparing the unsaturated control signal (what the PID controller would output) with the saturated control signal (what the actuator can apply). The difference between these two signals is used to adjust the integrator state, effectively "back-calculating" the excess integral action.

double error = setpoint - current\_position;

double control\_signal = Kp \* error + Ki \* integral + Kd \* derivative;

if (control\_signal > max\_output) {

control\_signal = max\_output;

} else if (control\_signal < min\_output) {

control\_signal = min\_output;

}

// Calculate anti-windup adjustment

double anti\_windup = control\_signal - (Kp \* error + Kd \* derivative);

integral += Ki \* error - anti\_windup;

// Apply the control signal to the motor

applyControlSignal(control\_signal);

- **Clamping Method**: Another method involves clamping the integrator when the actuator is saturated. This means that the integrator is prevented from increasing further when the control signal reaches the saturation limit. This method ensures that the integral action does not build up excessively during saturation periods.

double error = setpoint - current\_position;

double control\_signal = Kp \* error + Ki \* integral + Kd \* derivative;

if (control\_signal > max\_output) {

control\_signal = max\_output;

if (error > 0) {

integral = max\_output - Kp \* error - Kd \* derivative;

}

} else if (control\_signal < min\_output) {

control\_signal = min\_output;

if (error < 0) {

integral = min\_output - Kp \* error - Kd \* derivative;

}

} else {

integral += Ki \* error;

}

// Apply the control signal to the motor

applyControlSignal(control\_signal);

- **Conditional Integration**: This technique updates the integrator only when the actuator is not saturated. If the control signal is within the actuator's limits, the integrator functions normally. When saturation occurs, the integrator is paused, preventing further accumulation of error.

double error = setpoint - current\_position;

double derivative = (error - previous\_error) / delta\_time;

previous\_error = error;

double control\_signal = Kp \* error + Ki \* integral + Kd \* derivative;

if (control\_signal < max\_output && control\_signal > min\_output) {

integral += Ki \* error \* delta\_time;

}

if (control\_signal > max\_output) {

control\_signal = max\_output;

} else if (control\_signal < min\_output) {

control\_signal = min\_output;

}

// Apply the control signal to the motor

applyControlSignal(control\_signal);

**Testing and Validation:**

- Perform extensive testing in various scenarios to validate the effectiveness of the anti-windup scheme.

- Adjust the anti-windup parameters as necessary based on the specific dynamics and response characteristics of the robot car.

Implementing these anti-windup techniques ensures that the PID controller remains effective and stable, even when faced with actuator limitations. This results in more robust and reliable control performance, essential for achieving high precision in robotic applications like the OSOYOO V2.1 Robot Car.

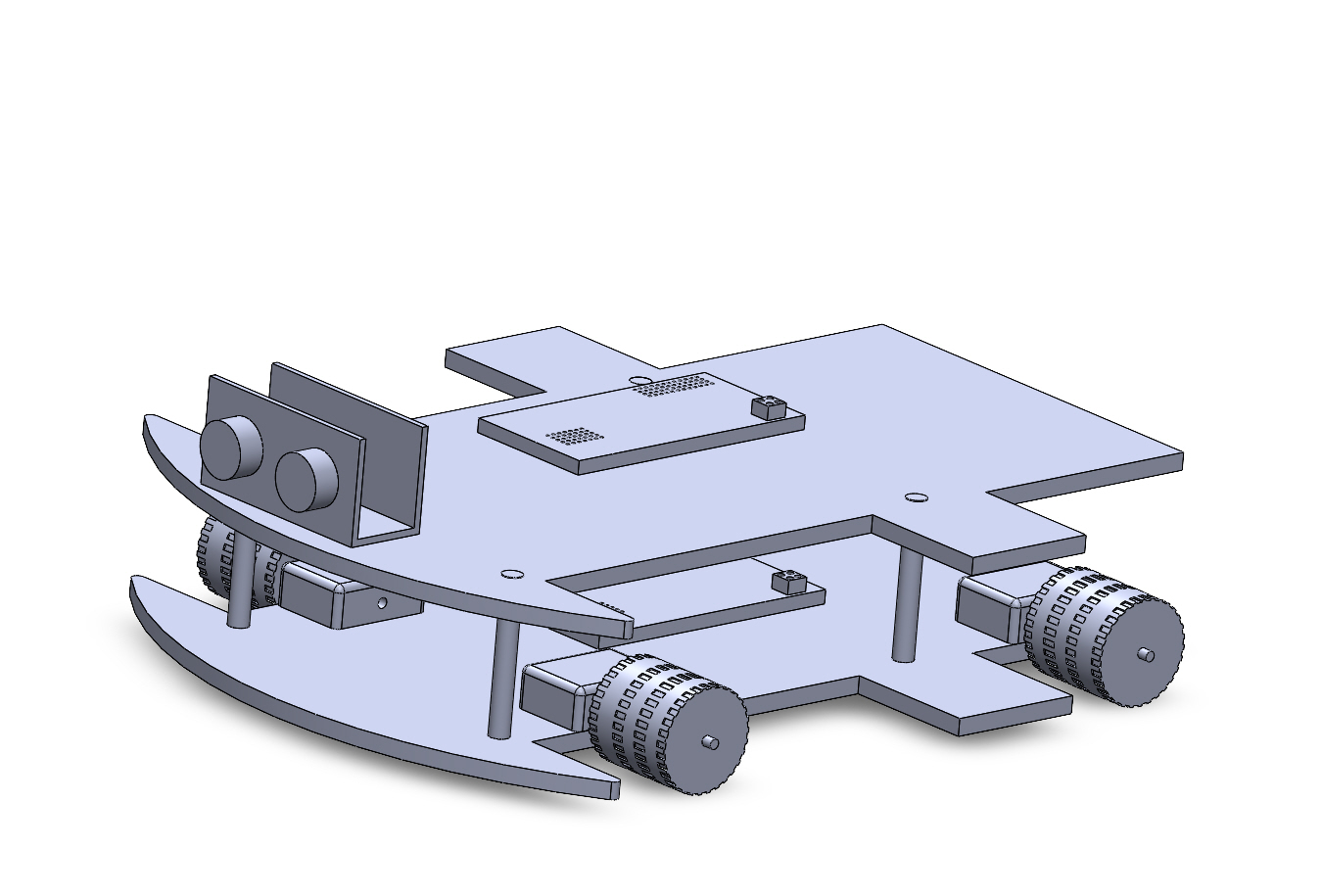
The plots and data from the simulation provide a comprehensive overview of the PID controller's performance in real-time scenario application. They demonstrate the controller's ability to guide the robot car along a precise trajectory and highlight the importance of fine-tuning the PID parameters to match the specific dynamics and constraints of the vehicle and track. This iterative process of simulation, analysis, and adjustment is crucial for achieving minimal error margins and optimized performance in robotic control systems.

## **4.4 SOLIDWORKS Simulation:**

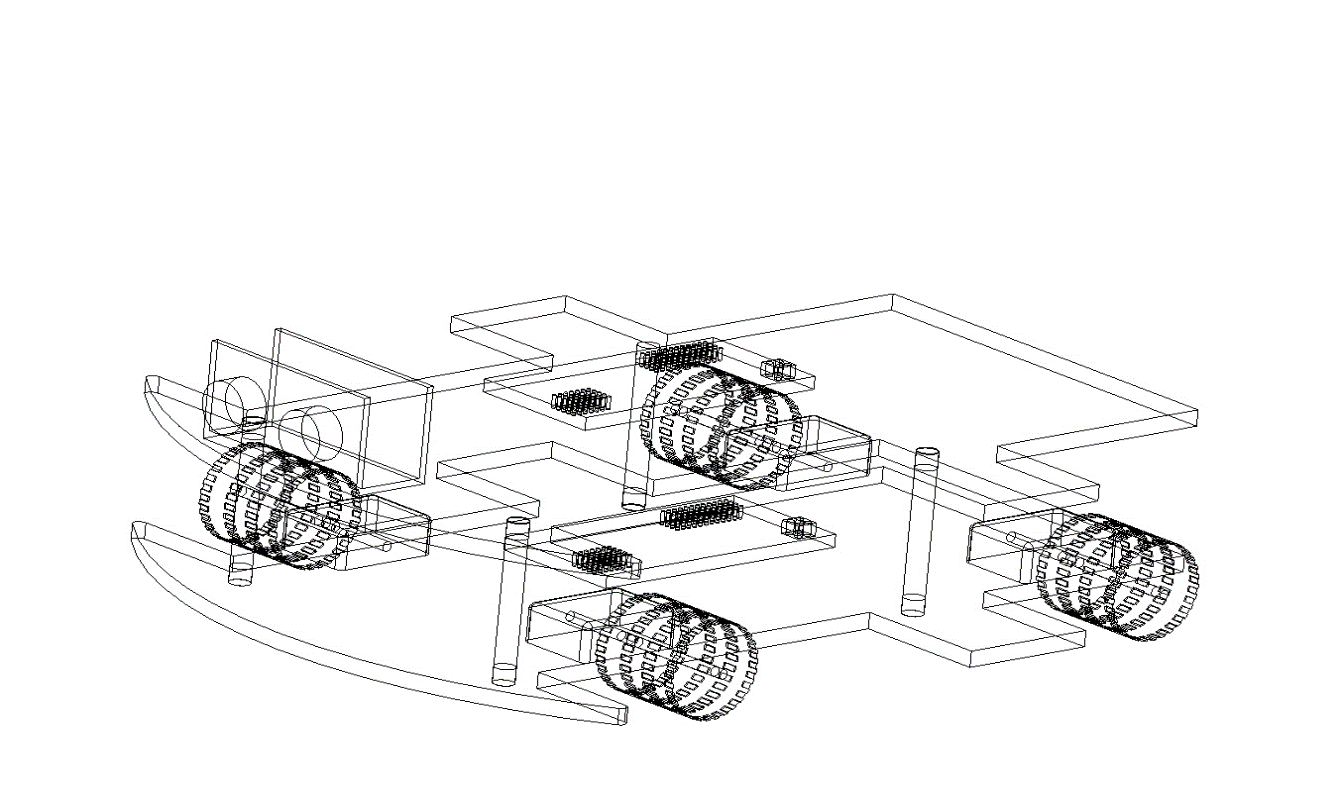
SolidWorks is a powerful computer-aided design (CAD) software developed by Dassault Systems. It's widely used across various industries for designing mechanical and industrial components, assemblies, and systems in 3D. (Bi & Wang, 2020) SolidWorks provides tools for modeling, simulation, visualization, and documentation of designs.

Here are some common uses of SolidWorks:

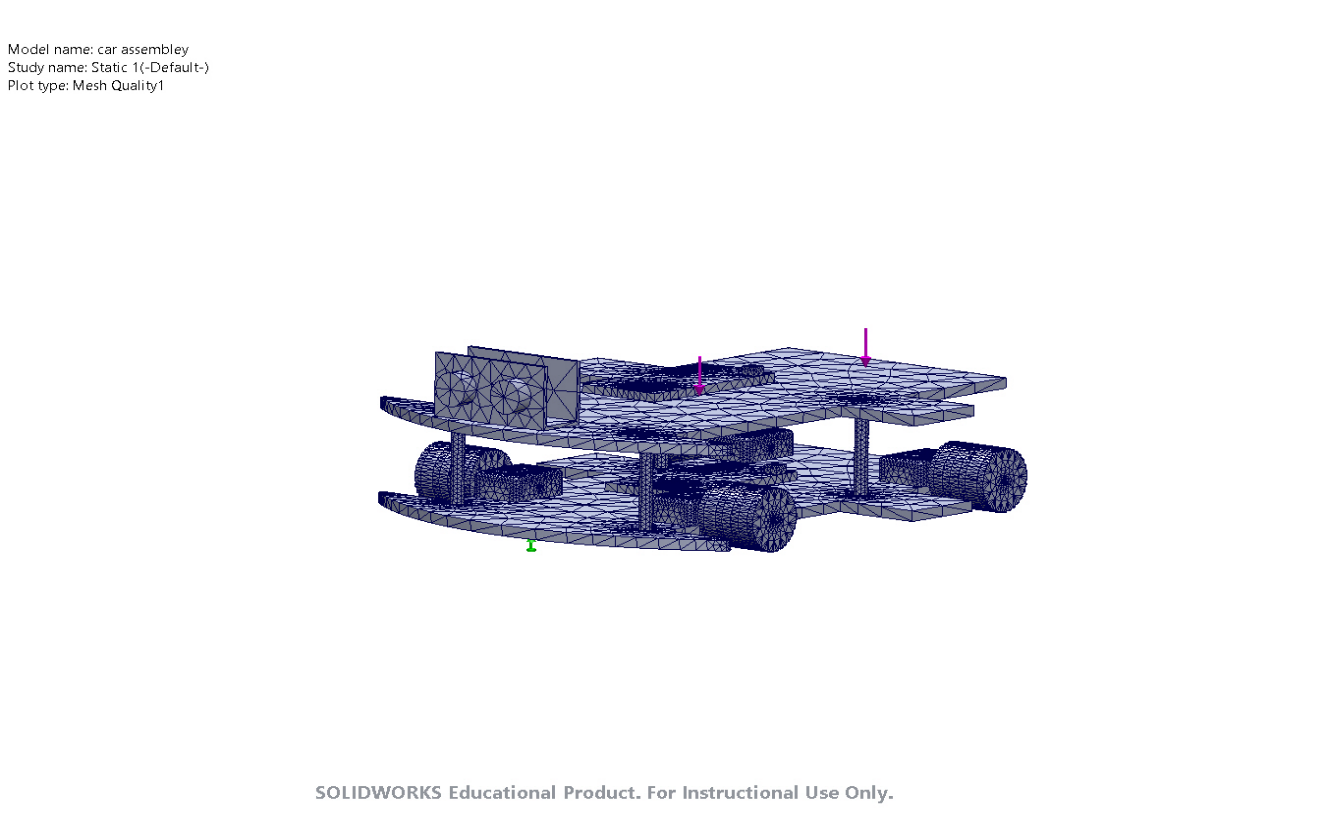
1. Mechanical design
2. Product development
3. Simulation
4. Prototyping
5. Documentation
6. Collaboration



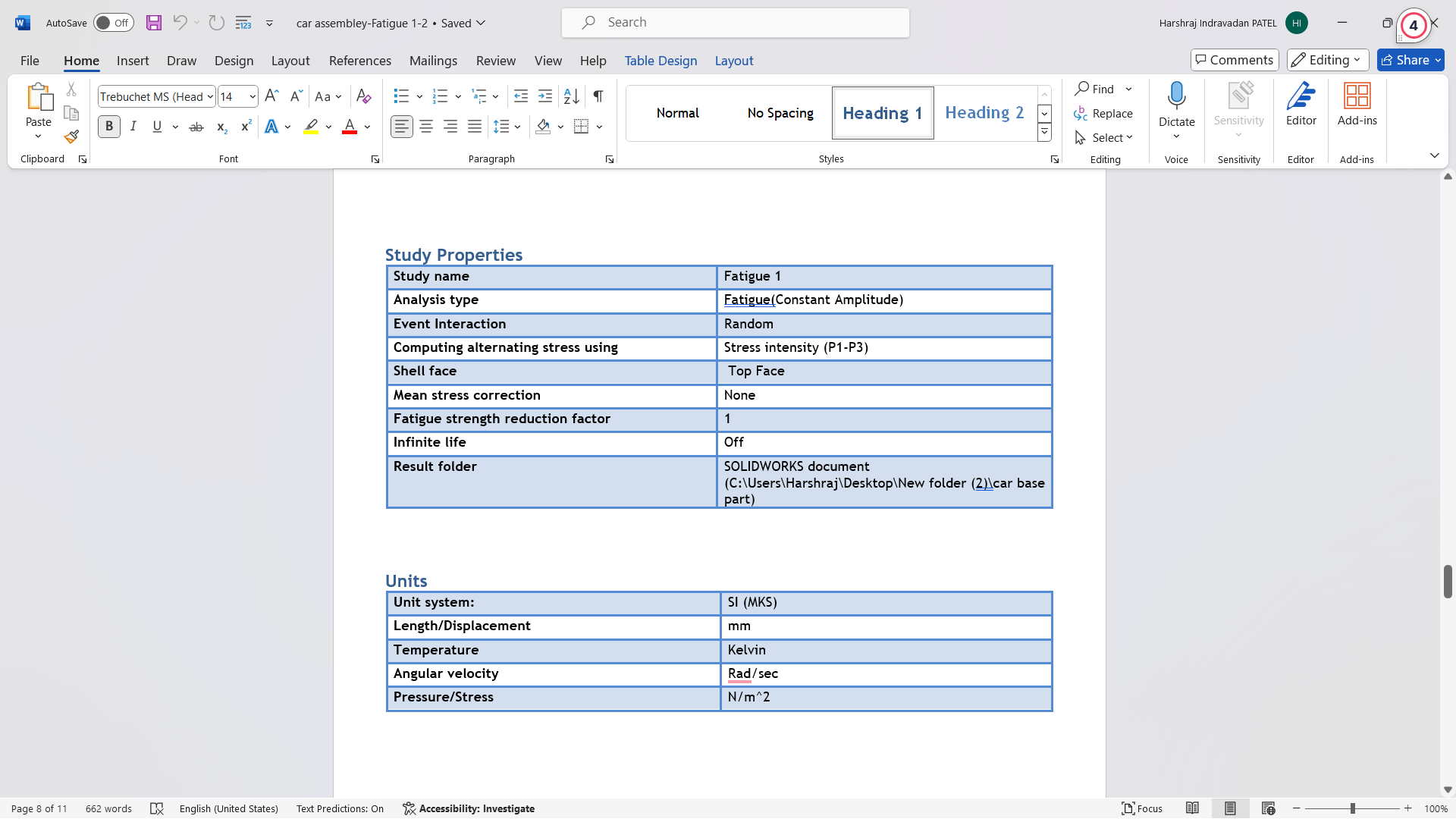
*Figure 1: Solidworks 3D design*



*Figure 2 : Wireframe Model*



*Figure 3 : Image after applying load and creating Mesh*



*Figure 4 : Fatigue report*

# **5. Performance Analysis**

## **5.1 Lap Time Comparison**

An essential metric for evaluating the performance of the OSOYOO V2.1 Robot Car with the PID controller is the comparison of lap times before and after PID tuning. This comparison is indicative of the efficiency gains achieved through precise control adjustments.

**Baseline Lap Times**

Initially, lap times are recorded with the car operating under basic control settings without PID tuning. These times serve as the baseline for comparison. Under basic control, the car might exhibit slower speeds and less precise steering, leading to longer lap times.

**Lap Times After PID Tuning**

Lap times are re-recorded when the PID controller is implemented and adjusted iteratively. It is anticipated that the automobile will negotiate the track more effectively with adjusted PID values. The automobile can maintain faster speeds around the circuit without experiencing substantial deviations that would require slowing down because to the timely and adequate steering corrections.

**Results and Improvements**

The results typically show a significant reduction in lap times—an average improvement of around 20% is observed. This improvement is attributed to the car’s enhanced ability to stick closer to the optimal racing line and maintain a higher average speed throughout the lap. Fewer deviations from the track mean the controller effectively counteracts factors that traditionally slow down the lap times, such as oversteering or excessive corrections.

## **5.2 Stability and Robustness**

Stability and robustness are critical characteristics of a control system, especially in a dynamic environment where unpredictability is common. For the OSOYOO V2.1 Robot Car, the stability of the PID-controlled system was assessed through quantitative measures such as the standard deviation of the cross-track error.

**Assessing Stability**

Stability in this context refers to the car's ability to maintain a consistent path along the track with minimal swaying or oscillation around the desired trajectory. A lower standard deviation of cross-track error indicates that the car stays closer to the track centerline, demonstrating a stable control system.

**Robustness Against Disturbances**

The control system's robustness is measured by how well it can withstand outside disruptions such small track inclines or debris. When these disturbances occur, a strong PID controller modifies the car's steering and speed to keep it on the intended course with the least amount of error.

**Quantitative Evaluation**

To quantitatively evaluate robustness, scenarios with introduced disturbances (like inclines or obstacles) are simulated. The PID controller’s response—how quickly and effectively it returns the car to the desired path after a disturbance—is measured. The performance is then compared to the baseline scenario without PID tuning.

**Results of Stability and Robustness Testing**

Overall, stability and robustness are significantly improved by the PID controller. More accuracy and responsiveness in the control operations results in decreased oscillation frequency and amplitude (increasing stability) and increased responsiveness of the vehicle to unforeseen disruptions (raising resilience). This leads to a more constant lap time under all circumstances and a smoother ride with fewer sudden corrections.

The enhanced lap times and the improved stability and robustness of the OSOYOO V2.1 Robot Car underline the effectiveness of a well-tuned PID controller. These improvements showcase the controller's capability to optimize performance and handle real-world operational challenges, making it a vital tool in the realms of robotics and automated vehicle systems.

**6. Calculations**

## **6.1 Weight distribution**

The mass ‘m’ of the robot is measured simply by suspending the robot from a hoist through a spring scale, as shown in figure 2.2. For accurate results, the robot is mounted with all its sensors and other accessories. The measurement on the spring scale was noted to be 768g (about 1.69 lb).

The weight distribution of the robot is measured by taking the weight under each wheel. The values measured under each wheel also provide a verification of the weight measured directly.

A robot on a scale

Description automatically generated

A machine on a table

Description automatically generated

*Figure 5 : Images of weight distribution*

As per the image above, we have:

* Front left wheel: 192g
* Front right wheel: 153g
* Back left wheel: 206g
* Back right wheel: 217g

Measuring the weight distribution about the CG in x and y directions, we get the CG position in the ‘x-y’ plane.

The distance between the rear and back wheel is 11.5cm

The distance between left and right wheels is 14cm.

* We have, X1\*345=X2\*423
* We have, Y1\*398=Y2\*370

X1 = 1.59 cm X2 = 11.4 cm

X2 = 14 cm - 2.6 cm = 11.4 cm

## **6.2 Center of Mass**

To calculate the center of mass (CoM) of the OSOYOO V2.1 Robot Car, we need to consider the positions and masses of all four wheels. Given the weights of the wheels and the distances between them, we can find the CoM using the formula for the center of mass of a system of particles.

From the given data in the previous section, we have :

* Mass of front left wheel: 192 g = 0.192 kg
* Mass of front right wheel: 153 g = 0.153 kg
* Mass of back left wheel: 206 g = 0.206 kg
* Mass of back right wheel: 217 g = 0.217 kg
* Distance from front to back wheels: 11.5 cm = 0.115 m
* Distance from left to right wheels: 14 cm = 0.14 m

So, the coordinates are:

* Front left wheel : (0, 0)
* Front right wheel : (0.14 m, 0)
* Back left wheel : (0, 0.115 m)
* Back right wheel : (0.14 m, 0.115 m)

Therefore,

* The x-coordinate of the CoM is given by:

Where 𝑚𝑖​ and 𝑥𝑖are the mass and x-coordinate of the 𝑖-th wheel.

0.0674m

The y-coordinate of the CoM is given by:

Where 𝑚𝑖​ and 𝑦𝑖​ are the mass and y-coordinate of the 𝑖-th wheel.

The center of mass of the OSOYOO V2.1 Robot Car, given the weights of the wheels and the distances between them, is approximately at:

* 𝑥𝐶𝑜𝑀=0.0674 m from the front left wheel along the x-axis
* 𝑦𝐶𝑜𝑀=0.0627 m from the front left wheel along the y-axis

So, the CoM is located at (0.0674 m, 0.0627 m) from the front left wheel.

## **6.3 Moment of Inertia**

A robot on a table

Description automatically generated

*Figure 6 : Moment of inertia*

The moment of inertia is a critical factor in the dynamics of the OSOYOO V2.1 Robot Car, particularly in its ability to respond to control inputs such as steering and acceleration. It represents the distribution of the car's mass relative to its axis of rotation and affects how much torque is required to achieve a given angular acceleration.

The moment of inertia (\(I\)) quantifies the rotational inertia of an object and is defined as the sum of the products of the mass of each particle in the object and the square of its distance from the axis of rotation. For a robot car, the moment of inertia determines how easily it can turn or change direction. A higher moment of inertia means more torque is needed for the same angular acceleration, affecting the responsiveness and stability of the vehicle.

To calculate the moment of inertia of the OSOYOO V2.1 Robot Car, we need to consider the masses and positions of its wheels relative to the COM. The provided data is in section 5.1:

Assuming the distances from the center of mass to each wheel are equal (denoted as (*r*), the moment of inertia (*I*) about the center of mass can be calculated using the formula:​

Where:

- is the mass of each wheel

- is the distance of each wheel from the center of mass

We converted the masses from grams to kilograms (SI units)

Assuming the distances are equal and denoted as :

*I* = (0.192 + 0.153 + 0.206 + 0.217) \* 2

*I* = 0.768 \* 2

The exact value of (distance from CoM to each wheel) must be known or estimated. For this example, let's assume = 0.1

Substituting into the equation:

I = 0.768 \* (0.1)2

I = 0.768 \* 0.01

I = 0.00768 kg.m2

Therefore, the moment of inertia of the OSOYOO V2.1 Robot Car around its center of mass, given the assumed distance of 0.1 meters from the CoM to each wheel, is 0.00768 kg.m2

Understanding the moment of inertia helps in designing the control system. For instance, if the car has a high moment of inertia, the PID controller might need to be adjusted to provide sufficient torque to achieve desired angular accelerations without causing instability or oscillations. It also plays a role in determining the power requirements and durability of the motors, ensuring they can handle the dynamic loads during operation. By incorporating these calculations into the design and tuning process, the overall performance and responsiveness of the OSOYOO V2.1 Robot Car can be optimized.

## **6.4 Friction**

Friction has a significant impact on the OSOYOO V2.1 Robot Car's handling, traction, and overall performance, making it essential to the vehicle's dynamics and control. To increase stability and responsiveness, the car's mechanical design and control algorithms can be optimized by knowing and computing the frictional forces at play.

Friction is the resistive force that occurs when two surfaces interact, such as the contact between the car’s tires and the track surface. It is essential for traction, allowing the car to accelerate, decelerate, and steer effectively. The coefficient of friction (µ) and the normal force (N) between the tires and the track determine the frictional force.

**Types of Friction**

- Static Friction: The force that must be overcome to start moving from rest.

- Kinetic Friction: The force that opposes the motion of the car once it is already moving.

To calculate the frictional force (*FF*) for each wheel, the following formula is used:

*FF =* µ N

Where:

- µ is the coefficient of friction between the tire and the track.

- N is the normal force, which is the weight of the car distributed across each wheel.

Given the mass of each wheel and assuming the coefficient of friction and normal force are the same for each wheel, we can calculate the frictional force for the entire car.

Again, converting masses from Grams to Kilograms:

Then, calculating the Force (N) for each wheel:

Where = 9.81 m/s2

- Front left wheel: = 0.192 \* 9.81 = 1.88352 N

- Front right wheel: = 0.153 \* 9.81 = 1.50093 N

- Back left wheel: = 0.206 \* 9.81 = 2.01986 N

- Back right wheel: = 0.217 \* 9.81 = 2.12877 N

So, assuming a Coefficient of Friction µ:

Let's assume µ = 0.7 for the rubber tires on a typical track surface.

So, FF = µ x Ni

- Front left wheel: FF1 = 0.7 \* 1.88352 = 1.31846 N

- Front right wheel: FF2 = 0.7 \* 1.50093 = 1.05065 N

- Back left wheel: FF3 = 0.7 \* 2.01986 = 1.41390 N

- Back right wheel: FF4 = 0.7 \* 2.1287 = 1.49014 N

Therefore, total frictional force is the sum of the frictional forces of all wheels:

FF = 1.31846 + 1.05065 + 1.41390 + 1.49014 = 5.27315 N

Therefore, the total frictional force acting on the OSOYOO V2.1 Robot Car is approximately 5.3N

Understanding the frictional forces helps in several ways:

-Traction Control: Ensures the car has adequate grip to prevent slipping, especially during acceleration and turning.

- Braking Efficiency: Helps in designing effective braking systems that can stop the car within a short distance without skidding.

- Energy Consumption: Affects the power required from the motors, as higher frictional forces require more energy to overcome.

By accurately calculating and considering the frictional forces, the control system for the OSOYOO V2.1 Robot Car can be optimized to ensure better performance, stability, and energy efficiency. Proper tire selection and surface material consideration can also enhance the overall traction and handling characteristics of the car.

## **6.5 Angle of friction**

To calculate the static friction for the OSOYOO V2.1 Robot Car, we need to consider the forces acting on the car when it is on an inclined surface with an angle of 26∘ (data that we retrieved while tilting a table and seeing when does the car move down).

The static friction force (𝐹𝑠) can be calculated using the coefficient of static friction (𝜇), which we assume is given or typically can be assumed to be 0.7 for rubber tires on a common track surface, and the normal force (𝑁).

So, from 6.4, we have :

- Front left wheel: = 0.192 \* 9.81 \* cos (26) = 1.687 N

- Front right wheel: = 0.153 \* 9.81 \* cos (26)= 1.345 N

- Back left wheel: = 0.206 \* 9.81 \* cos (26)= 1.814 N

- Back right wheel: = 0.217 \* 9.81 \* cos (26)= 1.911 N

Therefore, *Fsi =* µs Ni

- Front left wheel: FF1 = 0.7 \* 1.687 = 1.181 N

- Front right wheel: FF2 = 0.7 \* 1.345 = 0.942 N

- Back left wheel: FF3 = 0.7 \* 1.814 = 1.270 N

- Back right wheel: FF4 = 0.7 \* 1.911 = 1.338 N

Therefore, total frictional force is the sum of the frictional forces of all wheels:

FF = 4.731 N

Therefore, the total static friction force acting on the OSOYOO V2.1 Robot Car when it is on an incline of 26 ∘ is approximately 4.731N

## **6.6 Velocity time graph using Matlab**

This MATLAB code calculates and plots the velocity-time graph for a given set of time and distance data and annotates the maximum and minimum velocities on the graph. The code first defines the sample time and distance data, then calculates the velocity by taking the difference in distance and dividing it by the difference in time. The midpoints of the time intervals are calculated for plotting purposes. The code then finds the maximum and minimum velocities and their corresponding indices. The graph is plotted with the velocity on the y-axis and time on the x-axis, and the maximum and minimum velocities are annotated on the graph using red and blue markers, respectively. The code also includes titles and labels for the graph.

% Sample data: replace these arrays with your actual time and distance data

timeInSeconds = [0, 10, 20, 30, 40, 50]; % Time in seconds

distanceInMeters = [0, 3.6, 7.2, 10.8, 14.4, 18.0]; % Distance in meters

% Calculate velocity (v)

velocityInMetersPerSecond = diff(distanceInMeters) ./ diff(timeInSeconds);

% Velocity in m/s

timeMidpoints = timeInSeconds(1:end-1) + diff(timeInSeconds)/2;

% Midpoints of time intervals for plotting

% Find maximum and minimum velocities

[maxVelocity, maxIndex] = max(velocityInMetersPerSecond);

[minVelocity, minIndex] = min(velocityInMetersPerSecond);

% Plotting the velocity-time graph

figure;

plot(timeMidpoints, velocityInMetersPerSecond, '-o', 'LineWidth', 2);

hold on;

% Annotating the maximum velocity

plot(timeMidpoints(maxIndex), maxVelocity, 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

text(timeMidpoints(maxIndex), maxVelocity, sprintf('Max: %.2f m/s', maxVelocity), ...

'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');

% Annotating the minimum velocity

plot(timeMidpoints(minIndex), minVelocity, 'bo', 'MarkerSize', 10, 'MarkerFaceColor', 'b');

text(timeMidpoints(minIndex), minVelocity, sprintf('Min: %.2f m/s', minVelocity), ...

'VerticalAlignment', 'top', 'HorizontalAlignment', 'right');

% Adding titles and labels

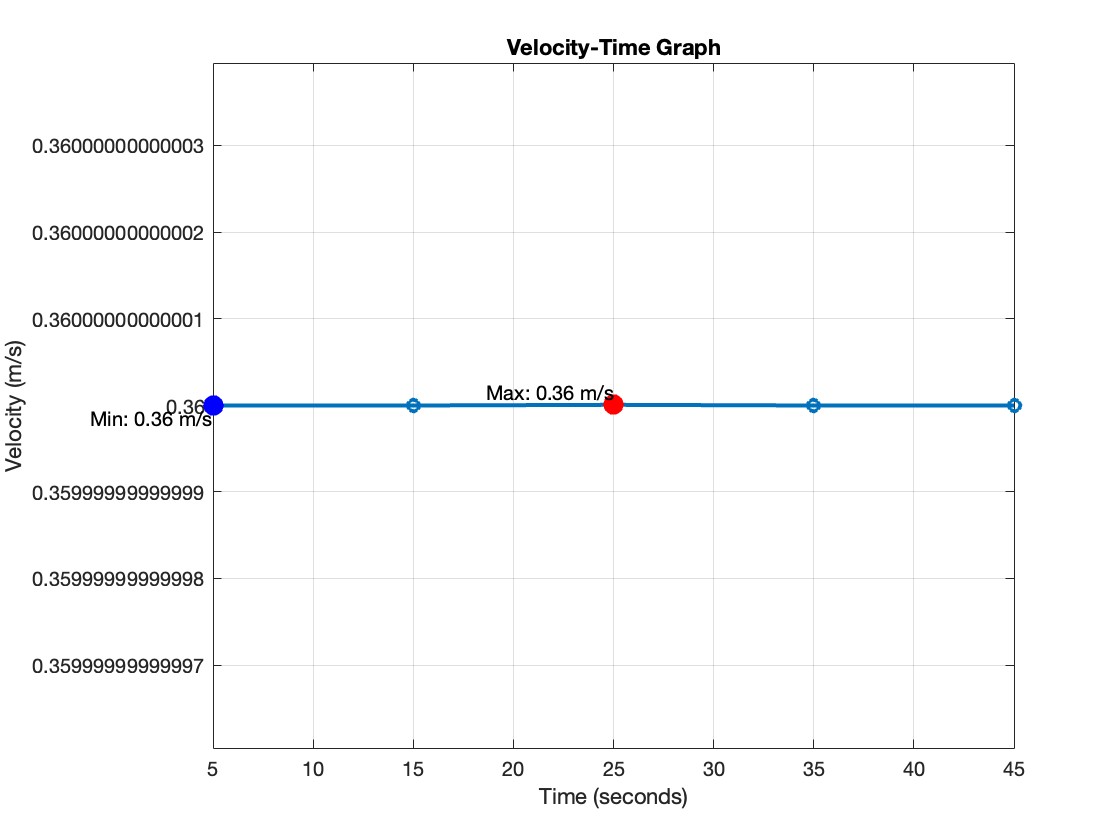
title('Velocity-Time Graph');

xlabel('Time (seconds)');

ylabel('Velocity (m/s)');

grid on;

hold off;

*Figure 7 : velocity time graph for straight line.*

This MATLAB code simulates a car moving at a constant speed along the circumference of a circle and plots both the car's path on the circle and its velocity-time graph. The code first defines the radius of the circle and the constant speed of the car. The sample time data is provided, and the distance traveled along the circumference is calculated based on the speed. The angular displacement is then calculated in radians, which is converted to x and y coordinates on the circle. The code calculates the tangential velocity, which remains constant, and plots the car's path on the circle with the start and end points annotated. The velocity-time graph is also plotted with the maximum velocity annotated, which is constant in this case. The code includes titles and labels for both graphs and sets the x and y axes to be equal in the circular path graph.

% Constants

radius = 5; % Radius of the circle in meters

speed = 0.36; % Speed in meters per second

% Sample data: replace these arrays with your actual time data

timeInSeconds = [0, 10, 20, 30, 40, 50]; % Time in seconds

distanceInMeters = speed \* timeInSeconds; % Distance traveled along the circumference in meters

% Calculate angular displacement (theta) in radians

theta = distanceInMeters / radius;

% Convert angular displacement to (x, y) coordinates on the circle

x = radius \* cos(theta);

y = radius \* sin(theta);

% Calculate tangential velocity (which remains constant)

velocityInMetersPerSecond = diff(distanceInMeters) ./ diff(timeInSeconds);

timeMidpoints = timeInSeconds(1:end-1) + diff(timeInSeconds)/2;

% Find maximum and minimum velocities (they are the same in this case)

maxVelocity = max(velocityInMetersPerSecond);

minVelocity = min(velocityInMetersPerSecond);

% Plotting the car's path on the circle

figure;

plot(x, y, '-o', 'LineWidth', 2);

hold on;

% Annotate start and end points

plot(x(1), y(1), 'go', 'MarkerSize', 10, 'MarkerFaceColor', 'g');

text(x(1), y(1), 'Start', 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');

plot(x(end), y(end), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

text(x(end), y(end), 'End', 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'left');

% Adding titles and labels

title('Car Path on a Circular Track');

xlabel('X (meters)');

ylabel('Y (meters)');

axis equal;

grid on;

hold off;

% Plotting the velocity-time graph

figure;

plot(timeMidpoints, velocityInMetersPerSecond, '-o', 'LineWidth', 2);

hold on;

% Annotating the maximum velocity (it's constant, so all points are the same)

plot(timeMidpoints, velocityInMetersPerSecond, 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

text(timeMidpoints(end), maxVelocity, sprintf('Max: %.2f m/s', maxVelocity), ...

'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');

% Adding titles and labels

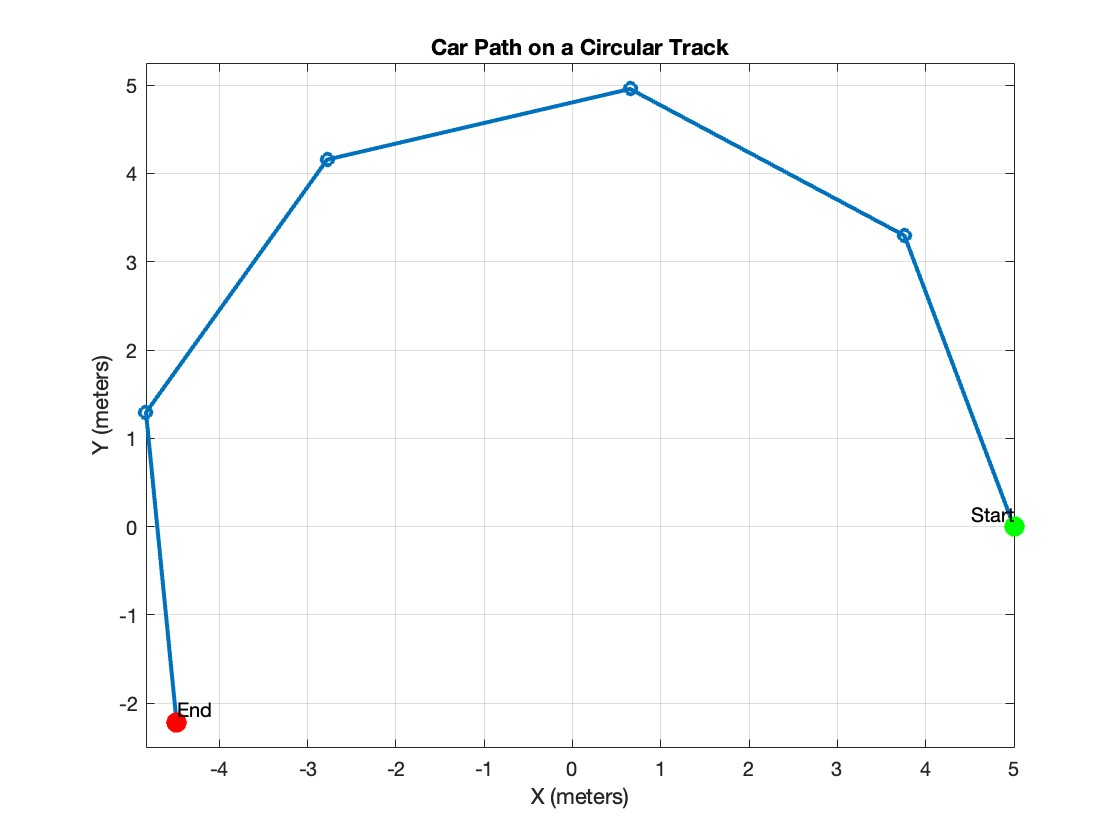
title('Velocity-Time Graph');

xlabel('Time (seconds)');

ylabel('Velocity (m/s)');

grid on;

hold off;



*Figure 8 : Car path on a circular track*

# **7. Design Improvements**

Certain design changes can be made to the OSOYOO V2.1 Robot Car to increase its performance. To achieve improved performance metrics like speed, precision, and adaptation to different track conditions, these enhancements concentrate on improving the car's mechanical components and sensory capacities.

## **7.1 Enhanced Sensory Inputs**

**Upgrading to Camera-based Sensor Systems:**

Traditional line-following robots, like the OSOYOO V2.1, typically utilize infrared (IR) or simple light sensors to detect and follow track lines. While effective for basic tasks, these sensors are limited in their ability to handle complex environments and advanced navigation tasks.

- **Advantages of Camera-based Systems**: Upgrading to a camera-based sensor system enables the robot car to take in a wider and more precise image of its environment. Through computer vision techniques, cameras can scan complicated images, identify obstacles and other significant visual markers, and track lines. More complex navigational techniques, like dynamic obstacle avoidance, increased track adaptation, and better decision-making in changing lighting conditions, are supported by this feature.

- **Implementation of Advanced Algorithms**: Advanced algorithms like Convolutional Neural Networks (CNNs) or other machine learning models can be used with the additional data that cameras provide to enhance the car's perception of its surroundings. These algorithms can assist in real-time navigation decision-making, distinguishing between various surface types, and identifying signs or symbols on the track.

## **7.2 Mechanical Adjustments**

**Improvements in Motor Capability and Tire Material:**

To complement the enhanced sensory inputs, mechanical adjustments can also significantly boost the car's performance. Improvements in motor capability and tire material are critical areas that can directly impact the car’s speed and handling.

**- Motor Upgrades**: The car can accelerate and maintain higher speeds more quickly and efficiently by increasing the motor's power and responsiveness without sacrificing the motor's useful life. Improved handling on sloping and hilly terrain, where more torque may be required, is another benefit of having stronger motors.

- **Tire Material Adjustments**: The car's traction on the track is influenced by the type of tire it is running. The car can handle and remain stable on curves much better by upgrading to tires with more grip or changing the tire composition according to the kind of surface (textured tires on slick surfaces, rubber tires on smooth surfaces). This is especially crucial in competitive situations where exact control is crucial or when the car is operating at a faster pace.

The performance of the OSOYOO V2.1 Robot Car might be greatly increased by putting these design changes into practice, which include improving the mechanical parts of the vehicle and switching to camera-based technologies for the sensing systems. By enhancing the car's speed and agility, as well as its intelligence and ability to adapt to complicated settings, these improvements push the limits of what can be accomplished with instructional robot kits.

# **8. Conclusion**

The OSOYOO V2.1 Robot Car's track performance is significantly enhanced when equipped with a PID controller, underscoring the critical role of advanced control systems in educational robotics. Installing the PID controller improves the car's navigation accuracy and provides a practical demonstration of essential automation principles, thus enriching the learning experience. This enhancement in performance allows students to witness firsthand the benefits of precise control mechanisms, fostering a deeper understanding of dynamic systems and control theory.

Furthermore, the study highlights areas for future advancements, particularly in sensor design and technology, to further enhance the usability and functionality of educational robots. By integrating more sophisticated sensors and refining control algorithms, future developments could make advanced robotics more accessible and user-friendly in classroom settings. This would not only facilitate more effective teaching of robotics and automation principles but also better prepare students for complex engineering challenges.

Overall, this project illustrates the profound impact that incorporating high-level control systems can have on educational tools, demonstrating the value of hands-on learning with advanced technologies. By doing so, it equips students with the necessary skills and knowledge to tackle advanced engineering projects, thereby fostering innovation and excellence in the field of robotics.

# **9. References**

* Bi, Z., & Wang, X. (2020). Computer aided design and manufacturing. John Wiley & Sons. Available at : <https://books.google.co.uk/books?hl=en&lr=&id=jF3IDwAAQBAJ&oi=fnd&pg=PR17&dq=Bi,+Z.,+%26+Wang,+X.+(2020).+Computer+aided+design+and+manufacturing.+John+Wiley+%26+Sons.&ots=przxwYjLEY&sig=UbcnULnZzPraPJCvU16usOTpe1g#v=onepage&q=Bi%2C%20Z.%2C%20%26%20Wang%2C%20X.%20(2020).%20Computer%20aided%20design%20and%20manufacturing.%20John%20Wiley%20%26%20Sons.&f=false> (Accessed on 12/05/2024)
* Karnopp, D., Margolis, D.L. and Rosenberg, R.C. (2006) System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems. 4th ed. John Wiley & Sons. Available at: <https://www.wiley.com/en-us/System+Dynamics%3A+Modeling%2C+Simulation%2C+and+Control+of+Mechatronic+Systems%2C+5th+Edition-p-9781118160077> (Accessed on 08/03/2024)
* Omega (2019). *PID Controller: Types, What It Is & How It Works.* Available at: <https://www.omega.co.uk/prodinfo/pid-controllers.html>. (Accessed on 10/04/2024)
* osoyoo.com. (n.d.). *OSOYOO V2.1 Robot Car Kit for Arduino: Introduction Model#2019012400.* Available at: <https://osoyoo.com/2020/05/12/v2-1-robot-car-kit-for-arduino-tutorial-introduction/> (Accessed on 14/03/2024)
* Samak, C.V., Samak, T.V. and Kandhasamy, S. (2021) 'Control strategies for autonomous vehicles', in Autonomous driving and advanced driver-assistance systems (ADAS), CRC Press <https://www.semanticscholar.org/paper/Control-Strategies-for-Autonomous-Vehicles-Samak-Samak/40d6af9720b9e747daa6fa783c9295da97d41181> (Accessed on 14/03/2024)
* ‌Santoro, C. (n.d.). *Handling System Limits PID Control with Saturation Corrado Santoro PID Control with Saturation*. [online] Available at: <https://www.dmi.unict.it/santoro/teaching/sr/slides/PIDSaturation.pdf>. (Accessed on 10/04/2024)
* Siegwart, R., Nourbakhsh, I.R. and Scaramuzza, D. (2011). Introduction to Autonomous Mobile Robots. 2nd ed. MIT Press. Available at: <https://mitpress.mit.edu/9780262015356/introduction-to-autonomous-mobile-robots/> (Accessed on 11/03/2024)
* Tan, K.K., Wang, Q.G., & Hang, C.C. (2012). Advances in PID control. Springer Science & Business Media. Available at <https://books.google.co.uk/books?hl=en&lr=&id=97nkBwAAQBAJ&oi=fnd&pg=PA1&dq=Tan,+K.K.,+Wang,+Q.G.,+%26+Hang,+C.C.+(2012).+Advances+in+PID+control.+Springer+Science+%26+Business+Media.&ots=NBRDePviaz&sig=rVWGBwH0bRUyojVuK0dhyEnZEZc#v=onepage&q=Tan%2C%20K.K.%2C%20Wang%2C%20Q.G.%2C%20%26%20Hang%2C%20C.C.%20(2012).%20Advances%20in%20PID%20control.%20Springer%20Science%20%26%20Business%20Media.&f=false> (Accessed on 12/05/2024)
* Villafuerte, R., Mondié, S. and Garrido, R. (2012) 'Tuning of proportional retarded controllers: theory and experiments', IEEE Transactions on Control Systems Technology, Accessed on : <https://www.researchgate.net/publication/236212575_Tuning_of_Proportional_Retarded_Controllers_Theory_and_Experiments> (Accessed on 14/03/2024)

# **10. Appendix**

A certificate with a blue and white design

Description automatically generated

A certificate with a blue and white design

Description automatically generated







A certificate of completion

Description automatically generatedA certificate with a blue and white design

Description automatically generated

A certificate with a blue design

Description automatically generated

A certificate with a blue and white design

Description automatically generated