THE HONG KONG POLYTECHNIC UNIVERSITY

BEng Honours Degree in Mechanical Engineering

Design of a Smart Mobile Robot with Active Suspension System for Automated Navigation in an Uneven Terrain

By

Kweon Tae Hyeon

Luo Yi

Sit Yu Hong

Report submitted in partial fulfilment of the requirements for the Honours Degree of Bachelor of Engineering in Mechanical Engineering

April 22, 2024

Acknowledgments

We would like to express our sincere gratitude to Dr. Henry Chu, our Final Year Project supervisor, for giving us continuous support and guidance throughout this project. His valuable insights and encouragement were instrumental in helping us achieve our goals.

I would like to express my sincere gratitude to Mr. Yuen Ka, Technical Officer, for his kind assistance in allowing us to utilize laboratory GH033 for our final year project. His willingness to provide us with access to necessary equipment, such as the multimeter and soldering station, was greatly appreciated.

We are grateful to Mr. Kwan-yau Chan, the Industrial Center staff member in charge of 3D printers, for his assistance with the 3D printing process throughout this project.

Abstract

This project proposes the design of a smart mobile robot equipped with an active suspension system aimed at autonomous navigation in uneven outdoor terrains. The robot aims to maintain high stability while carrying heavy objects and overcome obstacles to reach a specific destination. The robot is equipped with various sensors including a camera, LiDAR, and IMU to provide real-time environment data. The data will then be utilized to adjust the height of the wheel actively and determine the best route for the destination. The mobile platform, suspension mechanism, and sensor fusion will be designed and analyzed. The design criteria and constraints are identified and analyzed using software like SolidWorks and ANSYS. The robot's components are fabricated through sheet metal and 3D printing. With the prototype and performance evaluation through testing, the design and mechanism can be further improved.

The project also includes a study of the active suspension system, a vital component that significantly impacts vehicle stability. While passive suspension system is commonly employed in automobiles to provide comfortable driving on regular roads, active suspension excels passive suspension system in terms of stability, while traversing uneven terrains with obstacles. Our research has involved an exploration of various active suspension system designs and control methods for maintaining optimal stability on uneven terrain. The project attempts to apply a unique design method for the active suspension system in order to achieve autonomous navigation on uneven terrain with high stability.

Table of Contents

1. Introduction

Currently, there is a substantial amount of research focused on suspension systems for automobiles to ensure comfortable rides for drivers. Ride comfort is a crucial factor in the automobile industry for enhancing the overall quality of driving experience [1]. However, when it comes to robotic systems such as mobile robots, the suspension systems are often neglected due to their unmanned nature. Mobile robots are typically designed and optimized for indoor environments with flat and smooth surfaces. These controlled and predictable environments simplify navigation and operation. However, as these robots are increasingly tasked with navigating uneven surfaces with variations in height obstacles like rocks, construction debris, or uneven outdoor terrain, their limitations become evident. Without proper suspension systems, mobile robots may experience reduced stability, sensor data disruption impacting navigation, and increased wear and tear on components due to impacts.

1.1 Problem Identification

One of the main challenges for mobile robots operating on uneven terrain is maintaining stability. The interaction between the wheels and the rough surface can lead to a loss of traction and stability, resulting in slippage and limiting the robot's mobility. This can cause damage to the robot's body and components.

Figure 1. Search and Rescue Mobile robot for disasters [2]

Mobile robots venturing beyond controlled environments can face a diverse range of uneven terrains that challenge their stability and mobility, including outdoor exploration environments with rocky landscapes, slopes, and soft terrains like sand dunes or mud. Construction sites also present challenges with debris piles, trenches, and unevenly compacted soil. These challenging terrains highlight the limitations of traditional rigid chassis designs. For example, mobile robots operating on Mars face significant challenges due to the rough and rocky terrain. The Spirit and

Opportunity rovers, for example, needed to navigate craters, slopes, and loose soil. Search and Rescue robots need to navigate disaster zones with debris, collapsed structures, and uneven surfaces [2]. A traditional rigid chassis would struggle to maintain stability and traction in such conditions, potentially getting stuck or damaging components due to impacts.

Furthermore, traditional suspension systems often have limited shock absorption capabilities. As the robot traverses uneven surfaces, the resulting vibrations are directly transferred to the main body, impacting the entire system's stability. These vibrations can disrupt the functionality of onboard sensors, affecting navigation and object recognition. Additionally, the constant impact can increase wear and tear on the robot's components, reducing its lifespan.

Conventional robots are often incapable of dynamically sensing and responding to changes in terrain height and slope in real-time. They lack the necessary sensors or mechanisms to detect variations in the surfaces they navigate. As a result, these robots rely on pre-programmed locomotion strategies such as fixed gaits for walking robots or pre-defined paths for wheeled robots. These pre-programmed movements are often inefficient, limiting the robot's operational range. Additionally, the inability to adapt to the terrain in real-time can lead to instability, increasing the risk of the robot tipping over or getting stuck, hindering its ability to complete tasks.

1.2 Objectives

The objectives of the project are

- Design and prototype a mobile robot with an active suspension system.
- Process the angle from the Inertial Measurement Unit (IMU) sensor.
- Implement balancing method from the angle data through controlling four stepper motors that adapts to changes in terrain height and slope.
- Detect ArUco Markers with a camera as pre-defined path for navigation.

This project's purpose is to create a mobile robot capable of maintaining balance and navigate uneven terrain autonomously. The data gathering from the IMU sensor is crucial for maintaining balance and understanding the robot’s orientation. The robot can adjust its posture based on the terrain it encounters. Utilizing Aruco markers for navigation provides a structured approach to path planning, enhancing the robot's ability to navigate efficiently.

2. Literature Review

2.1 Suspension Systems Comparisons

Suspension systems are classified into three main categories.

- Passive Suspension Systems

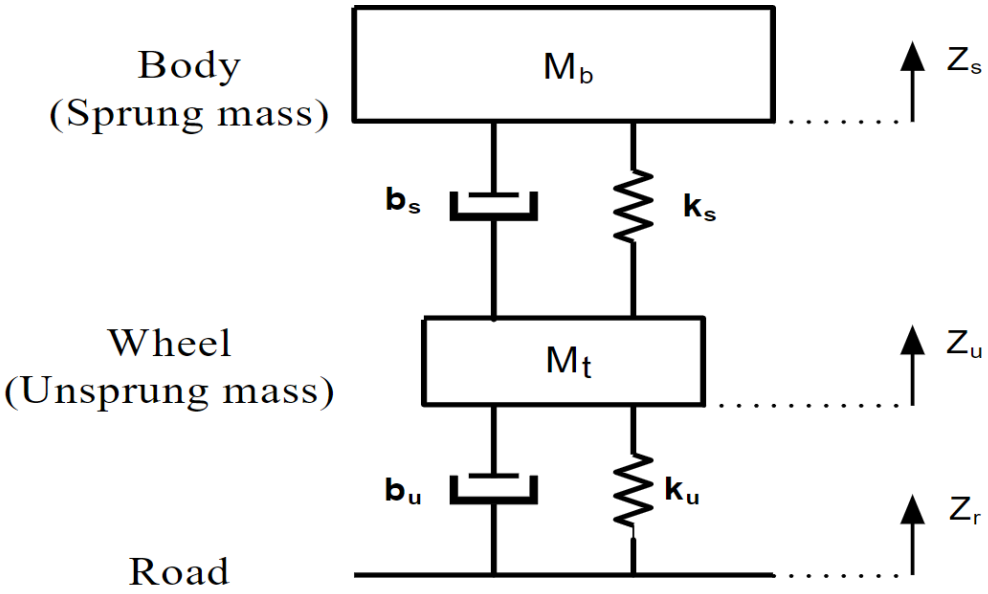

Figure 2. Passive Suspension System [3]

The conventional passive suspension systems consist of springs, dampers, and anti-roll bars to absorb bumps and control bounce. They lack the ability to actively adjust to changing road conditions. The stiffness of spring and the damping coefficient of shock absorber are fixed. This mechanism does not have a mechanism for feedback control [4].

- Semi-Active Suspension Systems

Semi-Active systems improve passive systems by electronically adjusting the shocks' damping characteristics in real-time dissipation of energy [5]. It requires a measuring device along with a controller board to tune the damping.

- Active Suspension Systems

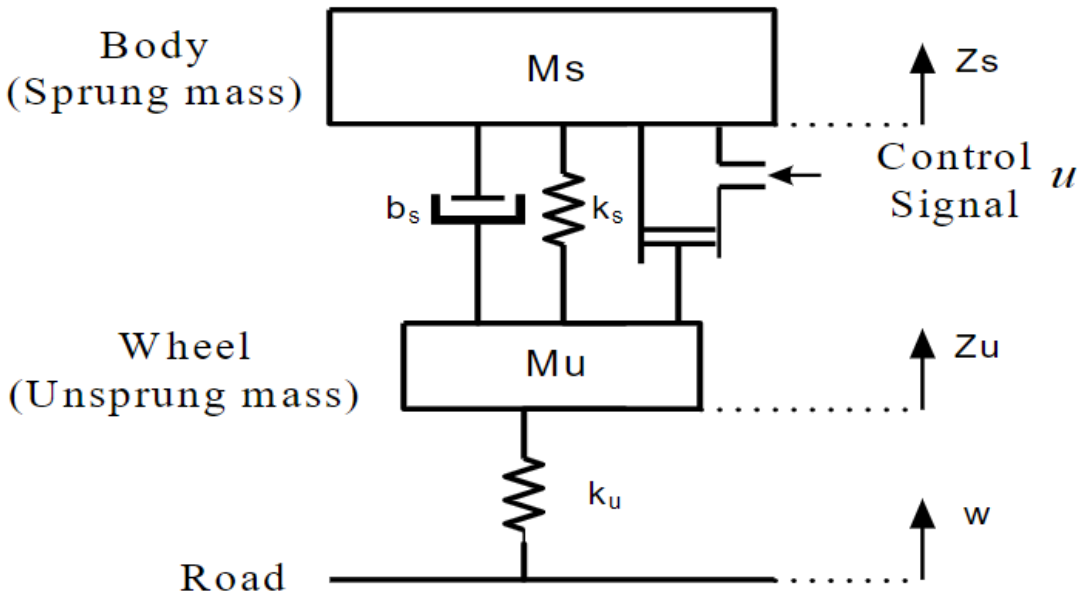

Figure 3. Active Suspension System [3]

Active suspension systems are the most advanced suspension technology. They utilize external power sources and actuators to actively counteract road disturbances and minimize body roll and pitch. Four actuators at each corner of the car are controlled simultaneously. This results in the most comfortable and controlled ride possible.

2.1.2 Advantages and Disadvantages of Active Suspension Systems

Unlike passive systems, active suspension can adapt in real-time to varying road conditions. By maintaining better tire contact with the road surface, active suspension systems can improve overall vehicle grip, enhancing safety.

Figure 4. Active Suspension Components [6]

Despite these advantages, active suspension systems also have limitations. Active suspension systems are significantly more complex than passive systems due to the additional actuators, sensors, and control electronics [6]. This complexity leads to higher manufacturing and maintenance costs. Additional power is required to operate the actuators, which can lead to increased fuel consumption or strain on the vehicle's battery [6]. While it offers significant benefits on uneven terrain, active suspension systems may not provide a substantial advantage on flat, smooth roads. In such scenarios, a simpler and more efficient passive suspension system might be sufficient.

Karnopp [7] examined the aspect of suspension performance related to the filtering of roadway unevenness. According to Karnopp's finding, active suspension can eliminate steady state roll in turns or dive during steady braking. It can respond to preview information or adjust its dynamics based on various sensed aspects of the vehicle motion and disturbances.

However, Karnopp's experiments revealed that a well-designed passive system with adjustable parameters or a semi-active system can achieve comparable performance to an active suspension system in filtering roadway disturbances. One of the limitations of the active suspension system is their suitability for terrain condition. Active suspension systems are not well-suited for flat, even terrain or indoor environments while a simpler passive suspension system is often a more viable option for such conditions. This highlights that active suspension may not always be the most practical solution, particularly for cost-sensitive applications.

2.2 Performance of Passive and Active Suspension Systems

Kumar et al. investigated the performance of vehicle suspensions under random road profiles by comparing a passive system with an active system [8]. They have designed a passive suspension system with a spring and damper, and an active suspension model for a full-vehicle system with a PID controller. Their research focused on the impact of suspension type on the vehicle's vertical displacement, pitching, and rolling motions.

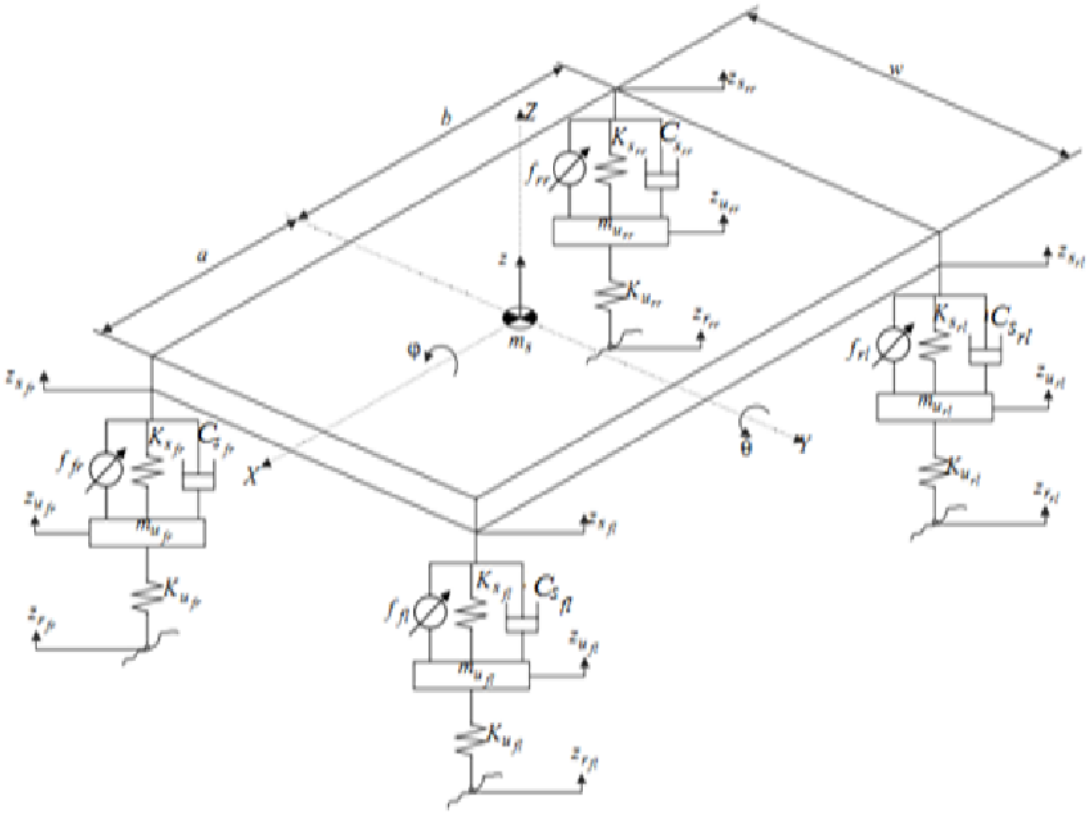

Figure 5. Active Suspension Model for Full-vehicle System [8]

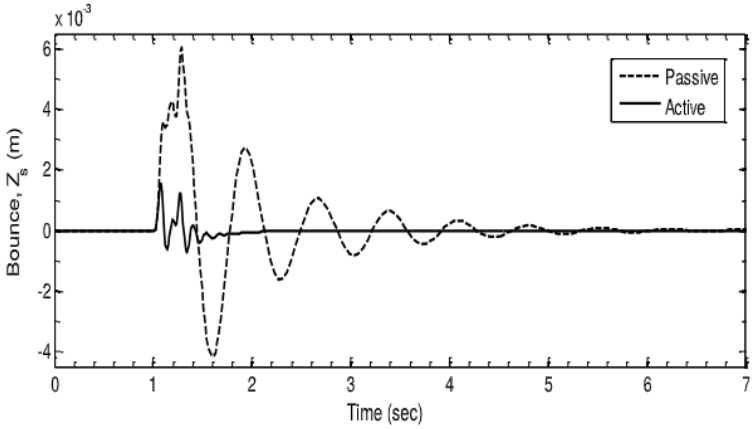

Figure 6. Vertical displacement vs. Time

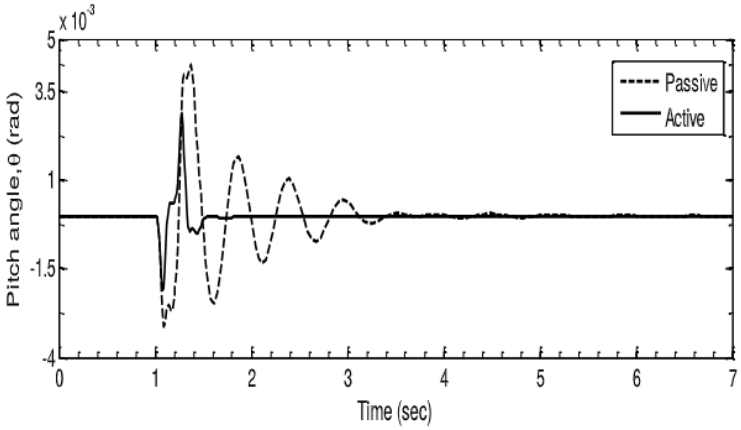

Figure 7. Pitch angle vs. Time

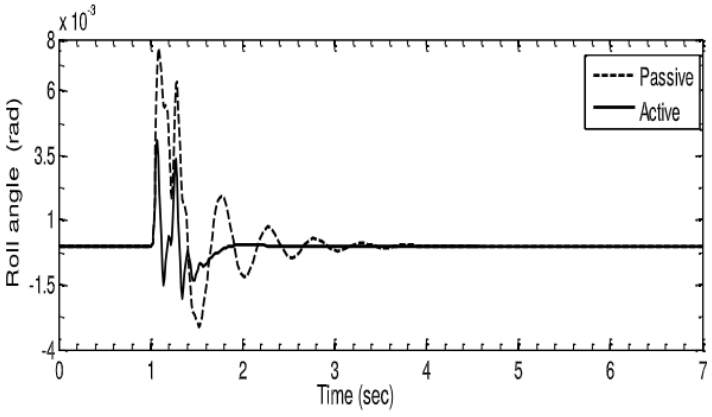

Figure 8. Roll Angle vs. Time

The findings demonstrate significant improvements with the active suspension system compared to the passive system. Active suspension significantly reduced body bounce (vertical displacement) by 75% and settling time by 68%. Pitching angle was also reduced by 35%, with a 55% improvement in settling time. For roll angle, active suspension achieved a 45% reduction in both the peak angle and settling time. These results clearly show the effectiveness of active suspension systems in enhancing ride comfort and safety by minimizing unwanted body movements like bounce, pitch, and roll.

2.3 Existing Mobile Robot with Active Suspension Systems

2.3.1 Agile Omnidirectional Mobile Robot with Active Suspension

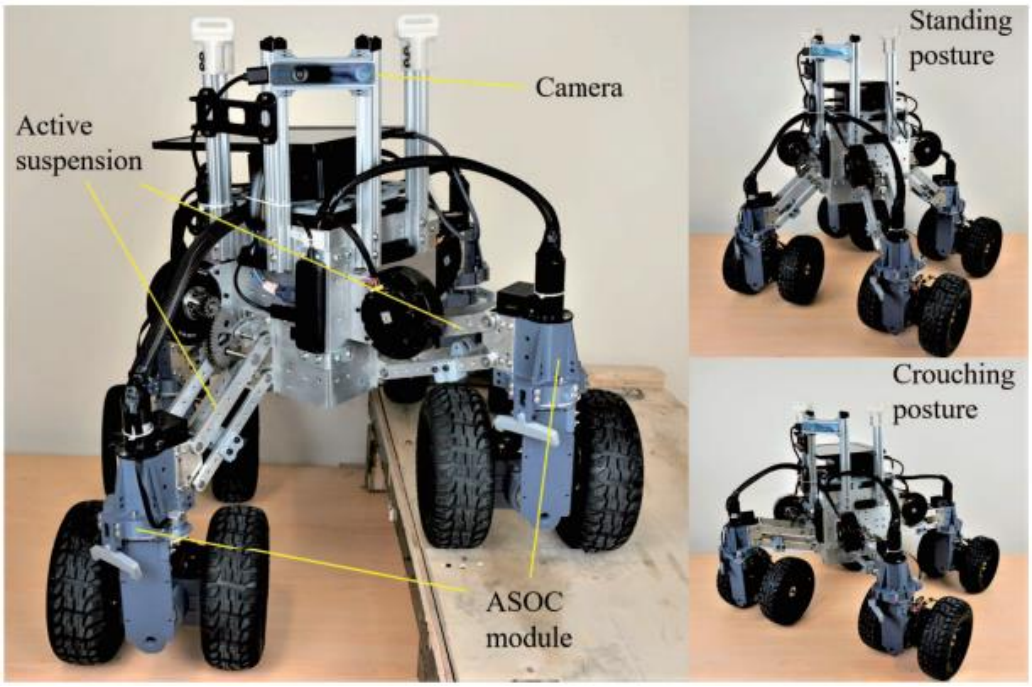

Figure 9. Omnidirectional Mobile Robot Prototype

In 2022, Jiang et al. [9] built an agile omnidirectional mobile robot with active suspension that enables it to navigate uneven terrains with active body posture control and execute sharp turns at high speed in both indoor and outdoor tests. The robot's design allows it to maintain its own balance with a maximum roll angle of 20 degrees when travelling through a maximum stair height of 14cm.

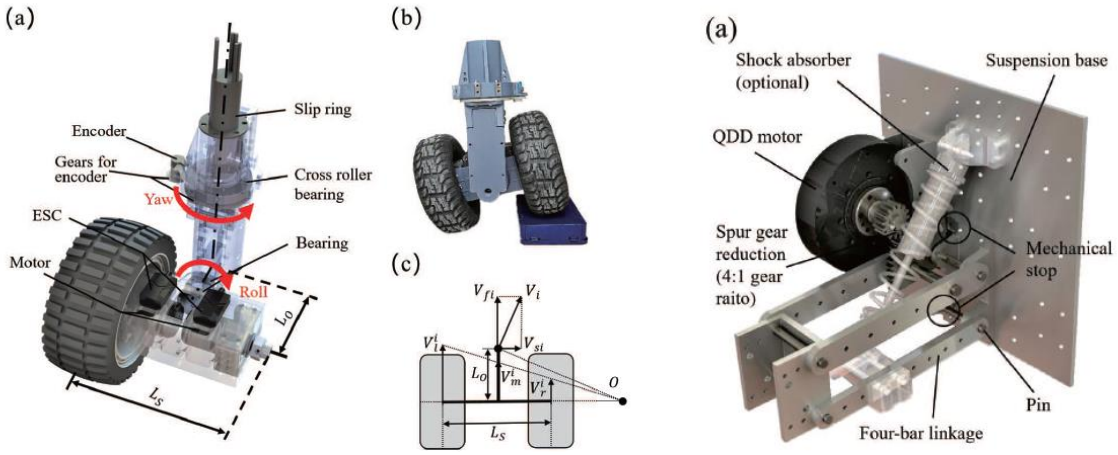

Figure 10. ASOC Module and Active Suspension System [9]

In their design, the active body posture control is achieved by controlling the ASOC module and the active suspension system of each leg.

For the ASOC module, two wheels are attached to two motors connected with a rolling joint. The design allows the whole module to better handle uneven terrain or obstacles and agile maneuvers with capability of fast pose and stiffness adjustment.

The active suspension system utilizes a QDD motor to drive the four-bar linkage, which is attached to the ASOC module, to adjust the height of the mobile robot leg. Additionally, the system incorporates reduction gears and a mechanical stop to enable the active suspension system to have a large torque capacity and fast response without causing damaging the active suspension through limiting the rotation range of the QDD motor.

2.3.2 Mobile Robot with an Active Adjustable Suspension using Pneumatic Linear Actuator

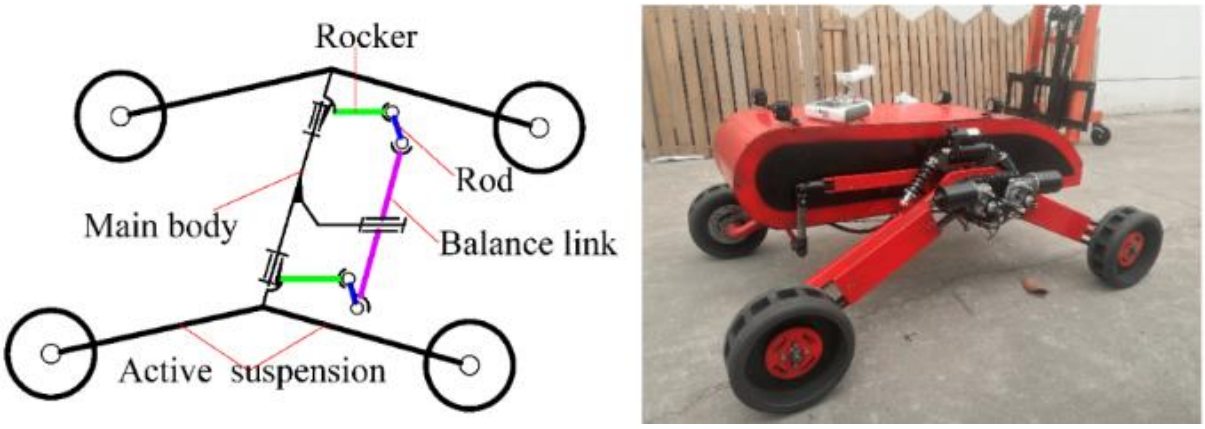

Figure 11. Mobile Robot Model [10]

In 2019, Jiang et al. [10] investigated the lateral stability of a mobile robot with an active adjustable suspension when it reconfigures itself to adjust its roll angle with the active suspension to traverse unstructured terrain, uneven terrain, and climb obstacles or slopes. Their findings revealed that the ground clearance of the mobile robot can be altered by the active suspension, and the robot is most stable at its lowest ground clearance.

In their design, the robot incorporates two modes of maneuver, which are passive mode with a balance–rocker mechanism, and an active mode with an active suspension. Passive mode allows the robot to run on rough and flat terrain with high velocity and an easy control system while active mode allows the robot to adapt itself to move on a complex terrain with high obstacles and steep slopes.

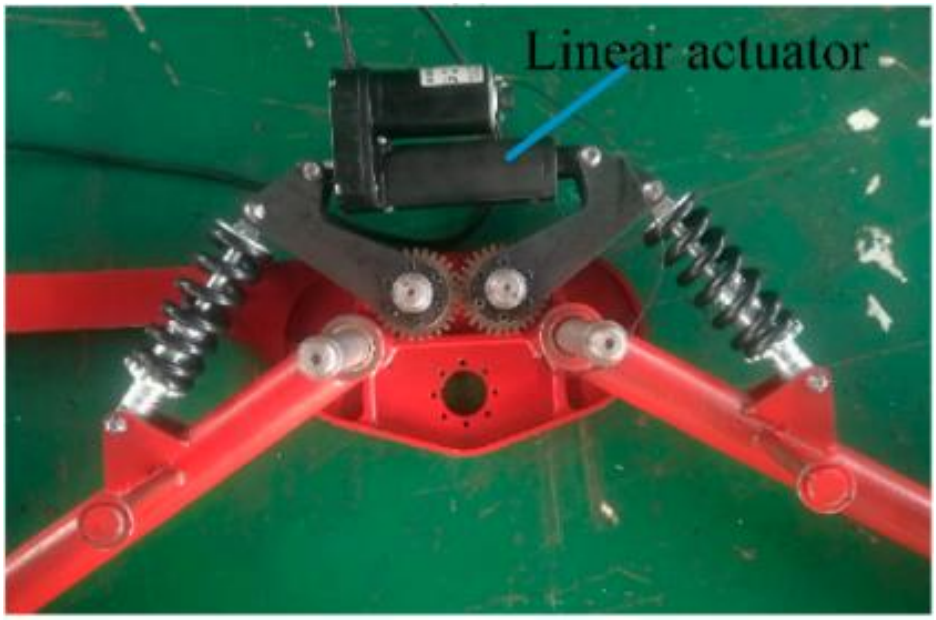

Figure 12. Active suspension system [10]

For the active suspension design of the mobile robot, the control of the active suspension system is controlled by a linear actuator. By adjusting the input of the linear actuator, the length of the linkage connected to the leg of the mobile robot changes, thus causing alternations in the angle between the legs of the mobile robot to achieve height adjustment of the mobile robot.

2.3.3 Mobile Robot with an Active System with Scissors System

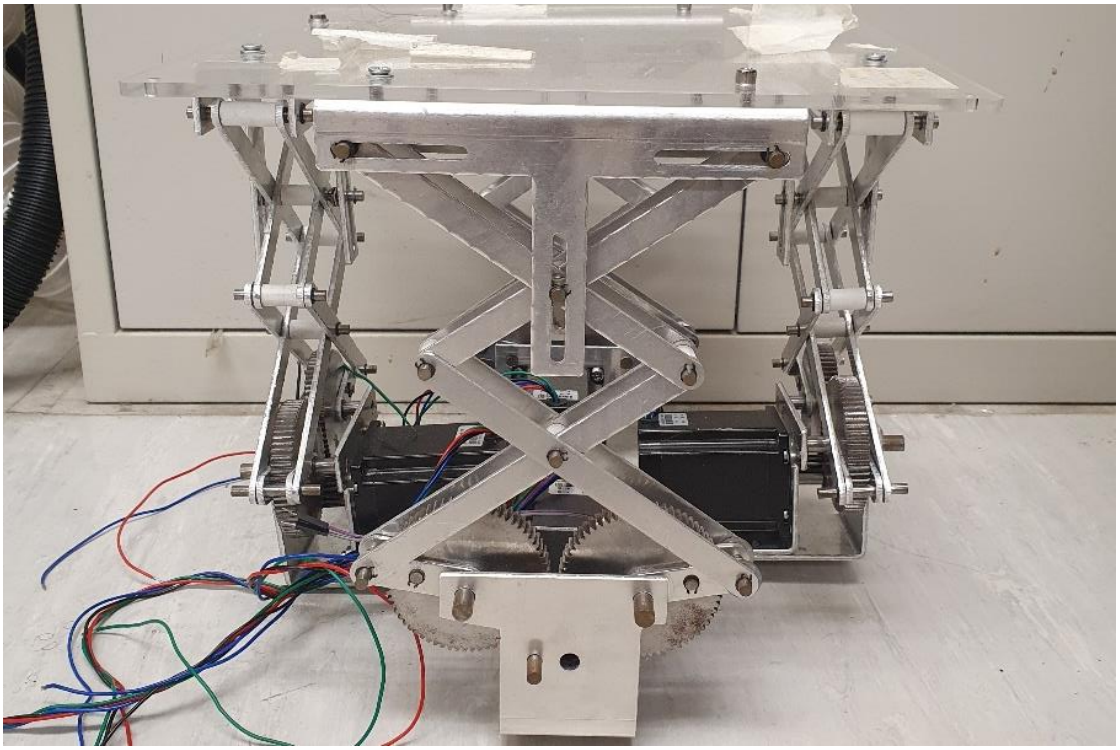

Figure 13. Active Suspension with Scissors System

The mobile robot is equipped with an active suspension system featuring a scissors mechanism. This innovation was developed by previous final year students under the guidance of our supervisor, Dr. Chu. The stepper motors adjust the robot's level through extending or retracting the scissors' structure to balance the top plate.

2.3 Autonomous navigation of Mobile robot

Autonomous navigation in uneven terrain has posed a significant challenge in the field of robotics [11]. It plays a crucial role in enabling a robot to navigate effectively in the given environment towards its target location. Decision-making requires continuous acquisition of knowledge from the surrounding environment while the robot is in motion [12]. The utilization of multisensory perception systems has facilitated accurate mapping and localization.

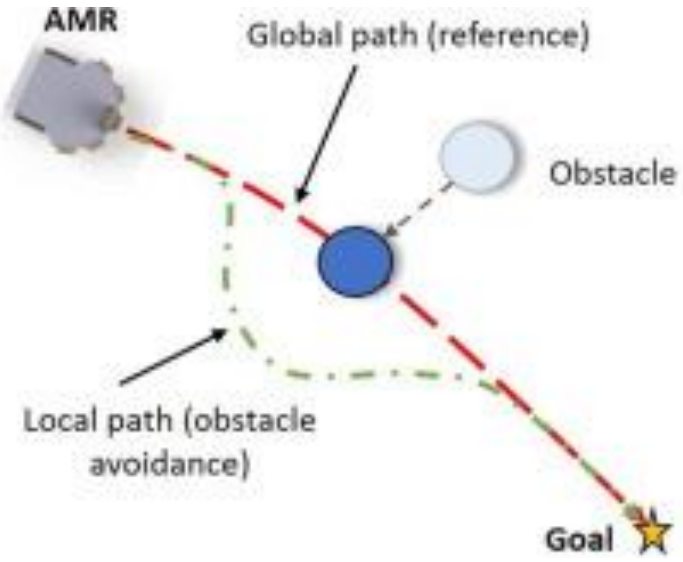

Figure 14. Global and Local Path Planning of a Mobile Robot [13]

Tradition Path planning in mobile robots can be classified into two types: global path planning and local path planning. Global path planning employs graph-based approaches such as Dijkstra and A* algorithms and sampling-based approaches such as RRT [14]. Dijkstra's algorithm computes the shortest path from a start node to all other nodes by exploring the neighboring nodes to find a shorter path. In global path planning, the robot must have prior knowledge of the entire environment before operation. The A* algorithm finds the most efficient route in robot navigation. It implements a best-first search method that has high performance and accuracy. In contrast, local

path planning operates in environments where most information is unknown to the robot. It proves more effective in dynamic environments, but it can get trapped in local minimum due to its reliance on fastest descent optimization [15]. A combination of global and local planning can be a promising solution, but it may require complex decision-making structures.

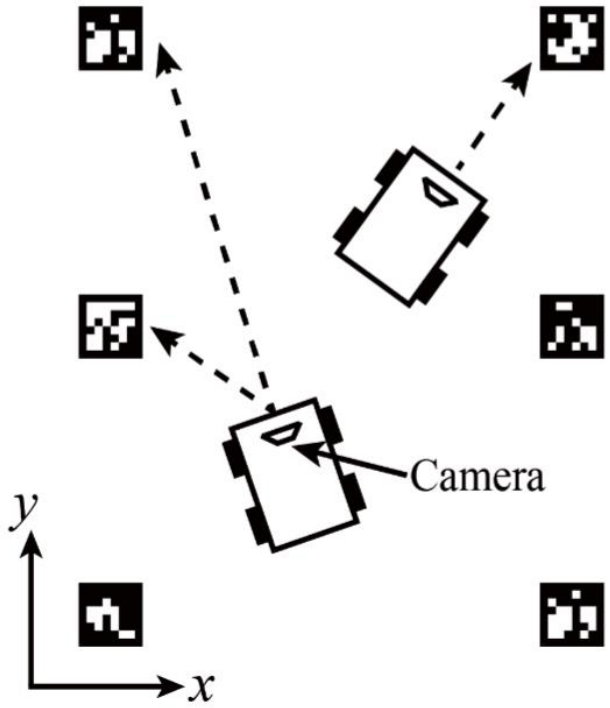

Figure 15. Solution of Visual Localization [16]

ArUco markers, a type of visual fiducial marker, can be strategically placed in the environment to aid in mapping and localization during autonomous navigation. While originally designed for augmented reality, ArUco markers are widely used for mobile robot localization due to their ease implementation and reliable detection capabilities [16]. By strategically placing markers in the environment, robots can use their cameras to detect and track the markers [17]. ArUco marker detection offers an alternative approach for path planning in controlled environments.

2.4 Terrain Identification

For effective path planning in outdoor robot navigation, the creation of a map and identification of terrain becomes essential [18]. Simultaneous Localization and Mapping (SLAM) is a technique that constructs and updates a map of an unknown environment while keeping track of the location of the robot. Sensors like a LiDAR (Light Detection and Ranging) are employed to perceive their surroundings. It is widely used in real life applications such as search and rescue field [19]. The data collected from sensors is then used to plan the robot's path. Machine learning techniques like deep learning are implemented to classification of objects like grass and rock.

2.5 Methodology Selected

2.5.1. Active Suspension Motor Selection

Selecting the appropriate motor type is crucial for the active suspension system's functionality. There are several options for active suspension systems and wheel driving, including stepper motor, DC motor, hydraulic and pneumatic linear actuators.

Figure 16. Ball Screw Stepper Motor

The active suspension system requires a motor that can deliver holding torque and precise control over rotational angle, speed, and extension distance for effective leg movement. This eliminates DC motors and pneumatic linear actuators from consideration as they lack precise control over these critical parameters. Both stepper motors and hydraulic actuators are suitable for active suspension systems. However, stepper motors offer advantages in terms of cost and control simplicity. Their lower cost and easier controllability make them the ideal choice for this project's active suspension system.

Based on an analysis of existing mobile robot designs with active suspension, our robot utilized lead screw stepper motors for each leg. These motors are paired with DC motors connected to the wheels, a module for each leg. The lead screw stepper motors directly control the vertical movement of the legs of the mobile robot to perform the function of an active suspension system.

2.5.2. Mechanism of ball screw stepper motor

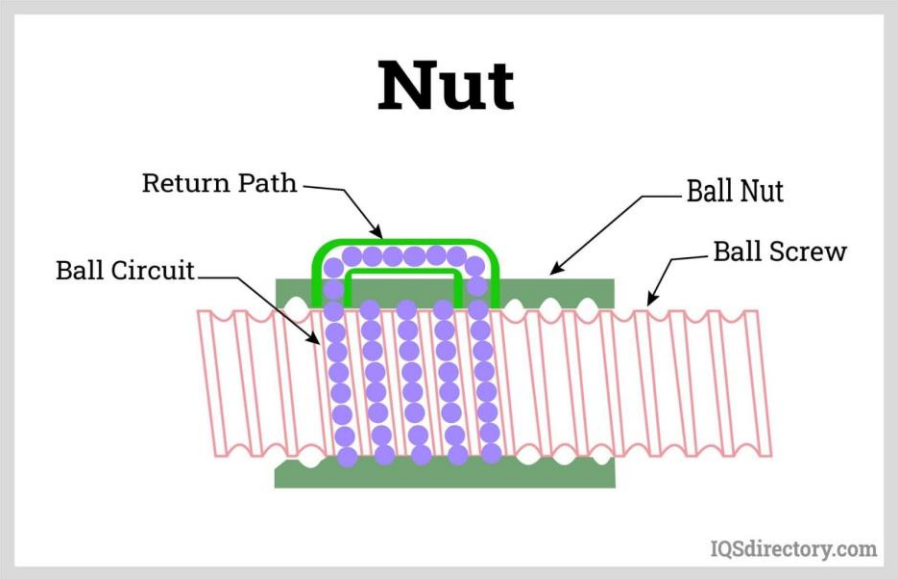

Figure 17. Ball Screw Stepper Motor Mechanism

The ball screw stepper motor is capable to convert rotary motion to linear motion without rotating the nut by applying a recirculating ball mechanism between the screw shaft and the nut. It minimizes friction, allowing the motor to convert rotational motion to linear motion with minimal energy loss. The rolling balls travel in the closed path and keep returning to the starting point through the return path as a cycle in the ball nut for transmitting forces with high accuracy, precision, and repeatability. This precise control is crucial for controlled leg movements of the mobile robot.

2.5.2. Active Suspension Control

The active suspension system on our mobile robot incorporates an Inertial Measurement Unit (IMU) to collect data on the robot's orientation. This data is crucial for the control system to adjust the lead screw stepper motors in each leg. By adjusting the leg positions, the active suspension system maintains the robot's stability and optimizes its performance on uneven terrain.

2.5.2. Mobile Robot Navigation

Our mobile robot implemented a range of sensors including a depth camera, LiDAR, and ultrasonic sensor to identify the surrounding environment and build a map. The robot can detect Aruco markers in its surroundings. These markers provide valuable reference points that aid in path planning. LIDAR (Light Detection and Ranging) is used to construct a comprehensive map of the environment using SLAM (Simultaneous Localization and Mapping). This map provides essential information for the robot’s navigation.

3. Design and Implementation

3.1 Design Criteria and Considerations

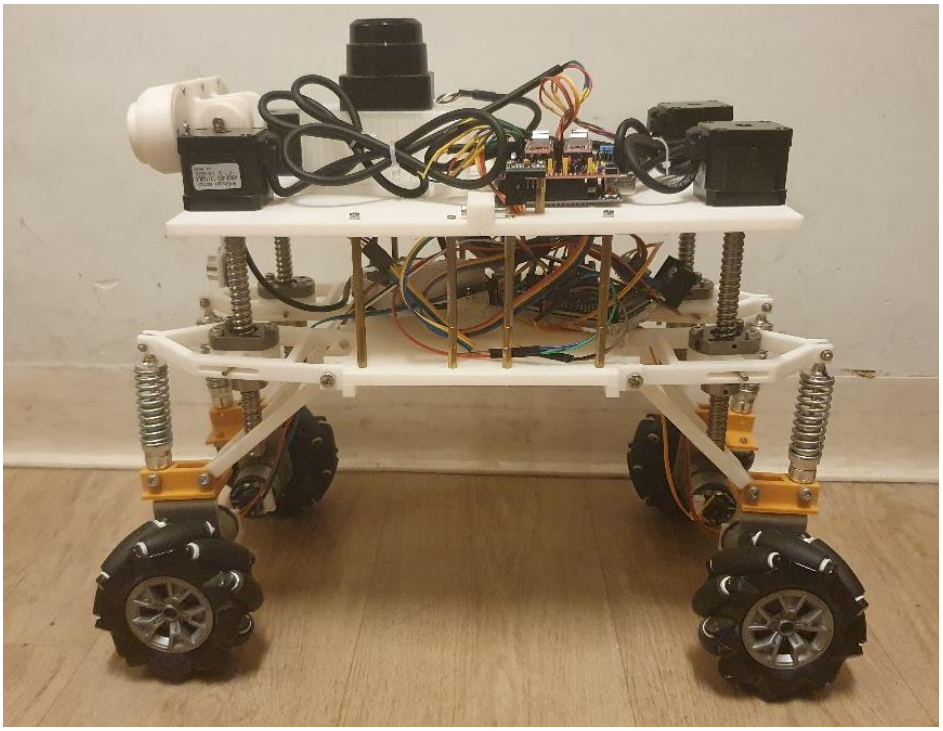

Figure 18. Overview of the Active Suspension Mobile Robot

- Dimensions of the robot: 1000 mm (Length) x 500 mm (Width) x 500 mm (Height)

In this project, medium to small size mobile robot is chosen preferred. The manufacturing method of the robot is 3D printing and purchasing customized components online. A larger robot would be more expensive to produce with these methods. The project's budget is limited to 3000 HKD, so a medium to small-size mobile robot is a better option for the cost of manufacturing and budget of the project.

- Weight: 5kg (robot itself)
- Expected Carry Weight: 5 kg.

Considering the conceptual design of the mobile robot, the mobile robot consists of 4 ball screw stepper motors and 4 DC motors, and therefore, the weight of the robot is expected to be around 5 kg. It is expected that the robot will carry the weight the same as itself without any problem.

- Operating Environment: Indoor and outdoor walking paths on campus

As the target of the mobile robot is to maintain balance when passing through uneven terrain with automatic navigation, walking pathway shown in the figure18 is a suitable location for testing both functions of the mobile robot as it includes uneven surface, slope, straight and turning path.

- Navigational Area: 100 meters long by 1 meter wide

The space the robot that the robot can travel is 1-meter-wide path and travel 100 meters long. The walking path includes variations in terrain, such as bumps, dips, and potential obstacles. The maximum incline/decline angle of the path it should climb is 20 degrees. Considering these variations, the robot's active suspension system is designed to handle height differences ranging from 5 cm to 10 cm.

- Climbing Ability: 2 cm high

The robot's design allows it to climb obstacles up to 3 centimeters high. This includes curbs, slight inclines, and small steps. The combination of motor power, traction control, and a well-positioned center of gravity enables this climbing capability.

- Active suspension range: 10 cm

Active suspension systems can achieve a wider range on larger robots. This medium-sized robot handles uneven terrain with an adjustable range of 5 cm to 10 cm.

3.2 Mobile Robot Design and Implementation

In the design, the active suspension system module includes the frame of the mobile robot, the wheel, the motor, springs, and the connecting parts. The frame is designed to accommodate electronic components' wiring and support the entire robot structure.

3.2.1 Active Suspension System Conceptual Design

In traversing uneven terrain, the active suspension dynamically adjusts the wheel's height based on the data from the IMU. Lead screw stepper control this system, ensuring precise vertical motion.

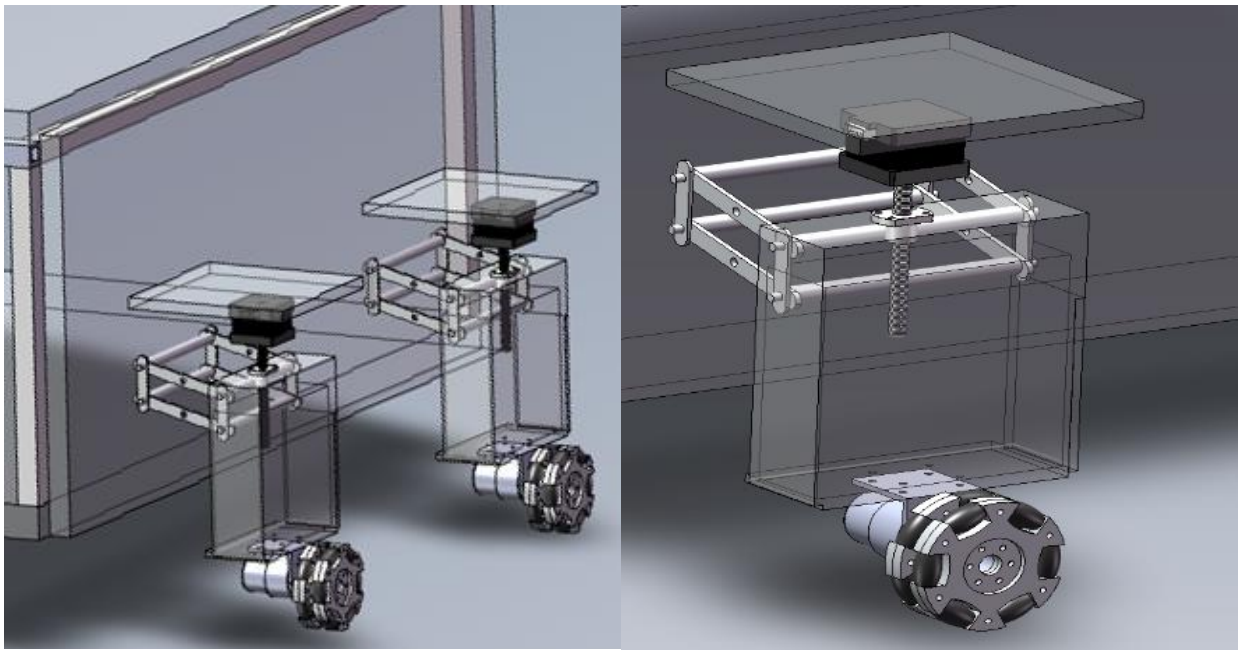

Figure 19. Model of Mobile Robot Conceptual Design (left) Active Suspension System (right)

Stepper mount is mounted on the plate adjust the level of the wheel height. However, the current design lacks proper mounting for the stepper motor. The connecting plate between the robot's body and the motor could bend under this pressure. Additionally, a spring can absorb shock and vibration, reducing stress on the mounting plate and improving overall stability.

3.2.2 Mobile Robot Overall Design

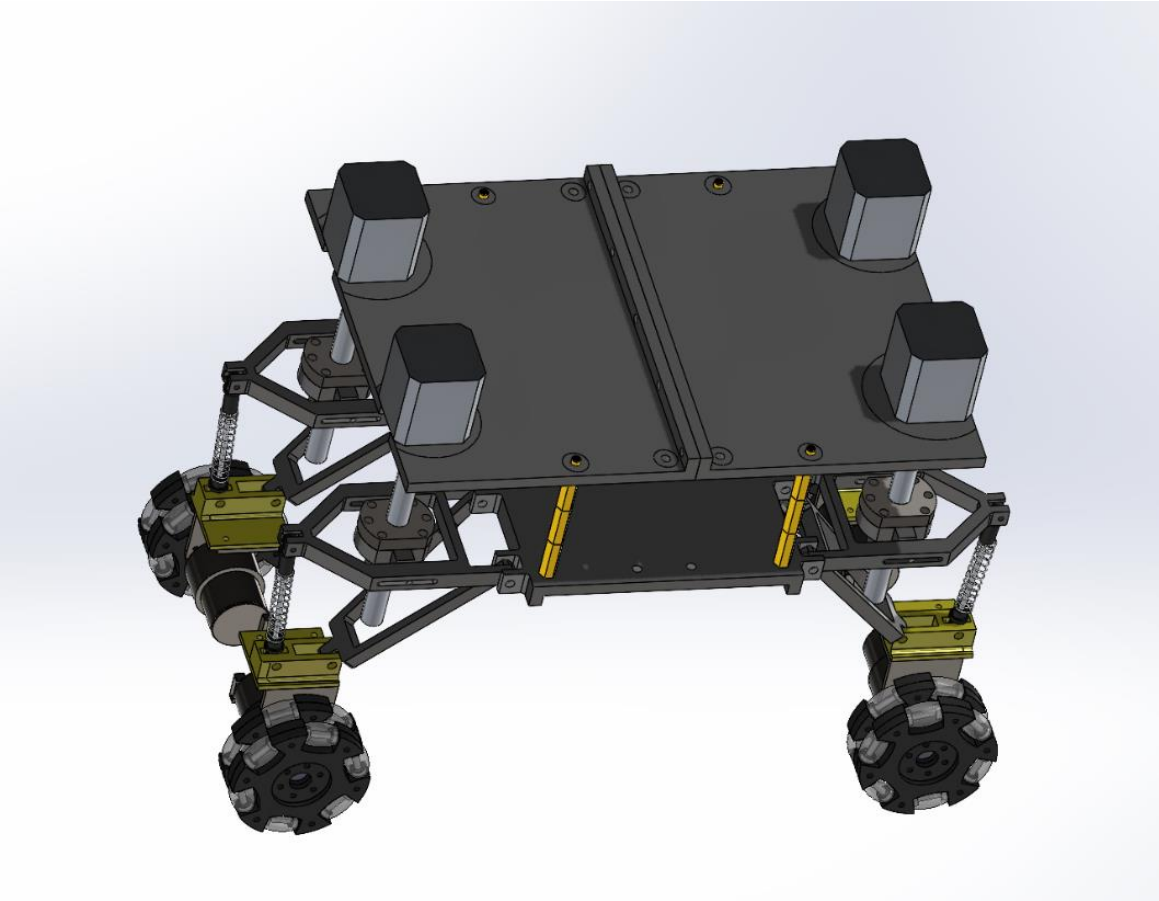

Figure 20. Mobile Robot Revised Design with Spring

The revised mobile robot design now includes a spring between new linkages and the DC motor mount. This setup effectively absorbs shocks and vibrations, while the redesigned linkages offer enhanced stability for the robot's body.

3.2.3 Active Suspension Design

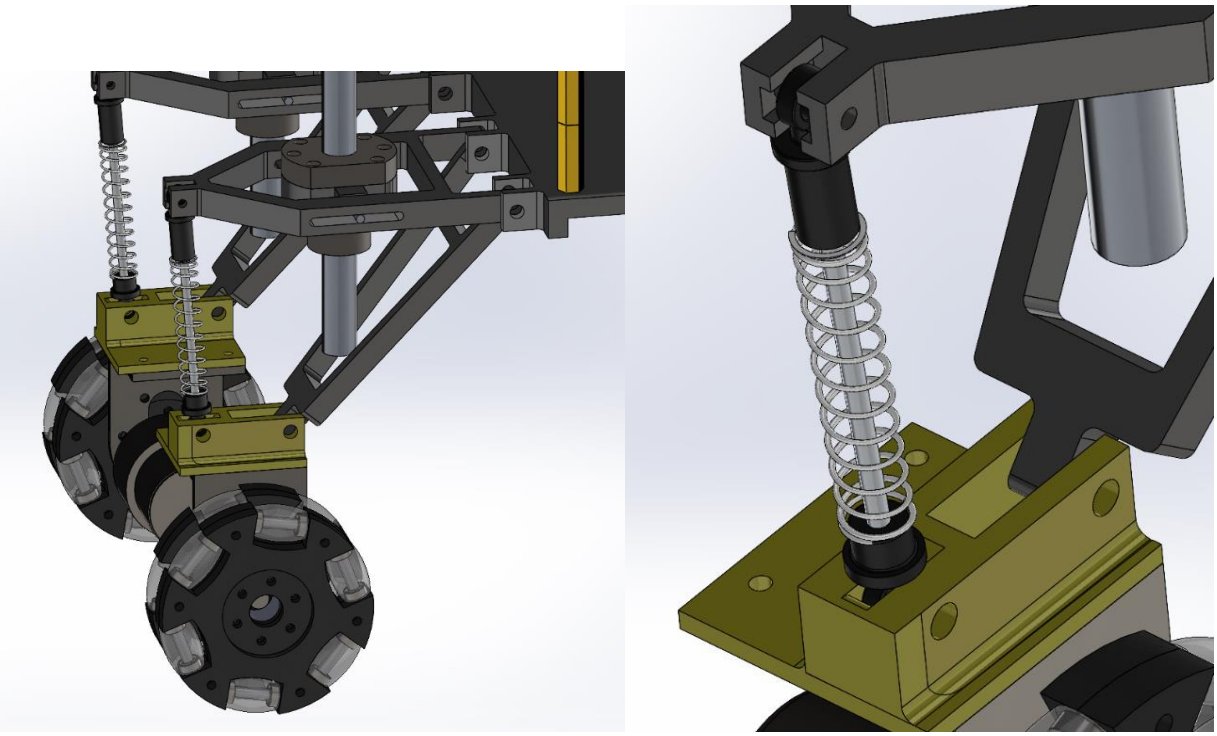

Figure 21. Mobile Robot Leg Configuration and DC Mounting Revised Design

Due to the weight of the DC motor, the wheel with the DC motor tends to bend towards the car body, which makes the mobile robot with the conceptual design unable to stand properly. Therefore, the orientation and configuration of the robot legs are changed, which allows the mobile robot body to better support the DC motors with wheels from falling due to its own weight.

Additionally, with the revised DC mounting, the connection point between wheels and linkages is closer to the wheel to allow the robot to stand more properly by moving the center of force applied to the L shaped mounting closer to the wheel and able to adjust the level of each leg while maintaining balance as more force is applied near the wheel to better cancel out the weight of the DC motor.

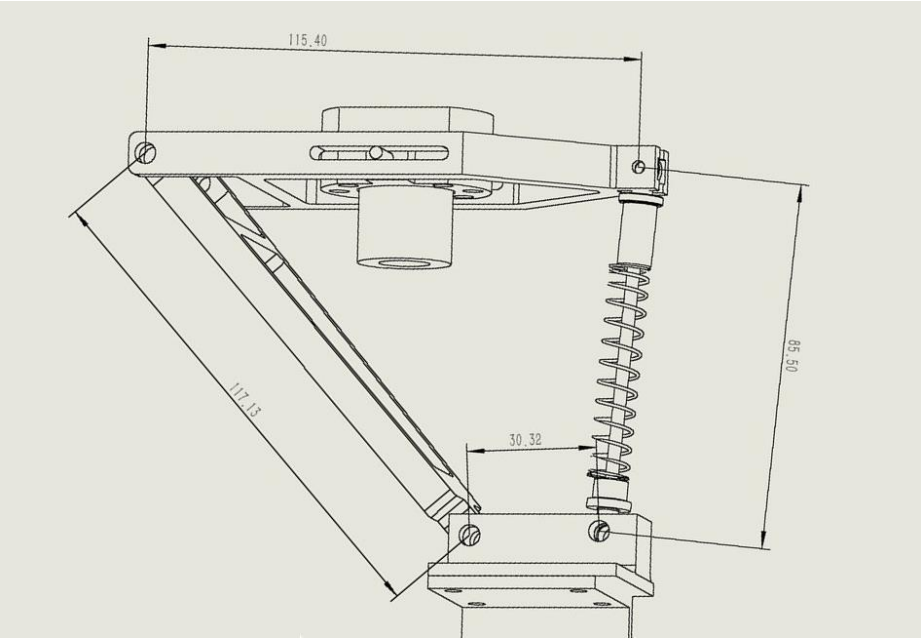

Figure 22. 4-bar Linkage DOF (Degree of Freedom) with Link Dimension Drawing

To examine the structure of the suspension system, Grashof's criterion is used.

From the figure, S=30.32mm, L=117.13mm, P=85.5mm, Q=115.4mm.

Where the length of the S represent mounting length.

The length of the L represent react linkage length.

The length of the P represent spring length.

The length of Q represent Slot linkage length.

$$S + L < P + Q$$

The active suspension system also includes a Class I Grashof Four-bar Linkage. However, our design of the mount restricts the movement of the spring, limiting its travel to 10 degrees. This changes the design to become a triple-rockers Class II Non-Grashof Triple Rocker design where no link can fully rotate. This minimizes the tilting of the bottom wheel mount during operation.

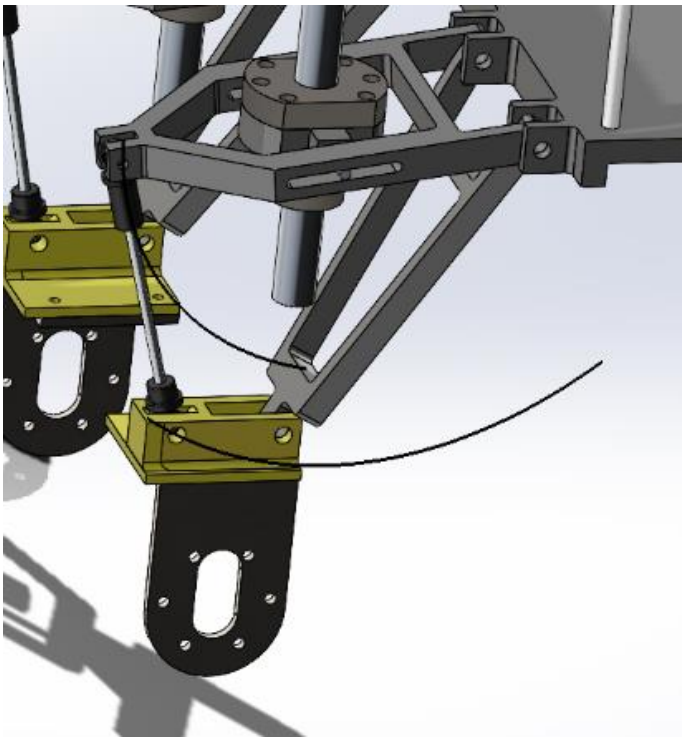

Figure 23. Motion Analysis of Active Suspension System

Based on the motion analysis, the chosen dimensions for the linkage design successfully achieve the desired vertical travel for the robot leg.

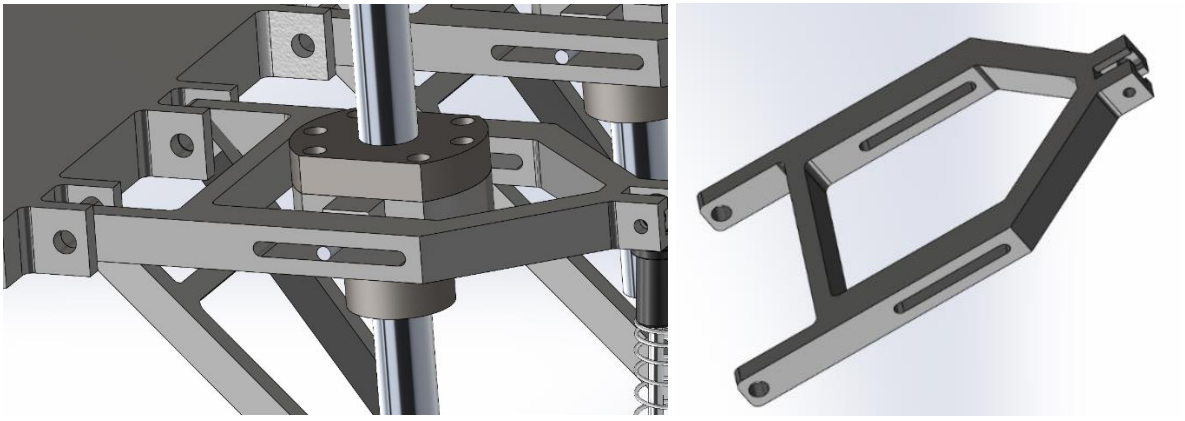

Figure 24. Mobile Robot Active Suspension and Stepper Motor Mounting

In the original design, the nut of the ball screw of the stepper motor is directly connected to the linkage for controlling the vertical movement of the mobile robot leg. However, adjusting the

vertical height of the mobile robot leg also causes horizontal movement. The leg follows a curved path in an arc-like trajectory in an arc shape.

The revised design addresses this issue by introducing a mount connected to the ball screw. Two metal cylinders are inserted into the hole of the mount. The other end of the metal cylinders is inserted into the slot created on the linkage of the active suspension system. This design with the slot creates two degrees of freedom where it allows translation and rotation of the metal rod to suit the trajectory of the linkage. This reduces stress on the structure and prevents damage.

The stepper motor is securely mounted on the upper plate with screws to ensure precise positioning and prevent movement during operation.

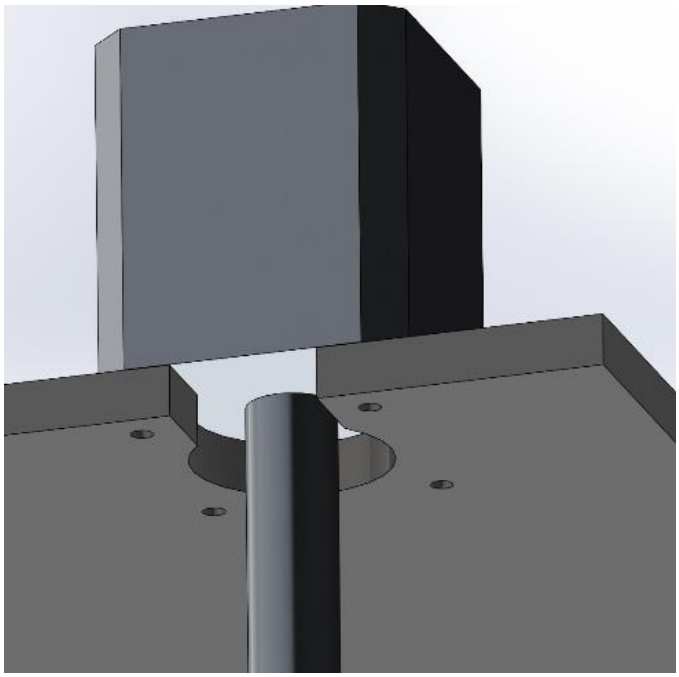

Figure 25. Stepper Motor Mounting

The stepper motor is securely mounted on the upper plate with screws to ensure precise positioning and prevent movement during operation.

3.2.4 Platform Design

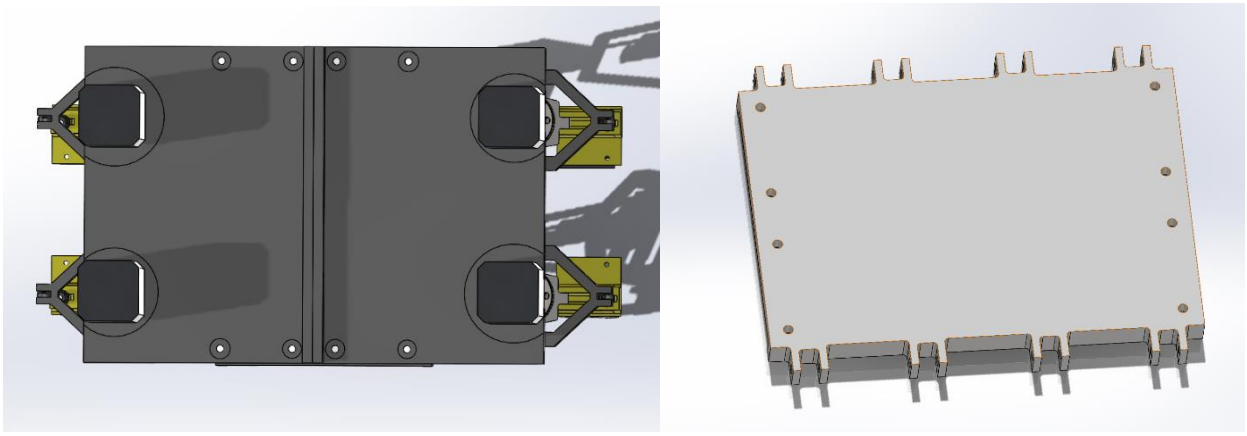

Figure 26. Upper Plate and Lower Plate Design

The upper plate serves as a platform for the stepper motor. The lower plate provides the joint for connecting linkages of the active suspension system. This connection allows for controlled movement of the legs of the mobile robot. Other electronic components would be mounted on the plates.

The components' dimensions and thickness consider the limitations of the available 3D printers. The maximum build volume of these printers is 256mm x 256mm x 256mm. Larger components, like the upper platform, are divided into sections to fit within the printer's build area. Parts are intentionally designed to be no longer than 256mm in any dimension for printing purposes. Despite using lower density materials, components are designed with a thickness of 7mm to 10mm. This ensures they can still support the weight of the entire robot, mount the stepper motors, and connect all the linkage of active suspensions system.

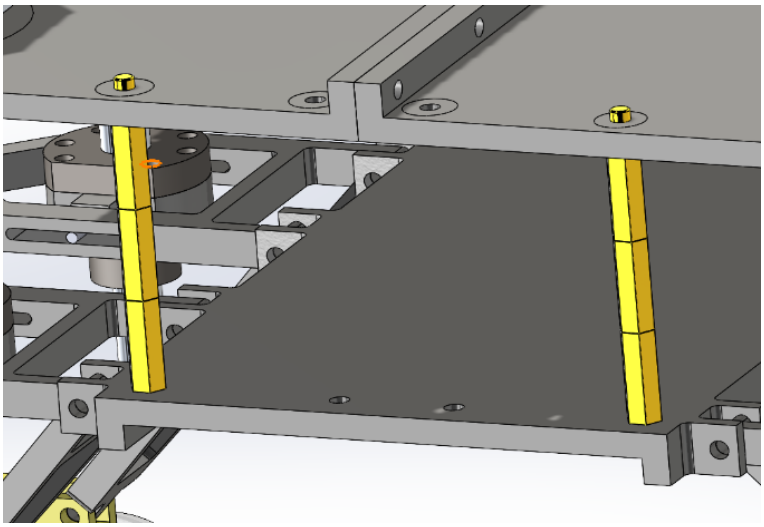

Figure 27. Plates Connection

The upper plate and the lower plate are connected using hexagonal standoffs and screws. They can withstand significant loads.

3.2.5 Spring connection design

During testing, the movement of the spring affects the control of the robot. Therefore we have added limiting structures to the design.

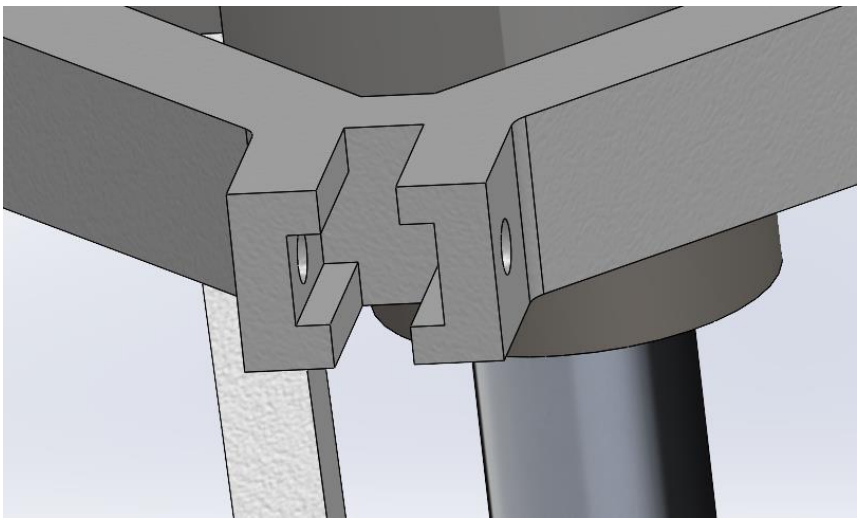

Figure 28. Slot Linkage Spring Connection Design

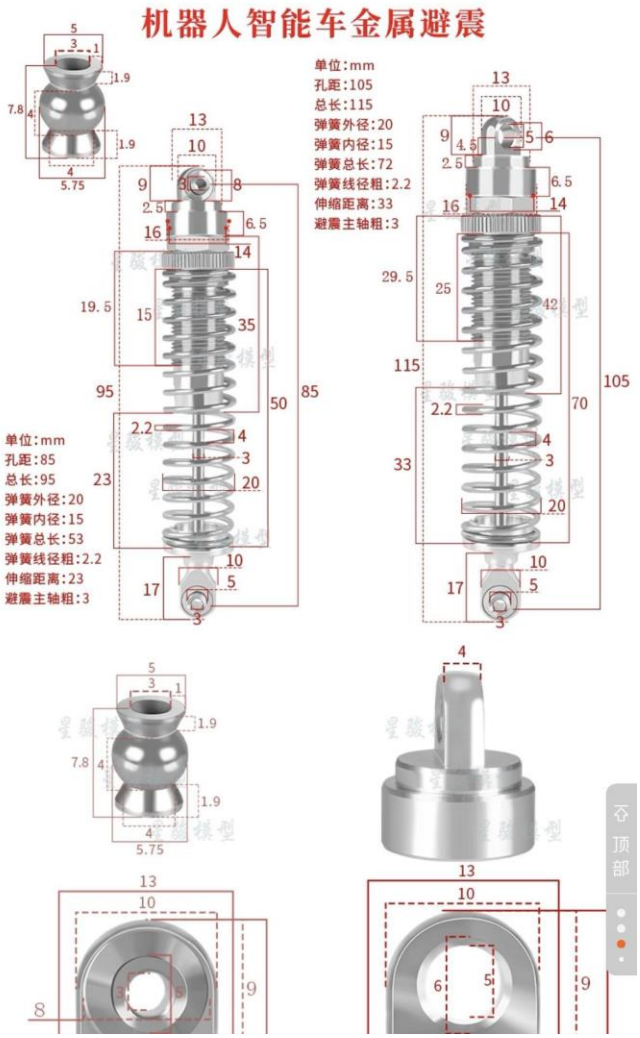

Figure 29. Spring Selection

When selecting springs for mechanical or engineering applications, consider the type of material, and spring size. Material selection affects the strength of the spring, while wire diameter and length dimensions determine the load-carrying capacity and deflection rate of the spring. Manufacturing tolerances and whether robotic connection standards are in place. Therefore, we have chosen the spring shown in the diagram, using aluminum combined and a length of 85mm. load strength between 35-45kg. The choice of spring meets the design requirements and increases the stabilizer of the car.

3.3 Material selection

Figure 30. PLA

In 3D printing, the common option for printing materials is Acrylonitrile Butadiene Styrene (ABS) and Polylactide (PLA). There are mainly two reasons for selecting PLA instead of ABS to be the material for 3D printing in this project.

The print components are mainly support for the mobile robot, which required higher hardness and stiffness. By comparing the properties of PLA and ABS according to results of compression tests of ABS and PLA material from Universitas Negeri Jakarta [21], when both material have a thickness of 0.35mm, PLA has an average ultimate tensile strength of 68.43 MPa, average yield strength of 67.48 MPa and average strain of 2.94 % while ABS has an average ultimate tensile strength of 36.59 MPa, average yield strength of 33.41 MPa and average strain of 4.92 %. Therefore, PLA has a higher Modulus of Elasticity and strength, which means that it has higher hardness and stiffness than ABS.

Additionally, the printability of PLA is better than that of ABS as it does not require a high print bed temperature for printing. Therefore, PLA is a better material for this project to be the material for 3D printing after considering the properties of PLA and ABS.

Figure 31. Pure Iron

The mobile robot frame includes the platforms, linkages, and different mountings of the mobile robot. The materials of the mobile robot frame are PLA and pure iron. These two materials are selected because 3D printing technology is the main manufacturing method of the prototype of the mobile robot. While the remaining part is using pure iron to ensure that those components can support the structure properly or to have lower friction to allow translation and rotation with a lower price.

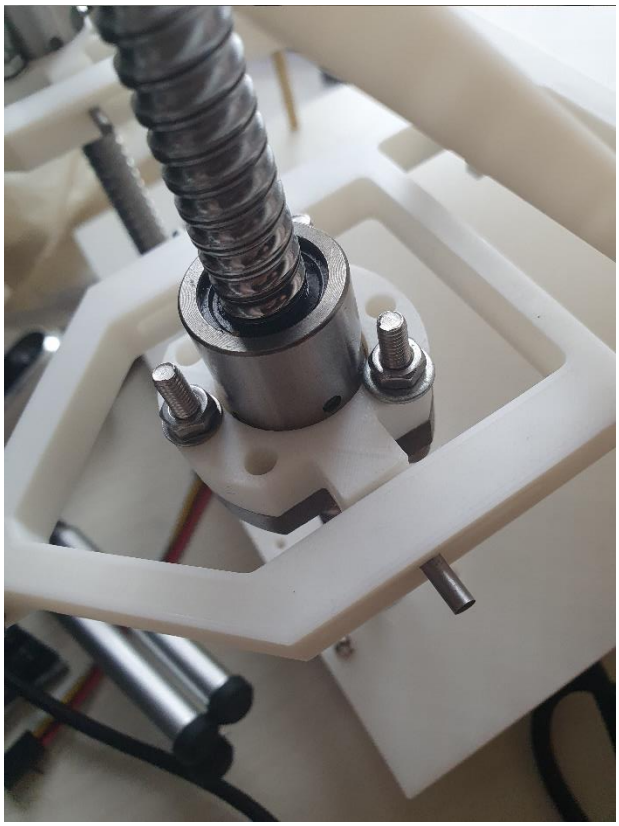

Figure 32. Metal Rod Connecting Ball Screw and Linkage

This figure 32 illustrates the connection between the ball screw and linkage mechanism. When the stepper motor rotates, it drives the ball screw up and down along its threaded shaft. This axial movement of the ball screw is translated into rotational motion of the metal rod. The metal rod then rotates within the slot of the linkage, enabling the upward or downward movement of the connected component.

By using PLA for most components, the overall weight of the mobile robot is greatly reduced. This lighter weight translates to less force dragging the robot down due to gravity. Consequently, the active suspension system requires less torque to adjust the height of each leg, improving its efficiency. The components have carried out stress analysis to ensure the weight of the entire mobile robot will not break or deform the whole structure when operating.

3.3.1 Manufacturing

Following the successful FEA analysis, 3D printing was employed to manufacture the parts including mountings for electronic components. The CAD model was 3D printed using the Bamboo Lab X1 with a nozzle size of 0.4mm (about 0.02 in) to construct the bases, linkages, and mounts.

3.4 Finite Element Analysis (FEA)

In the Finite Element Analysis (FEA), our group performed the FEA on SolidWorks to simulate the forces applied on the mobile robot in extreme cases. The load was set 400N, and the material applied is ABS plastic. Total force of 400N is applied to different designated location to simulation the force applied to the mobile robot from different source, including weight of mobile robot, force from stepper motor, etc. von Mises stands for equivalent force. The unit of von Mises is N/m^2. The distribution of colors indicates the distribution of stresses on the part. The red arrow represents the direction of the analog force, and the green arrow represents the fixture of the model.

The material mainly being applied is PLA in real life but in SolidWorks ABS is selected for the FEA analysis as PLA is not predefined material in SolidWorks, according to the results of compression tests of ABS and PLA material from Universitas Negeri Jakarta [21], PLA has an average ultimate tensile strength of 68.43 MPa, average yield strength of 67.48 MPa when the thickness is 0.35mm, which equals to 6.843e +07 N/m^2 and 6.748 e +07 N/m^2.

It is assumed that PLA use in the mobile platform has twice the value of average ultimate tensile strength and average yield strength of ABS use in the simulation of SolidWorks.

The safety of the FEA analysis is 4 to ensure the safety operation of the mobile robot, it is assuming that the robot is having a loading of 100N in total, and therefore a total force of 400N is applied to the mobile robot considering safety factor.

3.4.1 Stress Analysis

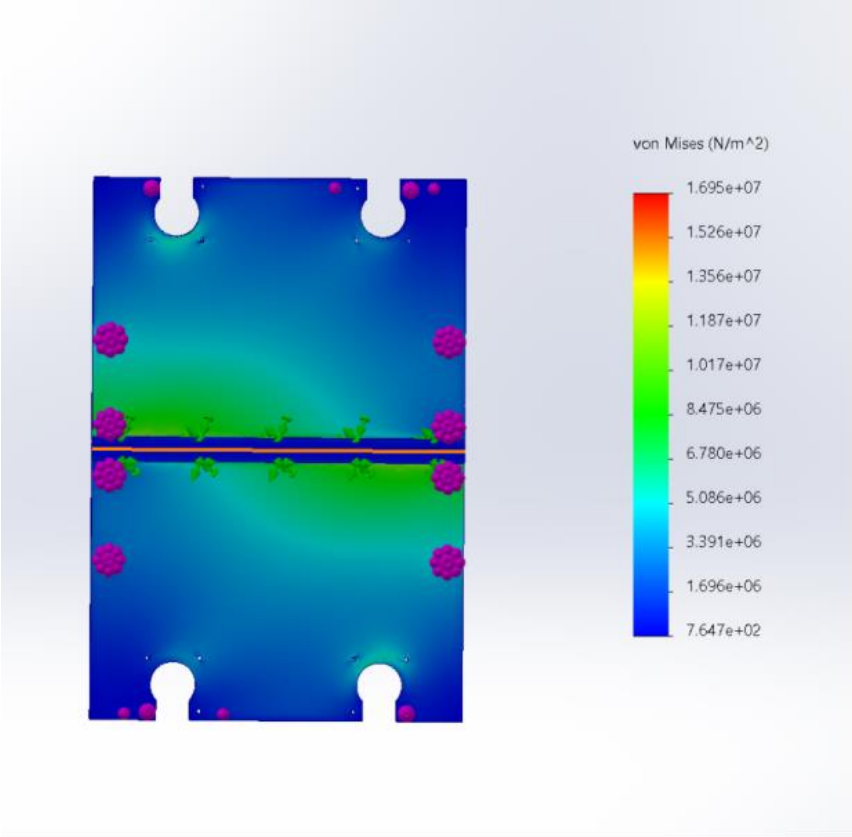

Figure 33. FEA of Mobile Robot Upper Platform

Figure 33 shows the FEA result of the mobile robot upper platform, which is connected to the lower platform and holds all the stepper motors. 400N is applied to the platform during FEA and the result of the FEA shows that the structure of the mobile robot platform does not deform or break in the material of PLA. The mobile robot platform can handle the weight of the stepper motor without any damage as the maximum stress (1.695 e +07 N/m^2 does not exceed the yield strength of PLA (6.748 e +07 N/m^2).

According to the color distribution, the force distribution in the upper platform is concentrated in the connection between the two plates. Therefore, users need to avoid placing the load at the center of the two plates.

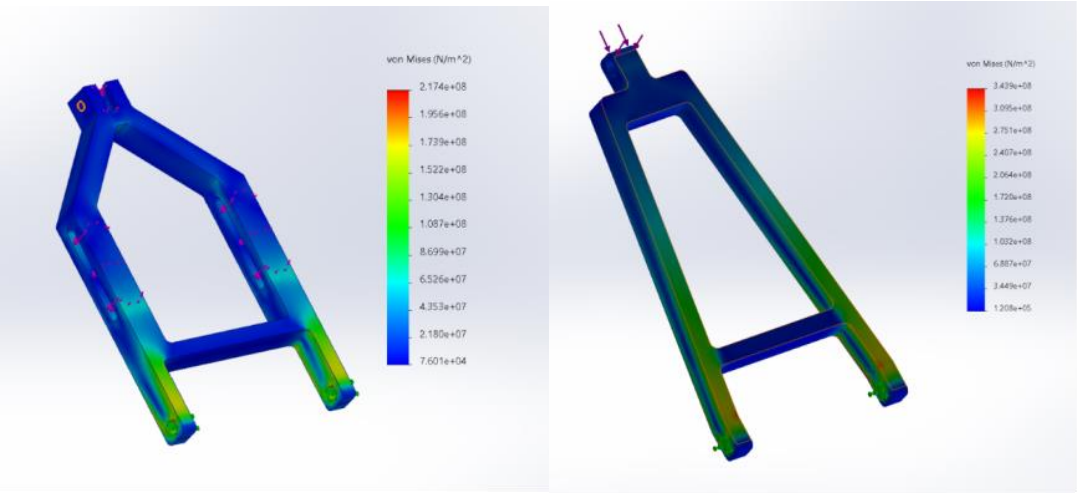

Figure 34. FEA of Linkages

As shown in Figure 34, The FEA results for the mobile robot linkages demonstrate minimal stress levels throughout the components. This suggests that the linkages have sufficient structural integrity to handle the designed weight load (400N) without failure.

The FEA results indicate that the mobile robot platform design is suitable and exhibits promising structural integrity. The analysis shows no deformation within the platform structure and the stress levels remained well below the material's elastic limit as the stress of the majority part are around 6 e +07 N/m^2, which does not exceed the yield strength of PLA (6.748 e +07 N/m^2). Therefore, basically no permanent damage will occur under 400N load applied when the material is PLA. The upper platform design can effectively handle the weight and potential operational stresses from the motors during real-world operation.

In the actual operation, the function of linkage is to act as a supporting structure for force transmission. The pressure is mainly distributed at the end of linkage, which are connected to the lower platform. Therefore, the load near the lower platform of the linkage connector needs to be less than 40 kg to ensure the safe use of the linkage, preventing from deforming or damaging.

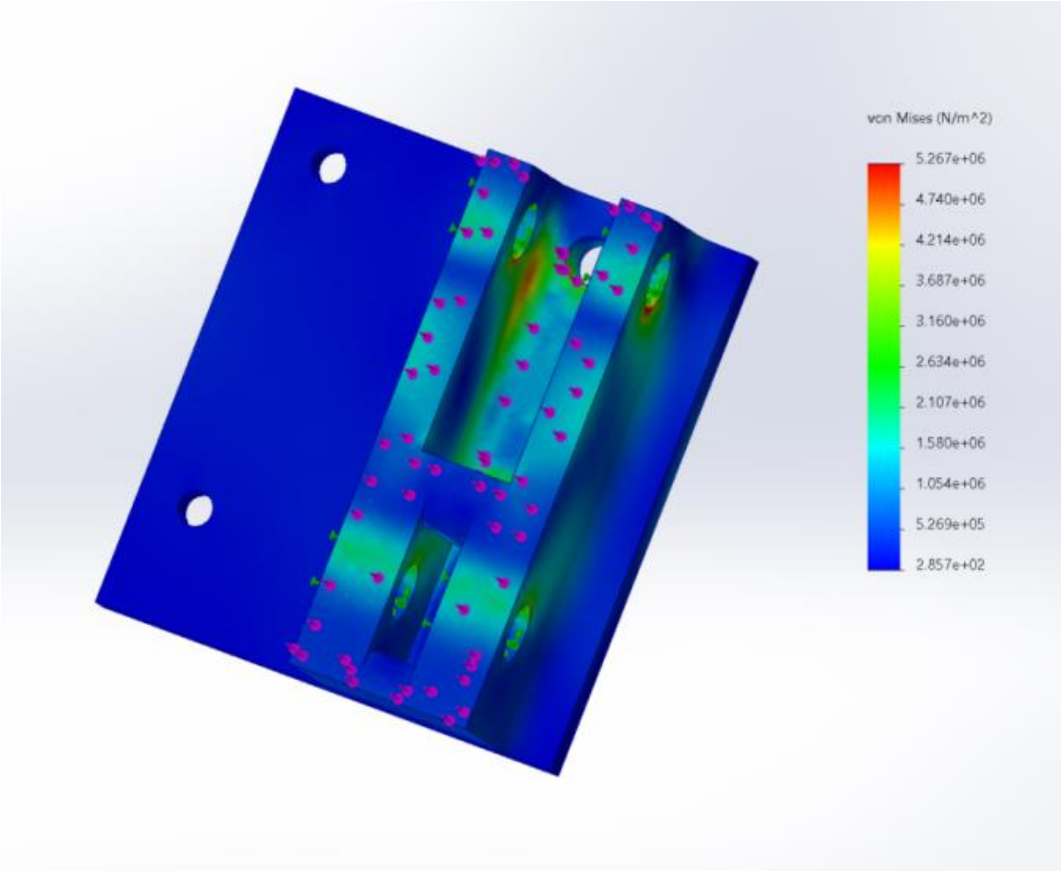

Figure 35. FEA of DC Mounting

Figure 35 shows the FEA of the DC mounting, it is found that the stress distribution of DC mounting performs well under extreme conditions as it holds all the force from the mobile robot body. Therefore, the fatigue test of DC mounting will not be discussed after the FEA analysis.

3.4.2 Fatigue Analysis (FA)

In the simulation of SolidWorks, AMPRES is a tool for evaluating the strength and stability of a model under stress. The load was set 400N, and the material applied is ABS plastic. It helps engineers to determine the reaction of the model when subjected to different loads and conditions for improving the design and ensure it meets safety requirements. Our team wanted to simulate the stability of the whole structure of each part under 400N.

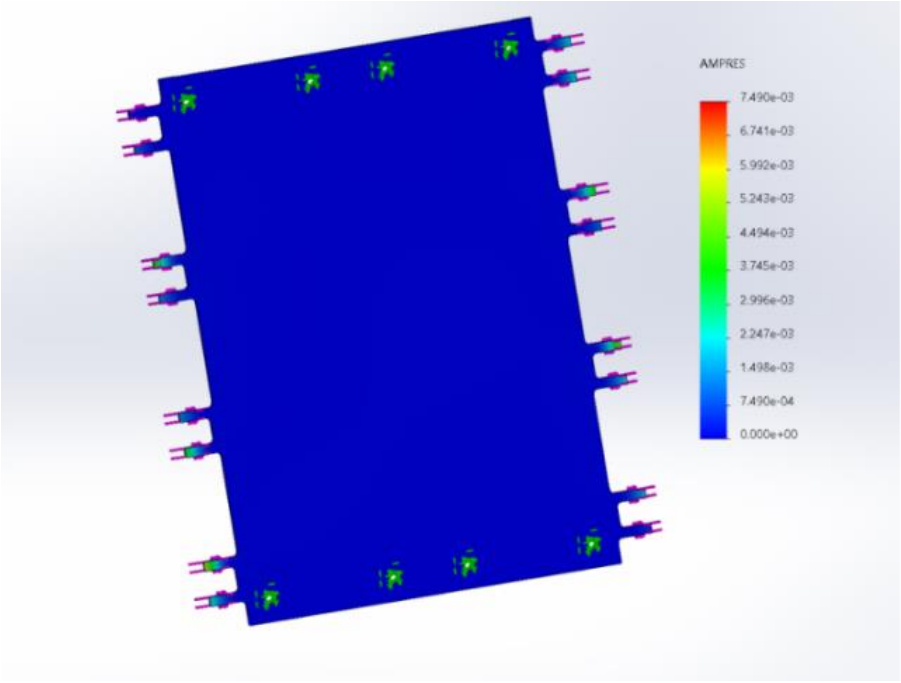

Figure 36. FA of Mobile Robot Bottom Platform

Figure 36 has demonstrated how the structural stability of the bottom platform is affected by the external force of 400N. The green areas in the picture show that the revolute joint for connecting the linkage is the areas where the structure is vulnerable to damage. This is the same conclusion reached from the FEA analysis, it is recommended to concentrate the load of 400N at the middle of the lower platform for safety.

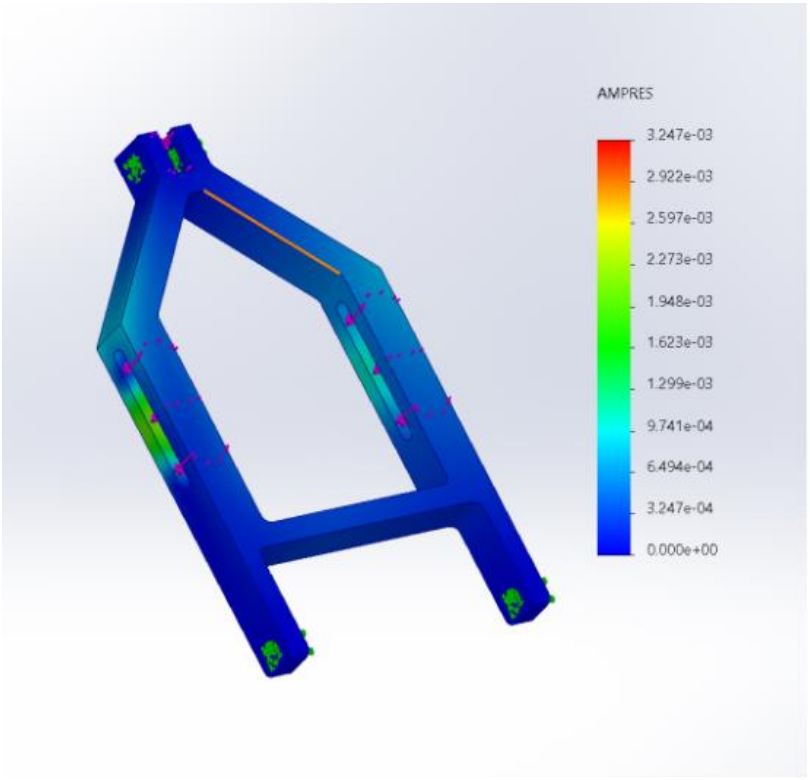

Figure 37. FA of Slot Linkages

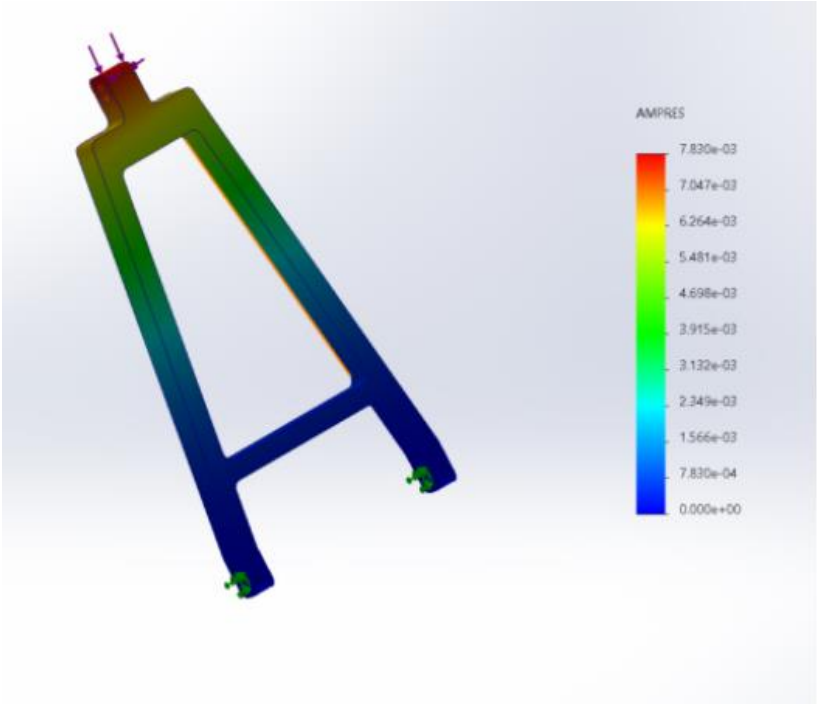

Figure 38. FA of React Linkages

In Figure 37, it demonstrates the stability distribution of the entire structure, the entire structure is stable except for the slot opened in the middle of the linkage in Figure 37. The stepper motor is connected to the slot through mounting for driving the whole active suspension system. The reason for the instability in the slot is due to the thickness of the whole linkage has been reduced for opening the slot for stepper motor. During operation, it is recommended to reduce the rotating speed of the stepper motor to reduce the momentum generated on the slot.

In Figure 38, it demonstrates the improve in stability due to the existence of the spring during compression. However, the connecting part is losing stability as the spring keeps compressing. The solution of the issue is to reduce the speed of the mobile robot controlled by DC motor to avoid over-compression of the spring, resulting in structural instability.

A comparison of Figure 37 and Figure 38 shows that slot linkage and react linkage have different stability in the simulation, but both are affected by motor speed. The speed of the DC motor will affect the stability of the stepper motor and compress the spring to lower the linkage structure. The vehicle speed should be limited for maintaining stability in active control systems.

3.4.3 Damage percentage and product lifetime

To simulate the damage percentage of parts under different usage times. We set up a simulation of damage and product life under 400N and 40N external forces. The frequency cycle was set at 100 times. Setting 400N to simulate the damage to the part under extreme conditions. 40N to simulate damage to the part under normal operation. Comparisons are made to draw conclusions and to evaluate the safety properties of the parts. Based on the simulations, it helps to provide recommendations for improvement. The left side of the figure simulates the scale of the demonstration damage, and the right side simulates how long the product has been in use.

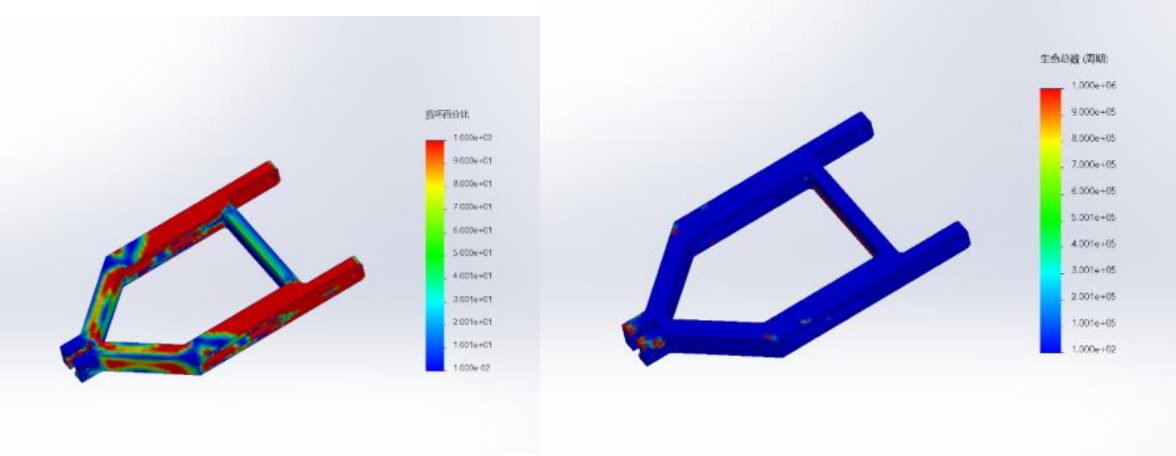

Figure 39. Damage Percentage and Product Life of Slot Linkage (400N)

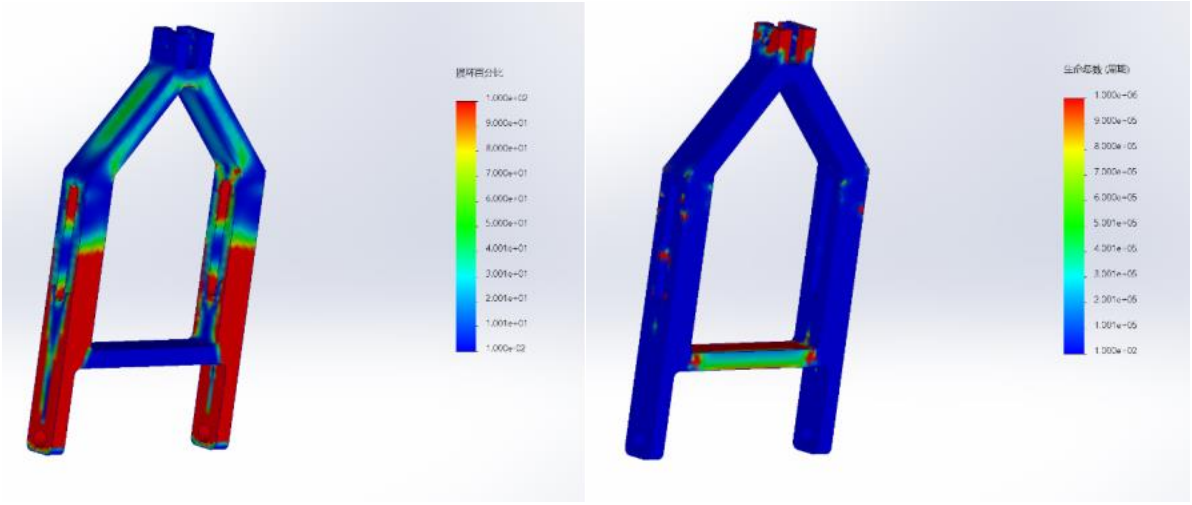

Figure 40. Damage Percentage and Product Life of Slot Linkage (40N)

In Figure 40, it is demonstrated, the percentage of damage to the parts and the use of the parts in the conditions of use. Comparison with Figure 35 leads to the same conclusion. During the life cycle of the product, the load-bearing process is involved in the structural load bearing and slows down the structural damage.

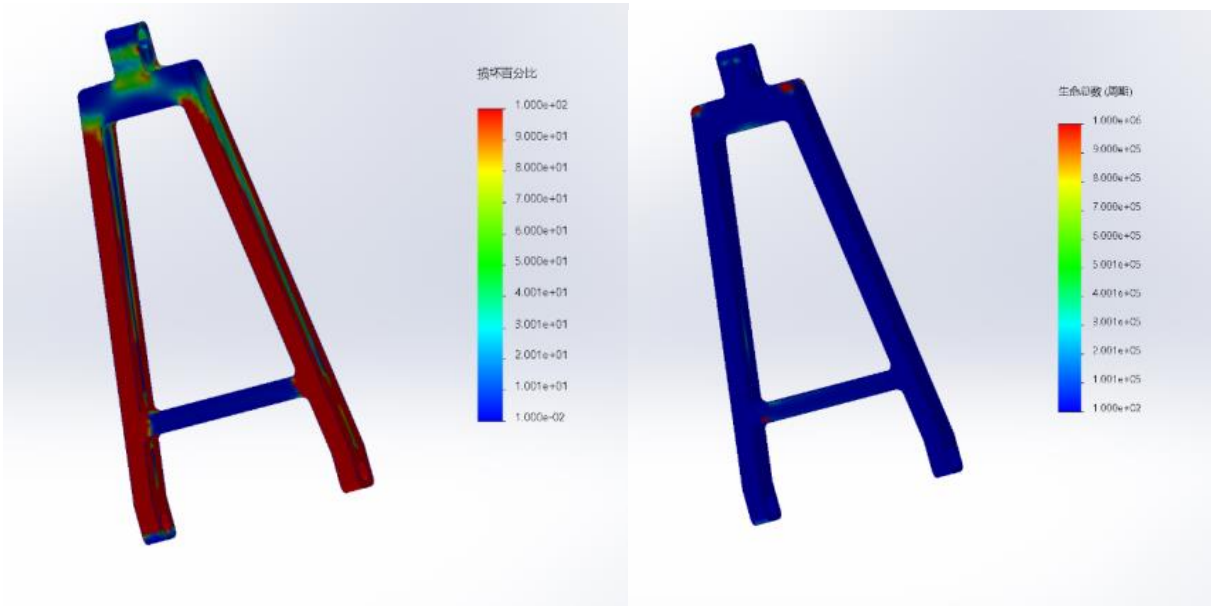

Figure 41. Damage Percentage and Product Life of React Linkage (400N)

The same simulation is used in react linkage. In the extreme case, the main structure receives damage in 10 frequencies. At the same time, the load-bearing structure also maintains functionality in the use process.

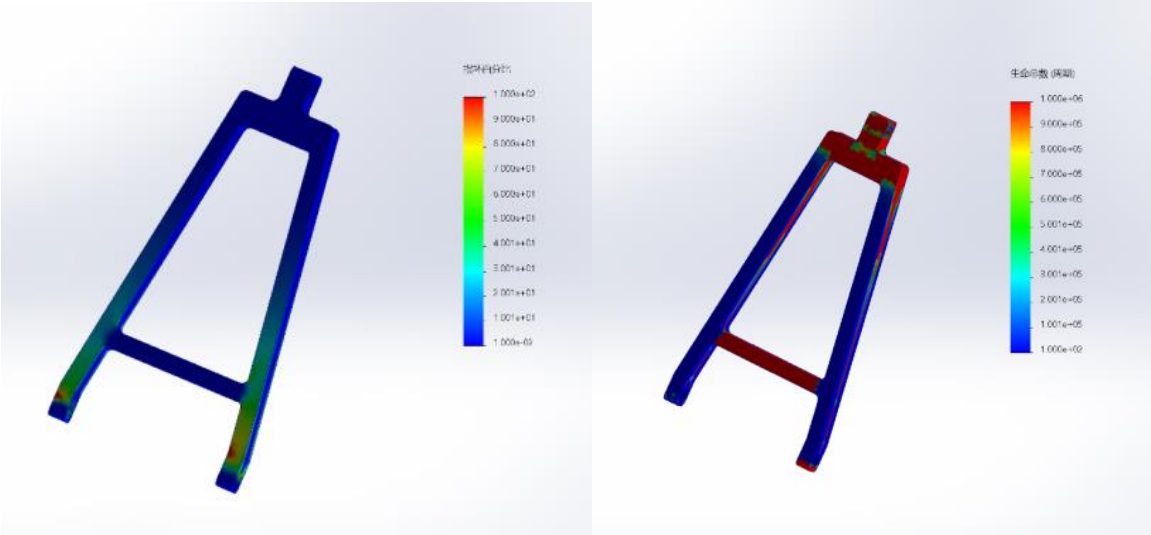

Figure 42. Damage Percentage and Product Life of React Linkage (40N)

In Figure 42, the main and load-bearing structures always remain functional at 100 frequencies under conditions of use and remain undamaged after simulation.

From the analysis of linkage's life cycle and damage, it is concluded that during linkage use, the load-bearing columns act as a pressure disperser and the main structure fulfils the role under the conditions of use. Therefore, our group's design meets the safety requirements.

4. Hardware Components

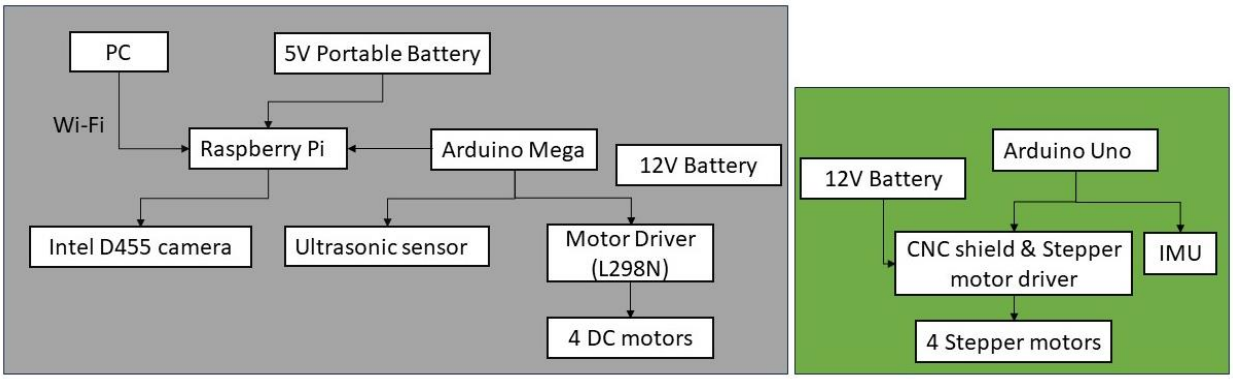

Figure 43. Component Overview

This system is designed to control two different types of motors: stepper motors and DC motors. Arduino Uno controls four stepper motors using data from an Inertial Measurement Unit (IMU). Raspberry Pi processes camera data and integrates it with sensor reading from Arduino to control four DC motors.

4.1 Stepper Motors, CNC Shield, and A4988 Stepper Driver Setup

The ball screw stepper motor is a verity of stepper motor, which fits the need of the active suspension system. By sending specific pulse signals to the ball screw stepper motor, the rotation angle and speed of the motor can be precisely controlled, which in turn controls the vertical motion of robot legs and speed in adjusting the robot legs. This control system enables the mobile robot to maintain balance on uneven terrain.

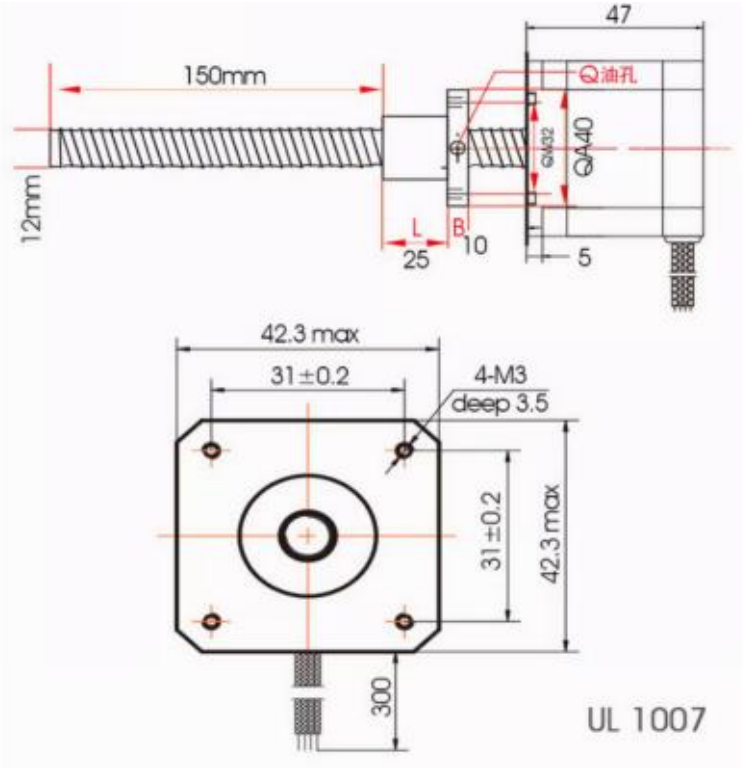

Figure 43. Specification of Lead Screw Stepper Motor

The amount of torque in a stepper motor also varies with the model. By calculation, 42 stepper motors have a holding torque holding torque of 0.05Nm. The advantage of using a stepper motor is that it can be controlled in a precise way, and it can provide holding torque when at rest and control the motion precisely.

i) Torque calculation of stepper motor

$$T_{Raise} = \frac{FD_m}{2}\left(\frac{L + \pi\mu D_m}{\pi D_m - \mu L}\right)$$

$$T_{Lower} = \frac{FD_m}{2}\left(\frac{\pi\mu D_m - L}{\pi D_m + \mu L}\right)$$

Where:

- T_{Raise} is the torque required to raise the suspension (Nm).
- T_{Lower} is the torque required to lower the suspension (Nm).
- F is the loading on the screw (N).
- D_m is the lead mean diameter of the screw (m).
- μ is the coefficient of friction.
- L is the lead of the screw (m).

The mobile robot is assumed to weigh 10kg. As four stepper motors holding the load, the loading on each screw would be 2.5kg. Therefore, the loading on each screw would be 25N.

- Loading on each screw (F): 25 N.

According to the datasheet of the lead screw stepper motor,

- Lead mean diameter (D_m): 12 mm = 0.012 m.
- Lead (L): 1 mm = 0.001 m.
- Coefficient of friction (μ): 0.25.

Substituting the given values into the formulas:

$$T_{Raise} = \frac{25 * 0.012}{2}\left(\frac{0.001 + \pi * 0.25 * 0.012}{\pi * 0.012 - 0.25 * 0.001}\right) = 0.0418Nm$$

$$T_{Lower} = \frac{25 * 0.012}{2}\left(\frac{\pi * 0.25 * 0.012 - 0.001}{\pi * 0.012 + 0.25 * 0.001}\right) = 0.033Nm$$

The lead screw stepper motor has a holding torque of 0.05Nm and therefore, the lead screw stepper motor is suitable for the active suspension system.

ii) Setup

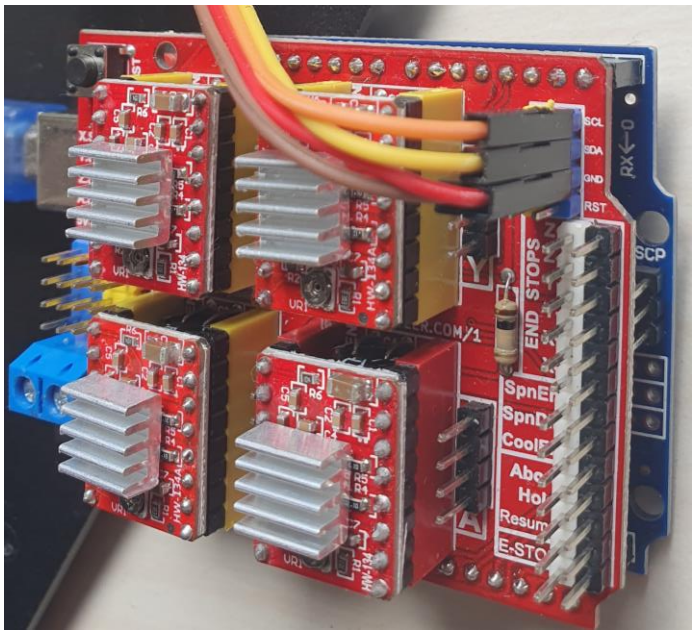

Figure 44. CNC Shield with Four A4988 Stepper Drivers

The **A4988 stepper driver** is a micro-stepping driver that allows precise control of stepper motors. It has an internal current limiting mechanism to prevent motor overheating. This current limit should to be set lower than the motor's continuous current rating to ensure safe operation and prevent overheating. The motor's current rating is 1.7A. The motor needs to run at 80% of its capacity for safety.

$$I_{max} = 1.7 * 0.8 = 1.36A$$

Hence, the safe current limit would be 1.36A.

The A4988 uses a current sense resistor (Rs) to monitor the current flowing through the motor coils. The sensing resistor (R_s) has the value of 0.1 Ω.

$$I_{max} = V_{ref}/(8 * R_s)$$

$$V_{ref} = 1.36 * 8 * 0.1 = 1.088\,V$$

where

- V_{ref} is the reference voltage (V)
- I_{max} is the desired current limit (amps)
- Rs is the current sense resistor value (Ω)

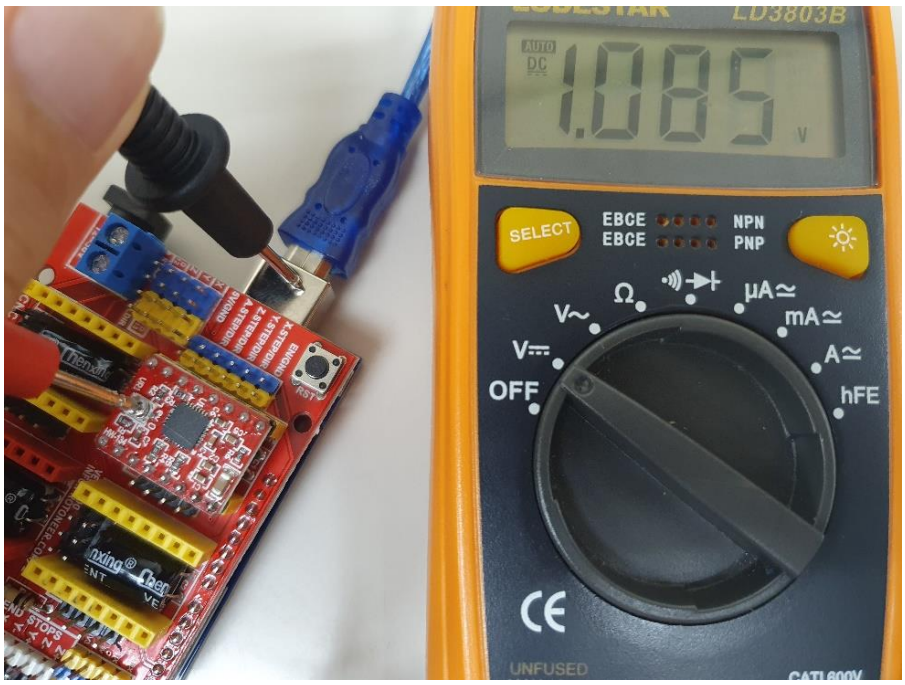

Figure 46. Measurement of Reference Voltage through a Potentiometer

A potentiometer can be used to measure voltage. The current can be adjusted through rotating the screw on the A4988 driver.

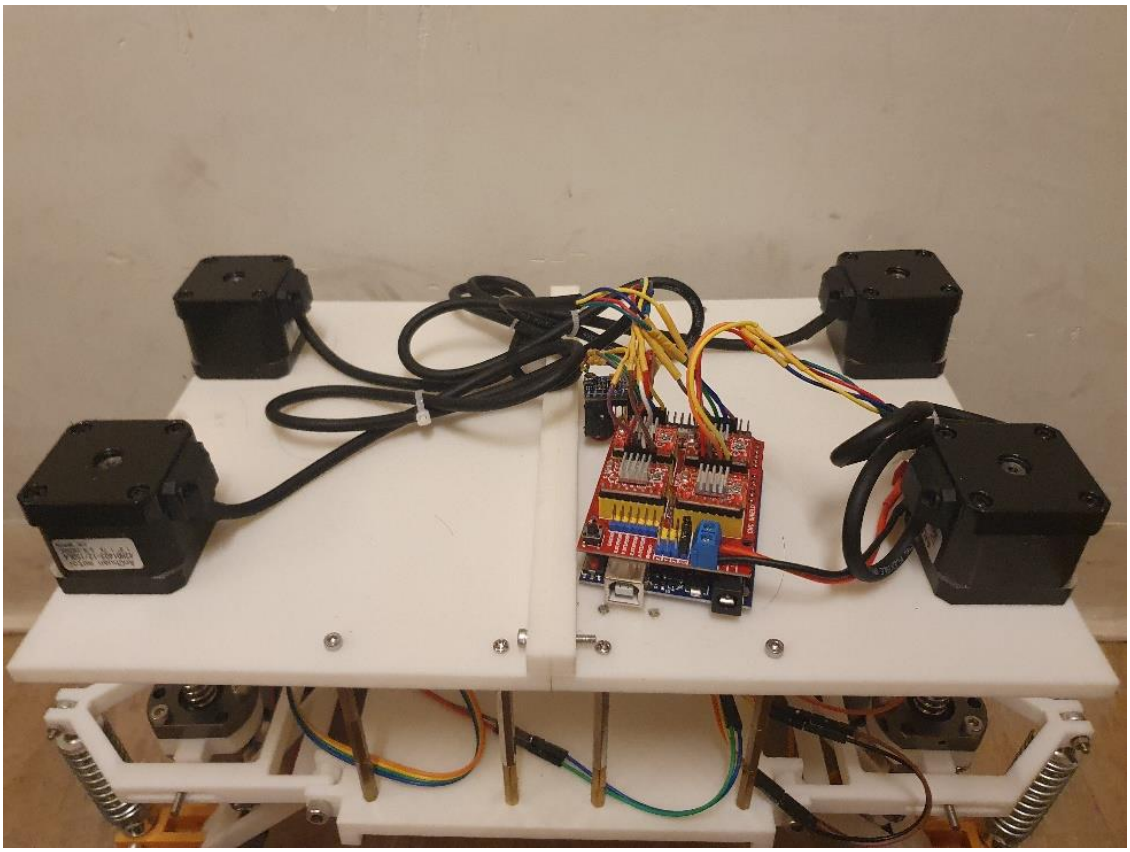

Figure 47. Stepper Motors Setup

The four stepper motors are securely mounted on the upper plate of the robot chassis. Each motor is connected to its respective A4988 stepper driver, which is housed on the CNC shield. This setup ensures precise control and coordination of the motors' movements. Additionally, the CNC shield provides a centralized interface for connecting and managing the motor drivers, simplifying the overall hardware configuration of the robot.

4.2 DC motor & motor driver

The DC Motor serves as power source for mechanical devices and is connected to the wheel to propel the mobile robot forward. By connecting the module with sensors, controller boards, and a motor shield, it allows the autonomous navigation of the mobile robot. The motor converts the electrical energy from battery into mechanical rotation to drive the wheel.

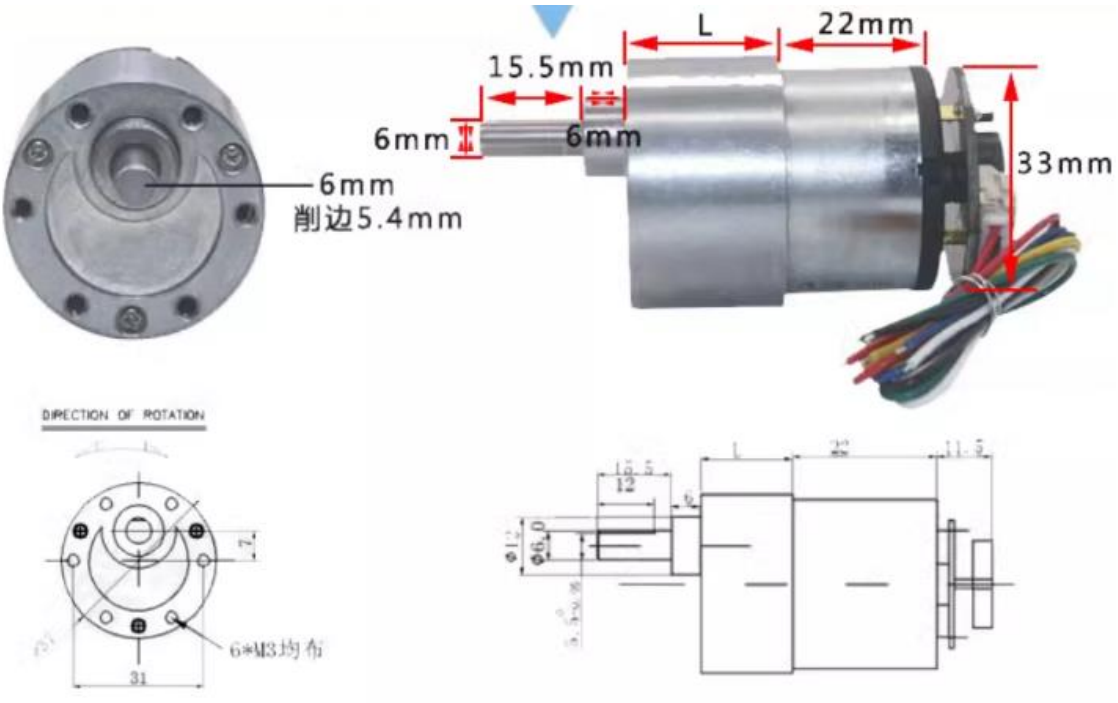

Figure 48. Specification of DC motor

The JGB37-520 DC motor was chosen for its ability to provide high-speed controllable rotation to drive the mobile robot forward. It provides 1Nm with a reduction ration of 1:10 under 12V.

i) Driving Torque Calculation

$$T = \mu \cdot m \cdot g \cdot r$$

Where:

T is the driving torque (Nm).

- μ represents the friction factor between the tires of the mobile robot and the ground.
- m represents the total mass of the mobile robot (kg).
- g represents the acceleration due to gravity (m/s²).
- r represents the radius of the tires of the mobile robot (m).

For our scenario, the following values are used:

- Friction factor (μ): 0.68 (representing the coefficient of friction of the dry concrete road surface).
- Total mass (m): 10 kg.
- Acceleration due to gravity (g): 9.81 m/s² (standard value for Earth's gravity).
- Radius of the tires of the mobile robot (r): 37.5mm = 0.0375m.

$$T = 0.68 \cdot 10 \cdot 9.81 \cdot 0.0375 = 2.5Nm$$

$$T_{motor} = \frac{T}{no.\,of\ motors} = \frac{2.5}{4} = 0.625Nm$$

The driving torque of 2.5NM is required and each DC motor has to provide 0.625Nm. As the JGB37-520 DC motor can provide 1Nm, it is suitable for mobile robots.

ii) Setup

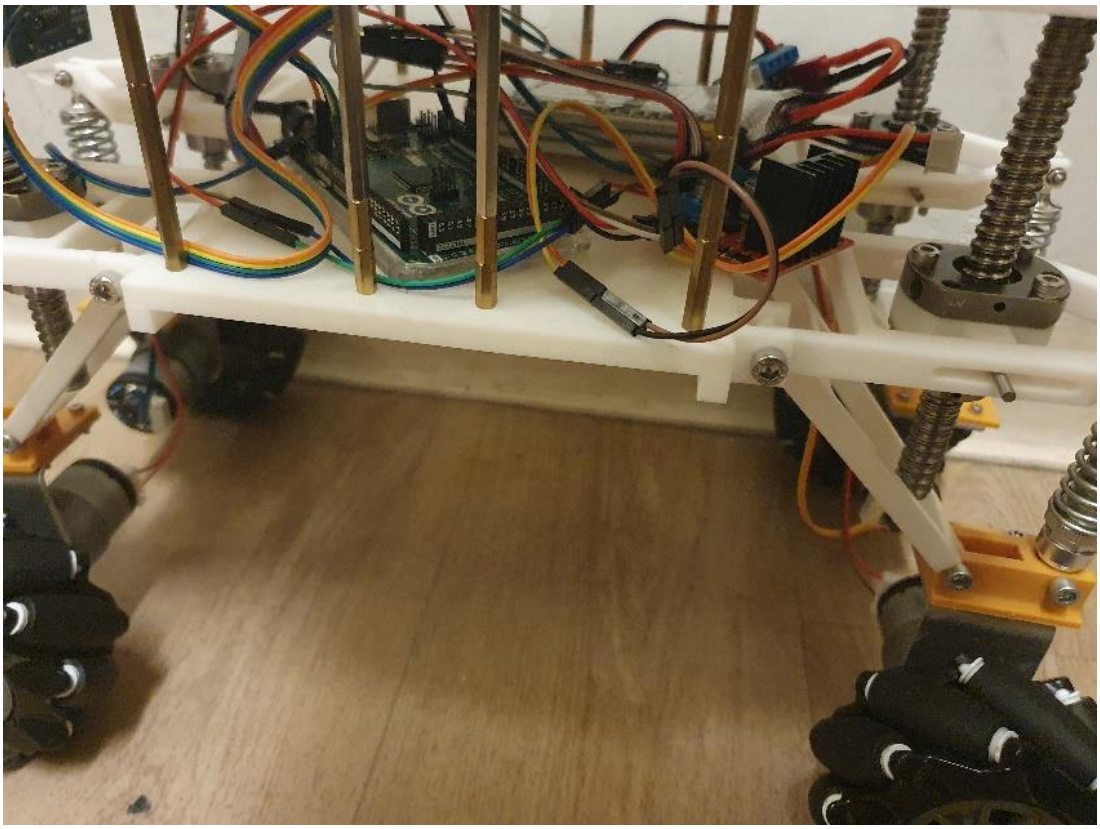

Figure 49. Four Motors with L298N Motor Driver

An L298N Motor Driver is used to control DC motors. It receives control signals from an Arduino board and provides necessary voltage and current to the motor. It allows the microcontroller to control the direction and speed of the DC motors. The motors can rotate in both directions (forward and backward) by controlling the polarity of the voltage applied to the motor.

The motors are mounted onto the L mount, ensuring stability on the mobile robot's chassis. This configuration reduces vibrations and movement during operation. This can enhance motor performance and lifespan.

4.3 Depth Camera

Figure 50. Intel RealSense Depth Camera D455

RGB-D cameras, also known as depth cameras, can provide depth information to create a depth map of the environment. It consists of two cameras placed a distance apart to operate like the visual systems of human. It is crucial in performing tasks including obstacle detection and navigation. It provides a depth map of the environments where color represents the distance from the sensor. Intel RealSense Depth Camera D455 will be used for artificial marker detection.

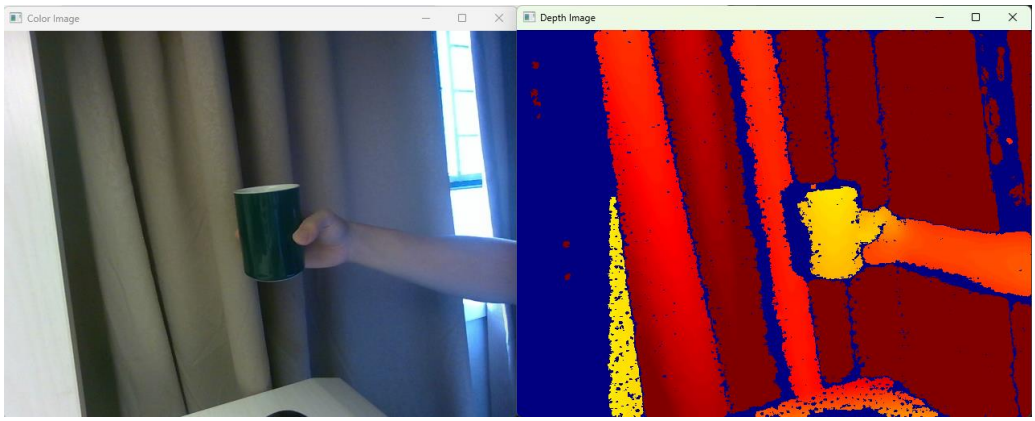

Figure 52. Depth Image

4.4 IMU sensor

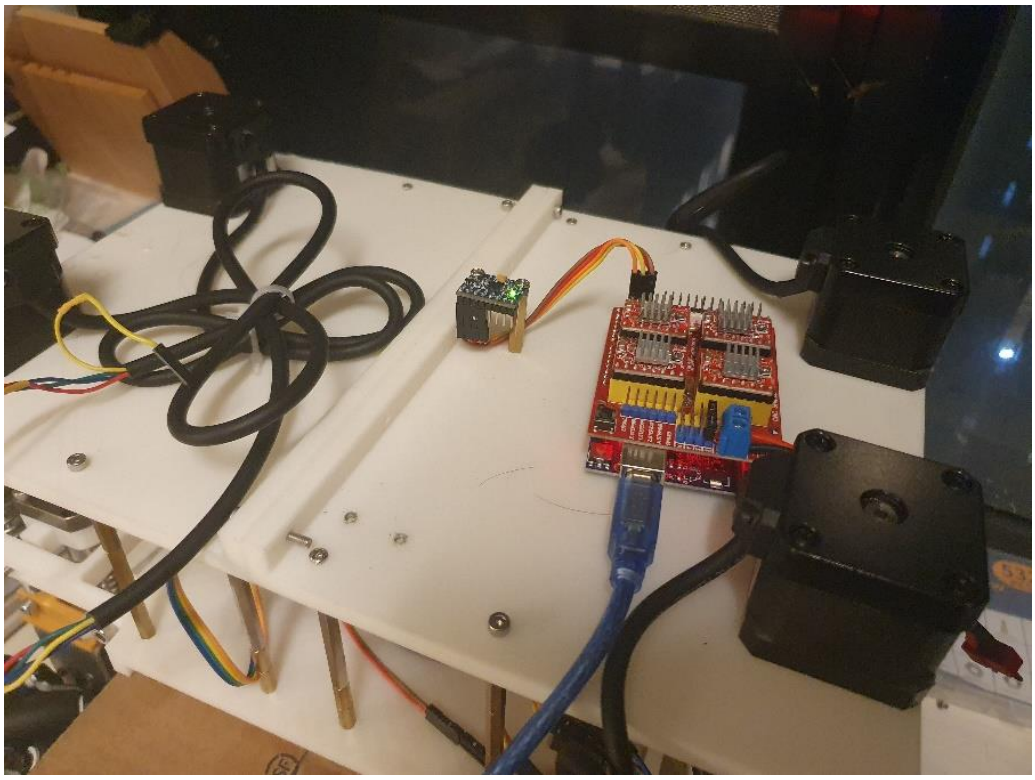

Figure 52. MPU6050 Sensor

The MPU6050 sensor is an Inertial Measurement Unit (IMU), which combines a gyroscope and accelerometer to determine a device's motion and orientation. An MPU6050 has a 3-axis gyroscope to measure rotation and 3-axis accelerometer to measure acceleration. By measuring the tilt angle on the X and Y axes and the rotation angle on the Z axis, the robot’s orientation can be determined.

4.5 Ultrasonic sensor

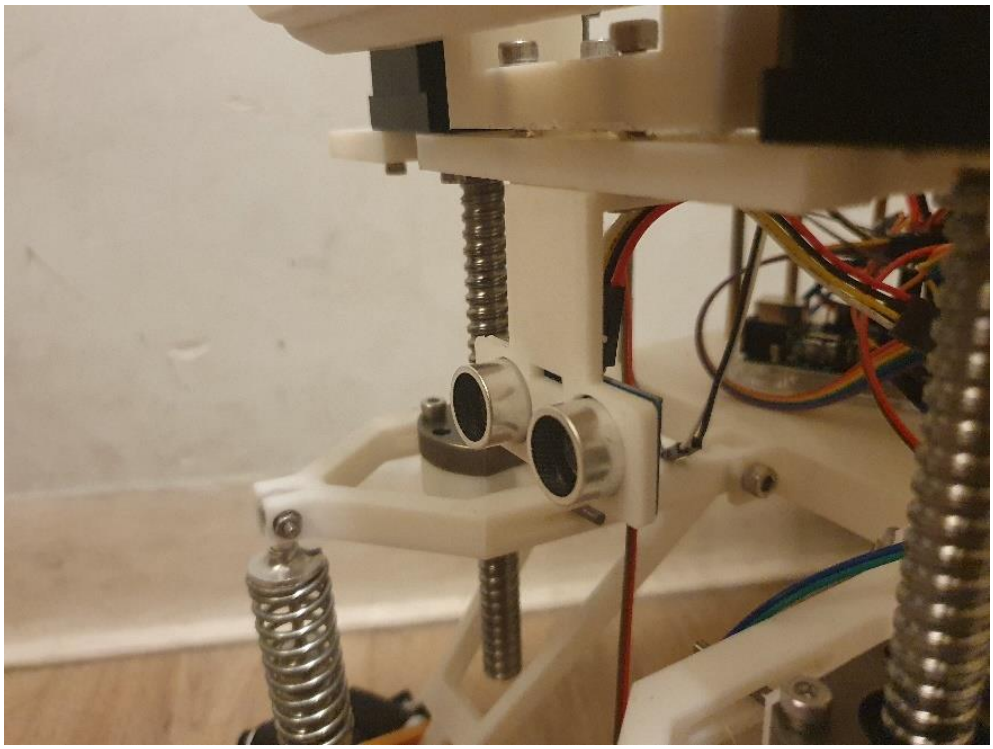

Figure 53. Ultrasonic Sensor

Ultrasonic sensors emit ultrasonic waves and determine the distance of an object by measuring the time it took for the soundwave to return to the sensor. This sensor can help the robot in detecting obstacles and prevent collisions. It is useful in an environment where visibility conditions are challenging such as foggy or rainy days.

4.6 Wheel selection

Shabalina et al. [20] conducted a comparative analysis of mobile robot wheel designs based on general, mechanical, and technical parameters of wheels.

Table 1. General, Mechanical and Mechanical Parameter of Wheels

Type of a wheel	Sensitivity to a rough surface	Sensitivity to small objects on a surface	Degree of Freedom (DoF)	Maximum load
Conventional	Low	Low	1	Up to 40-60 kg
Steering	Low	Low	2	Up to 40-60 kg
Caster Wheel	Low	Low	1, 2, 3	15 kg
Caster Ball	Low	High	1, 2, 3	15 kg
Universal Omni	High	High	3	2-30 kg
Mecanum	High	High	3	7–15 kg

In our project, the mobile robot with active suspension system is required to traverse uneven terrain with obstacles. The wheels are required to have lower sensitivity to rough surfaces and small objects on a surface if running in an actual uneven terrain with different size of obstacles for a real test. However, the test of mobile robot will be carried out in the campus or the laboratory so sensitivity to rough surfaces and small object on a surface do not need to be considered in this case.

Table 2. Testing of wheels

Type of a wheel	Move Forward	Rotation
Universal Omni	O	X
Conventional wheel	O	X
Macanum wheel	O	O

These three wheels are tested for basic movement. While the other wheel types were considered, the Mecanum wheels achieved efficient rotation due to their angled rollers.

4.7 Controller Boards

Table 3. Comparison of Controller Boards

Controller Board	Raspberry Pi 4	Jetson Nano Developer Kit	Orange Pi 5 Plus
			
Price (HKD)	700	1300	900
CPU	Quad-core BCM2838B0 @1.5GHz	Quad-core ARM A57 @1.43 GHz	Octa-core RK3588 @2.4GHz
Memory	8GB	4GB	8GB
Size	85mm x 56mm	69mm x 45mm	75mm x 100mm

Jetson Nano Developer Kit is more expensive and the specification of it is similar to Raspberry Pi 4 with a slightly smaller size. Similarly, Orange Pi 5 Plus has a better performance with a higher price. By comparing the three different controller boards, the Raspberry Pi 4 is adopted to be the controller board to be used in this project.

5. System Design

5.1 Active suspension System

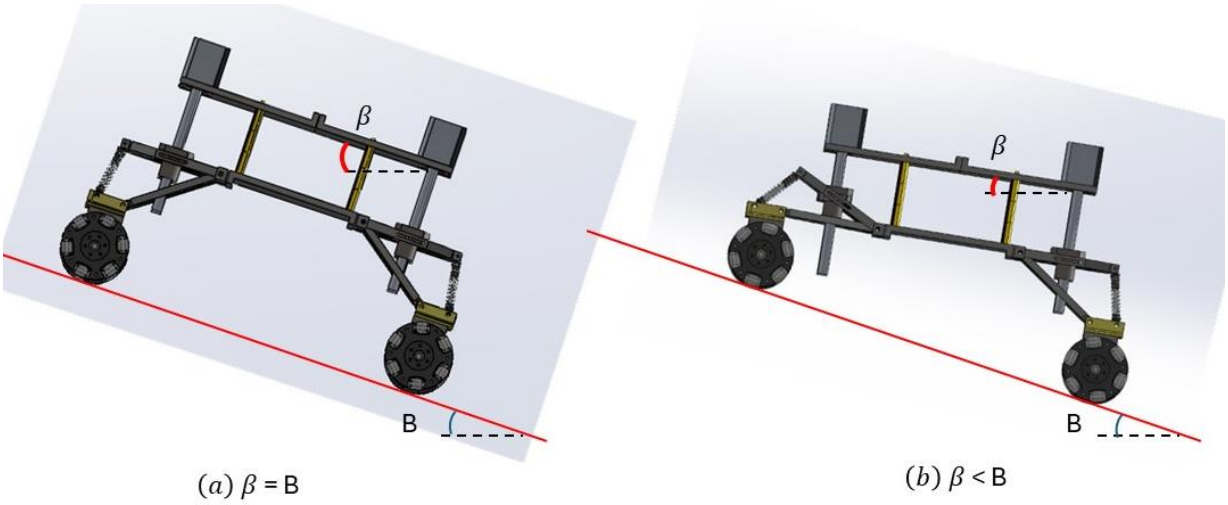

Figure 54. Slope Compensation

The active suspension can be utilized for slope compensation. B denotes the slope surface inclination angle and β is the upper plate angle. By actively adjusting the lengths of the suspension actuators, the mobile robot can tilt the upper body in the opposite direction of the slope. This strategic tilt effectively lowers the β angle, allowing the robot to be better oriented.

5.1.1 Angles from IMU

The MPU6050_light library is utilized to obtain roll, pitch, and yaw angles from the IMU (Inertial Measurement Unit). Additionally, the wire library is utilized for I2C communication with the MPU6050 sensor.

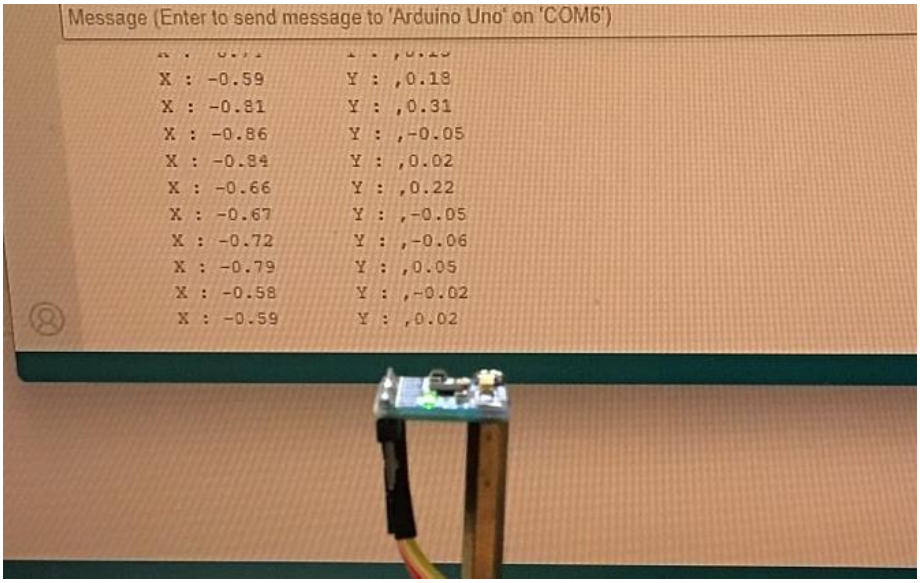

Figure 55. X and Y Angles

It can detect motion and orientation changes in both the X and Y directions within a range from -180 degrees to 180 degrees.

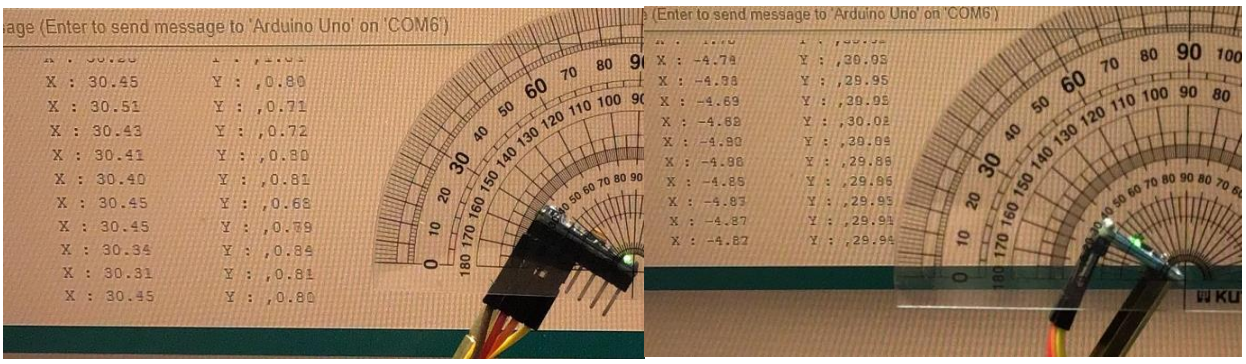

Figure 56. 30 Degrees Test

The angles of 30 degrees were tested to validate the accuracy of the IMU readings. The IMU provided correct and reliable angle measurements at the specified 30-degree inclination. This ensures the precision of the IMU data. It is crucial for the accurate performance of the active suspension system.

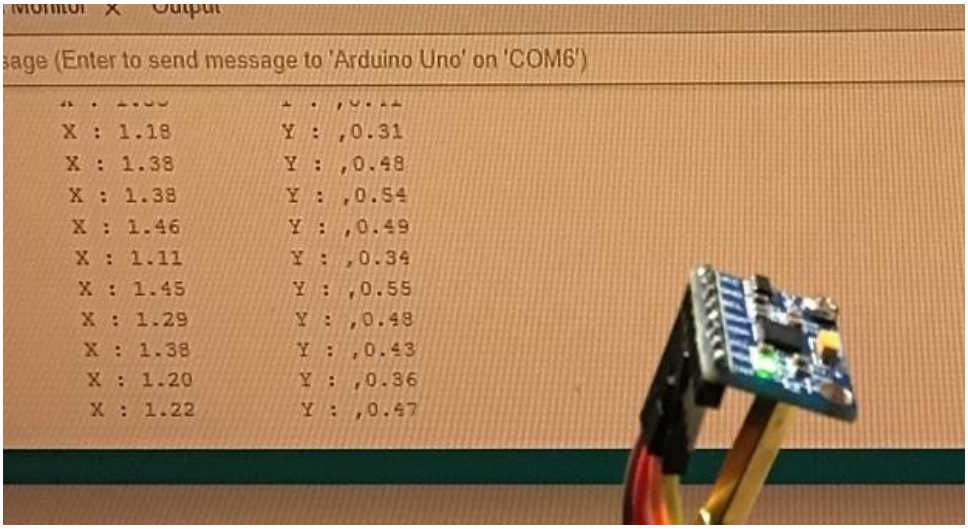

Figure 57. Offset Calibration

Offset calibration is accomplished on the IMU using the mpu.calcOffsets() function to achieve optimal performance. This process addresses inherent biases within the accelerometer, which can lead to slight inaccuracies in angle measurements. By setting all angles to 0 upon initialization, the control system gains the ability to calculate changes in angles from their initial values. This can be used when the robot needs to maintain a specific orientation.

5.1.2 Control System and IMU integration

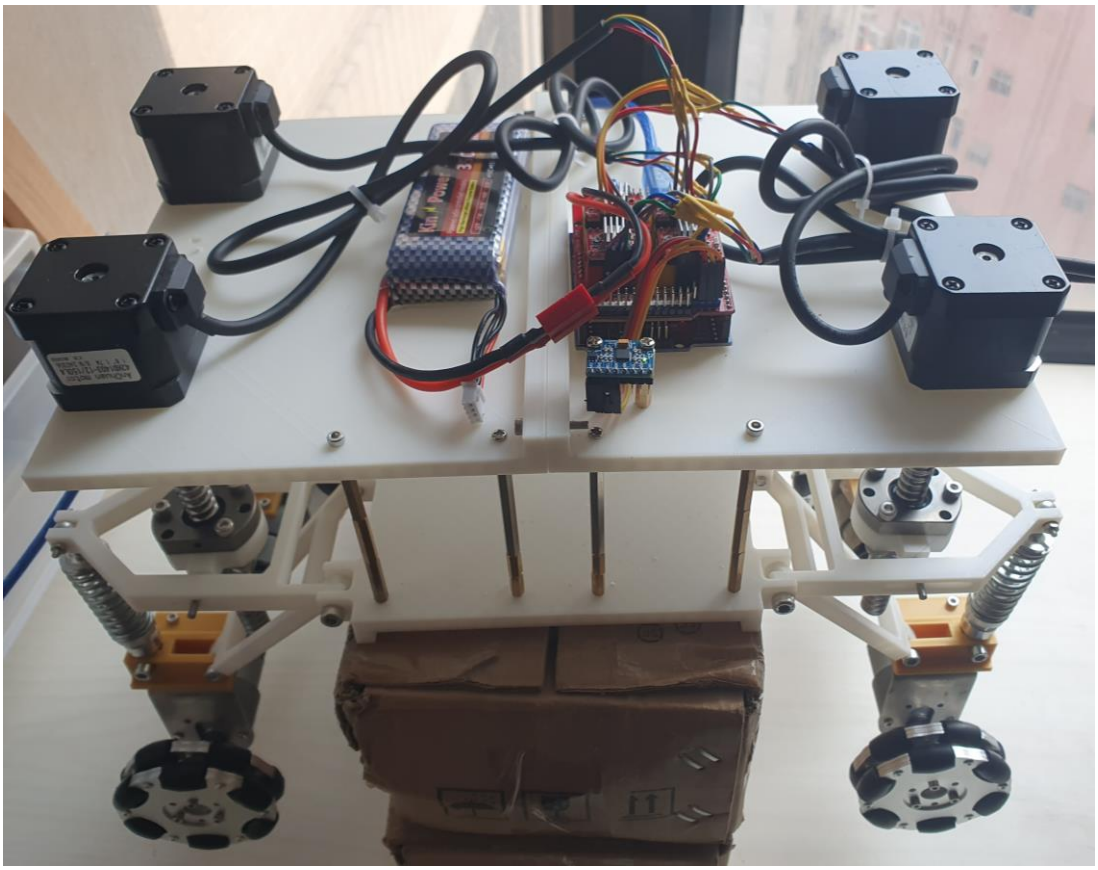

Figure 58. Active Suspension System

A balancing algorithm is implemented within the control system. This algorithm processes the real-time IMU data, focusing on the roll (X-axis) and pitch (Y-axis) angles. Based on the deviation of the angles, the stepper motor moves in a direction to keep the robot upright.

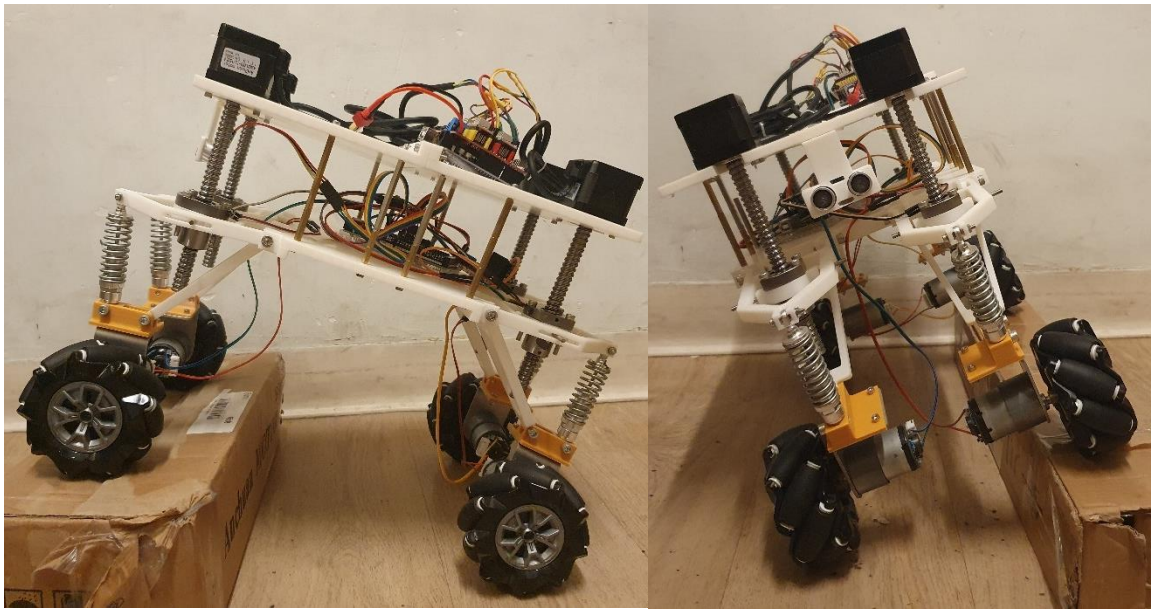

Figure 59. Positive X Angle (Left) Positive Y Angle (Right)

Based on the orientation of the robot, the positive or negative angle on X and Y are given.

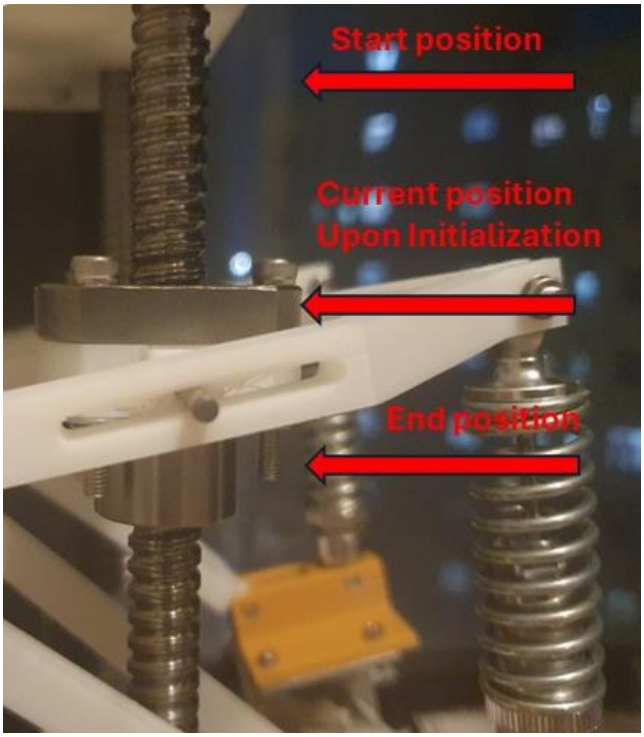

Figure 60. Range of the Motor Movement

The four stepper motors (motorX, motorY, motorZ , and motorA) operate within a safe zone. The ‘startPosition’ and ‘endPosition’ arrays are used to store the initial and final positions of the four stepper motors relative to their current location at startup. The startPosition values are calculated by adding a constant value of 1000 to the initial position of each motor. The endPosition values are calculated by subtracting another constant value of 1000 from the current position of each motor. A virtual boundary is created for each motor’s movement. This boundary prevents the ball screw from running off the lead screw in stepper motors and tounching the top plate.

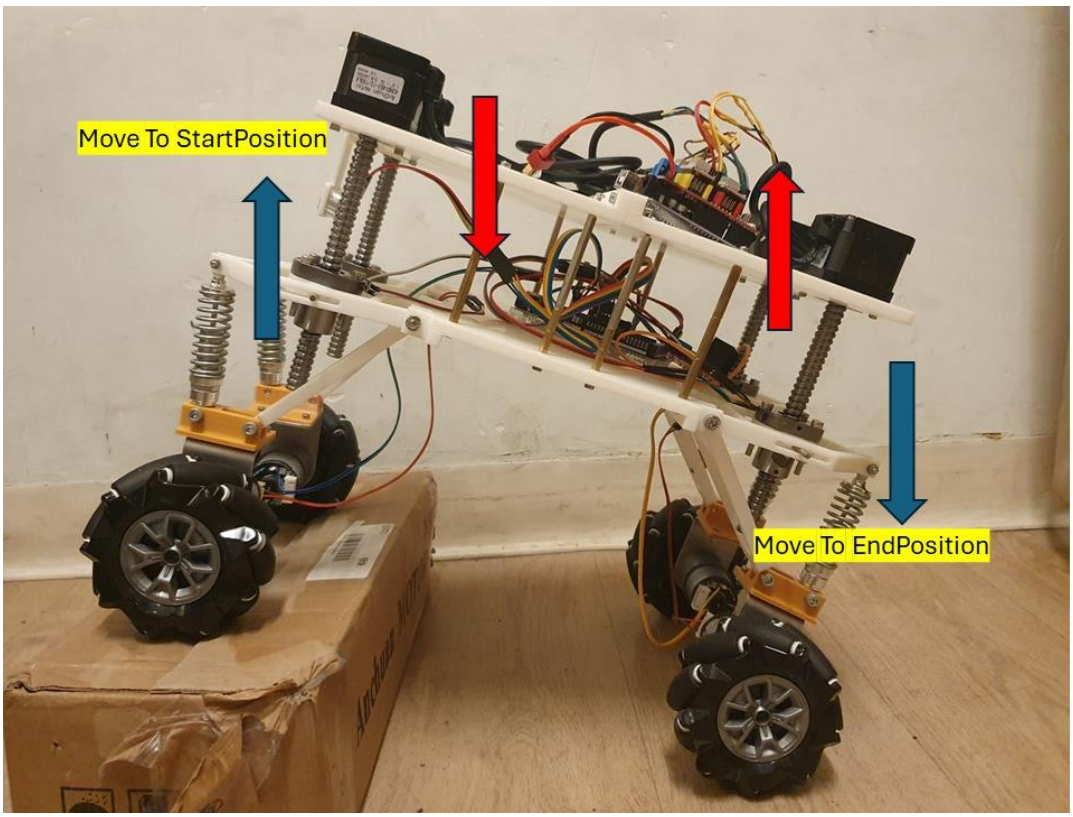

Figure 61. Motor Direction

When the X angle is positive, the motors controlling the front legs (Motor X and Motor Y) should move to the start position. The motors controlling the back legs (Motor Z and Motor A) should move to the end position. This movement combination counteracts the forward tilt.

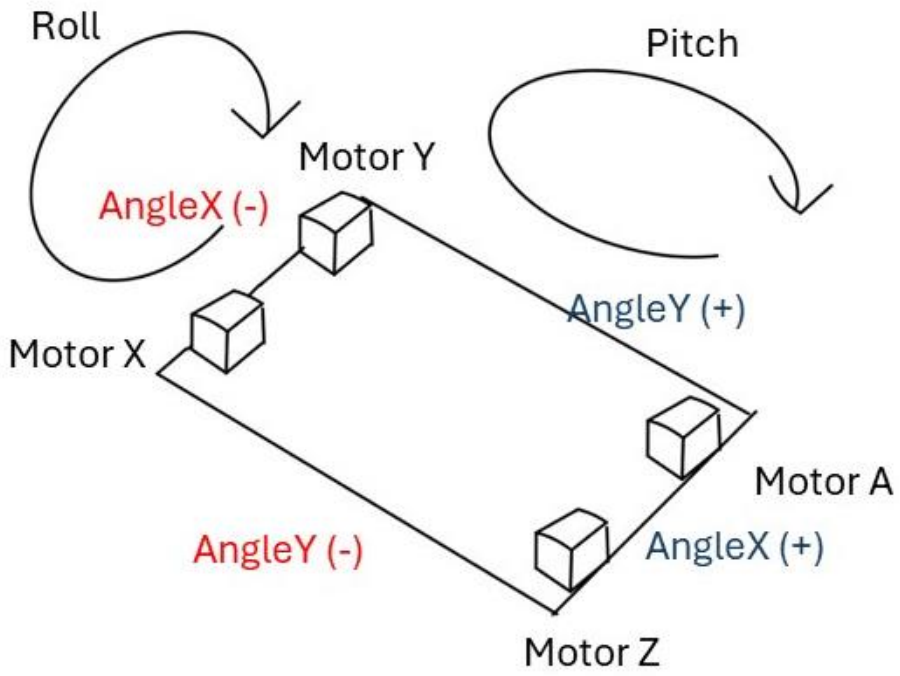

Figure 62. Motors and XY Angles

A positive X-axis angle indicates the robot is tilted downward on the back side. Conversely, a negative X-axis angle means a tilt downward on the front side. Similarly, a positive Y-axis angle indicates the robot is tilted downward on the right side. A negative Y-axis angle is a tilt downward on the left side. Using this information about the tilt angles, the four stepper motors are controlled to achieve balance.

Table 4. Tilt Angle and Motor Control

IMU angles		Stepper Motors			
X	Y	X	Y	Z	A
(+)	(+)	Start Position			
(+)	(-)		Start Position		
(-)	(+)			Start Position	
(-)	(-)				Start Position
(-)		Start Position	Start Position	End Position	End Position
(+)		End Position	End Position	Start Position	Start Position
	(+)	Start Position	End Position	Start Position	End Position
	(-)	End Position	Start Position	End Position	Start Position

This table refers to the logic behind the adjustments. It considers the direction of the robot's tilt and determines the appropriate motor movements for each leg. When the motor goes to the start position, it pulls the platform downward on the motor's side. Moving to the end position pushes the platform upwards on the motor's side. By obtaining the tilt angles and activating the appropriate motors based on the table, the system can achieve balance.

5.2 Navigation System

5.2.1 Aruco Marker Detection

Aruco markers are special square-shaped patterns with unique codes that can be easily identified by computer vision algorithms. The code utilizes a pre-defined Aruco dictionary to identify specific marker patterns. This allows the robot to not only detect the presence of a marker but also determine its unique ID.

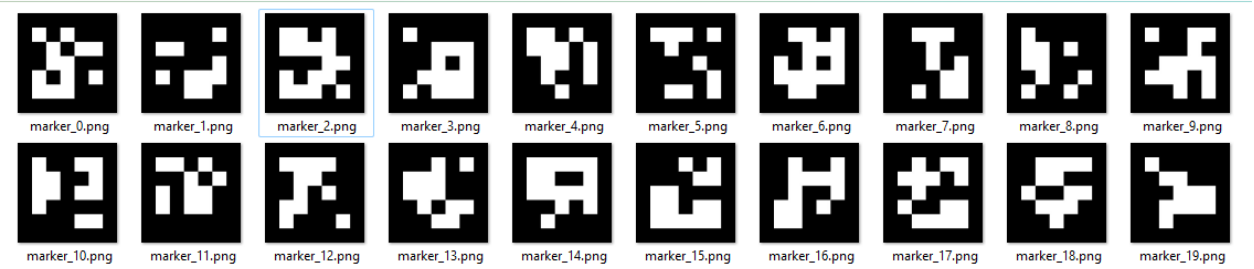

Figure 63. Generated Aruco Markers

The Aruco markers were generated using Aruco marker generation code.

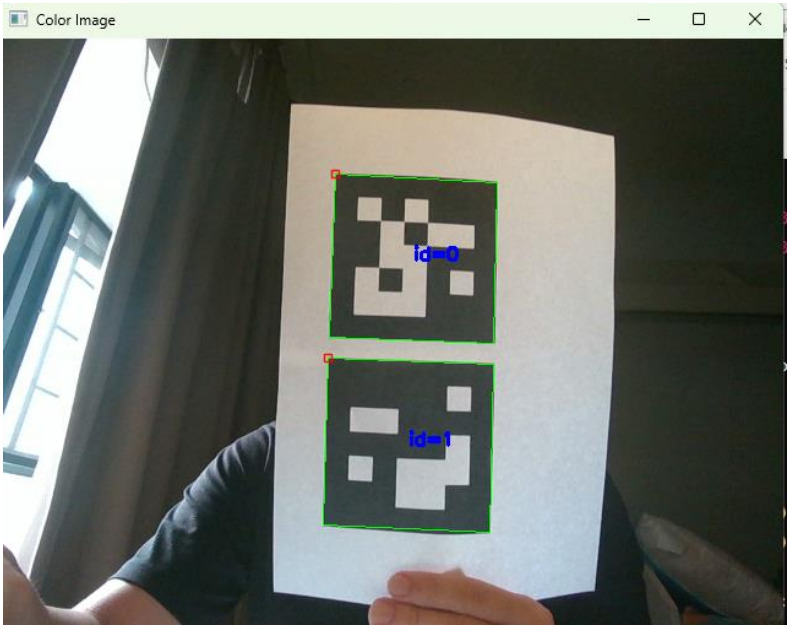

Figure 64. Marker Detection in Color Image

The Aruco makers are identified within the captured color frame. The patterns matching the chosen marker are searched and return the corner points and marker IDs.

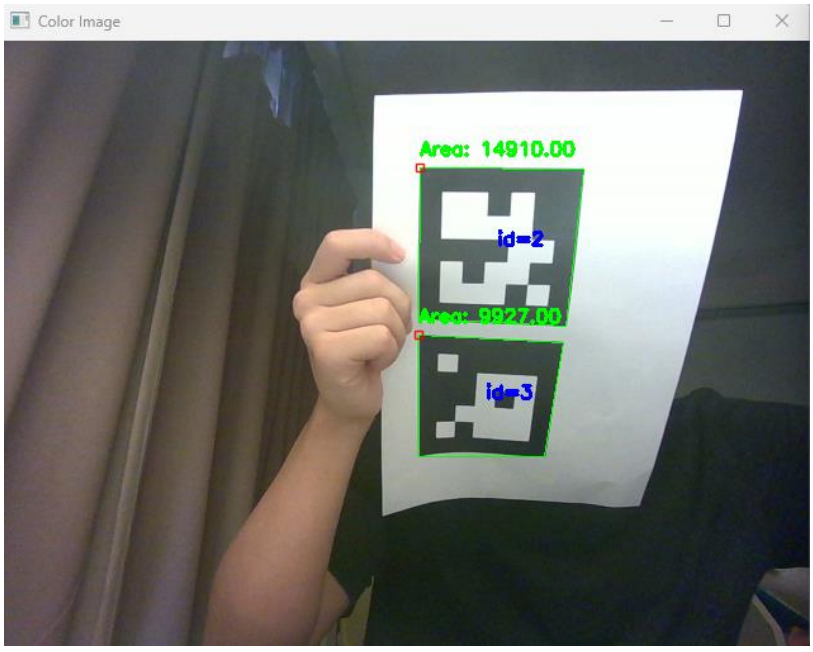

Figure 65. Area Calculation

The area of each detected marker is calculated and is displayed next to the marker.

By identifying specific markers and their corner points within the camera frame, their exact location relative to the camera can be determined. This can be valuable for robot navigation as it allows the robot to precisely plan its movement accordingly. Additionally, marker-based actions enable pre-programmed responses based on the detected marker. This combination provides flexible control and interaction with the environment based on marker placement.

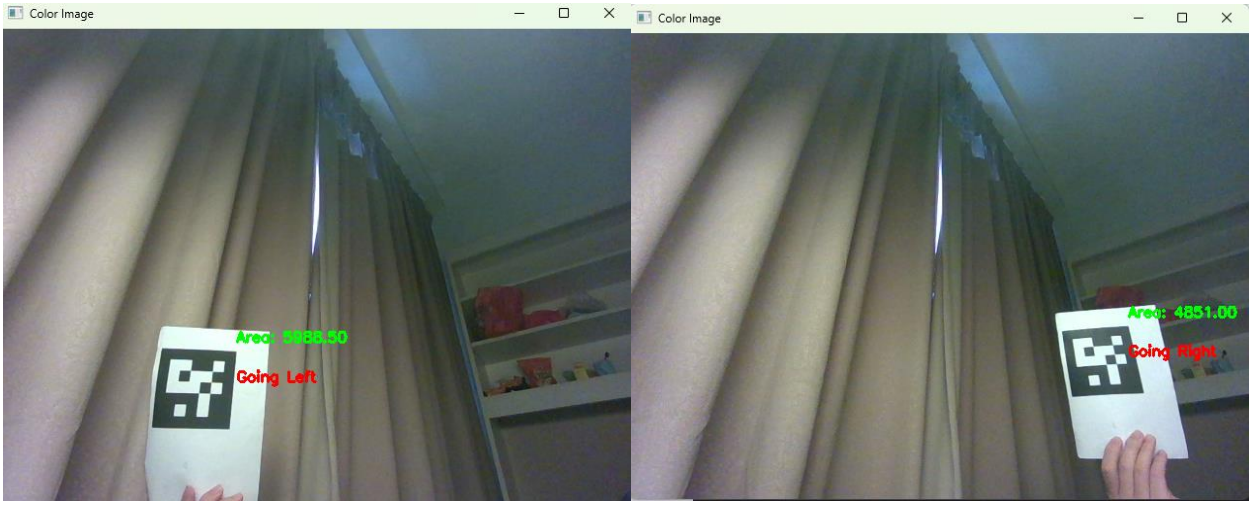

Figure 66. Go Left or Right Command

If the marker appears larger (or closer) on one side, the robot increases the speed of the wheel on the opposite side. For instance, if the marker is larger on the right side, the robot would slow down the right wheel and speed up the left wheel to turn left until the marker appears centered in the camera view.

5.2.2 Marker-based navigation

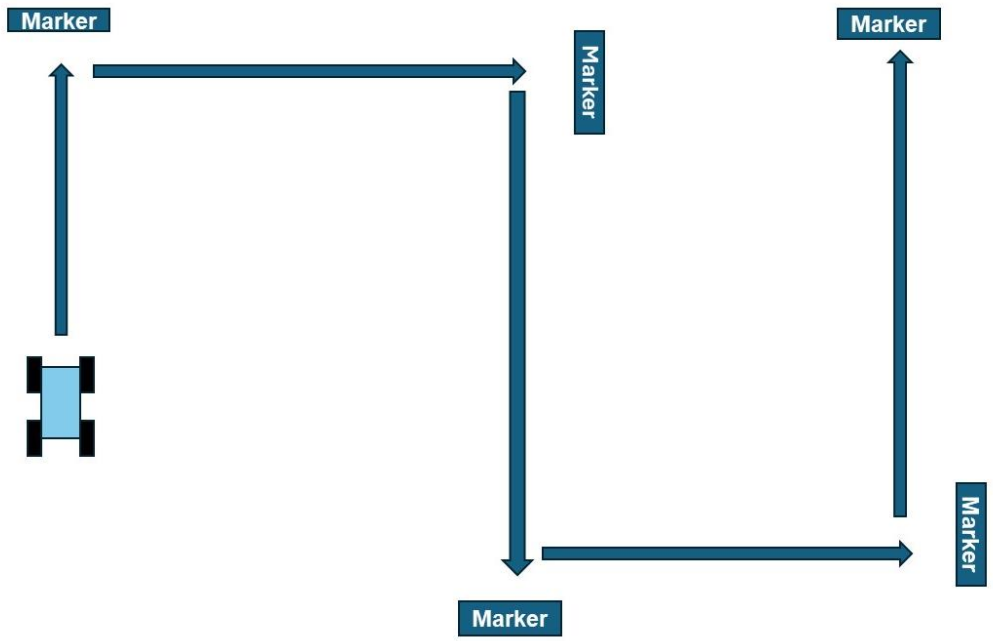

Figure 67. Marker-based Robot Navigation System

By identifying specific markers and their corner points within the camera frame, their exact location relative to the camera can be determined. The robot's navigation relies on the detection and interpretation of Aruco markers placed strategically within the environment. If a marker with the desired ID is found, it goes forward until the area is enlarged to a set value. If the detected area meets the set value (the robot has reached the marker), it rotates until it detects the next marker. Then it goes forward until the area meets the value. If the marker is off-center, the robot turns left or right to align with the marker and proceeds in the intended direction. If the desired marker is not found, it rotates for 8 seconds to search for a lost marker. This process continues until the robot reaches the final destination marker (final marker).

5.2.3 Sensor Fusion

The ultrasonic sensor is integrated to enable robots to detect distance and provide more accurate information. Data provided with these sensors are used to make decisions on its movement via path planning algorithms.

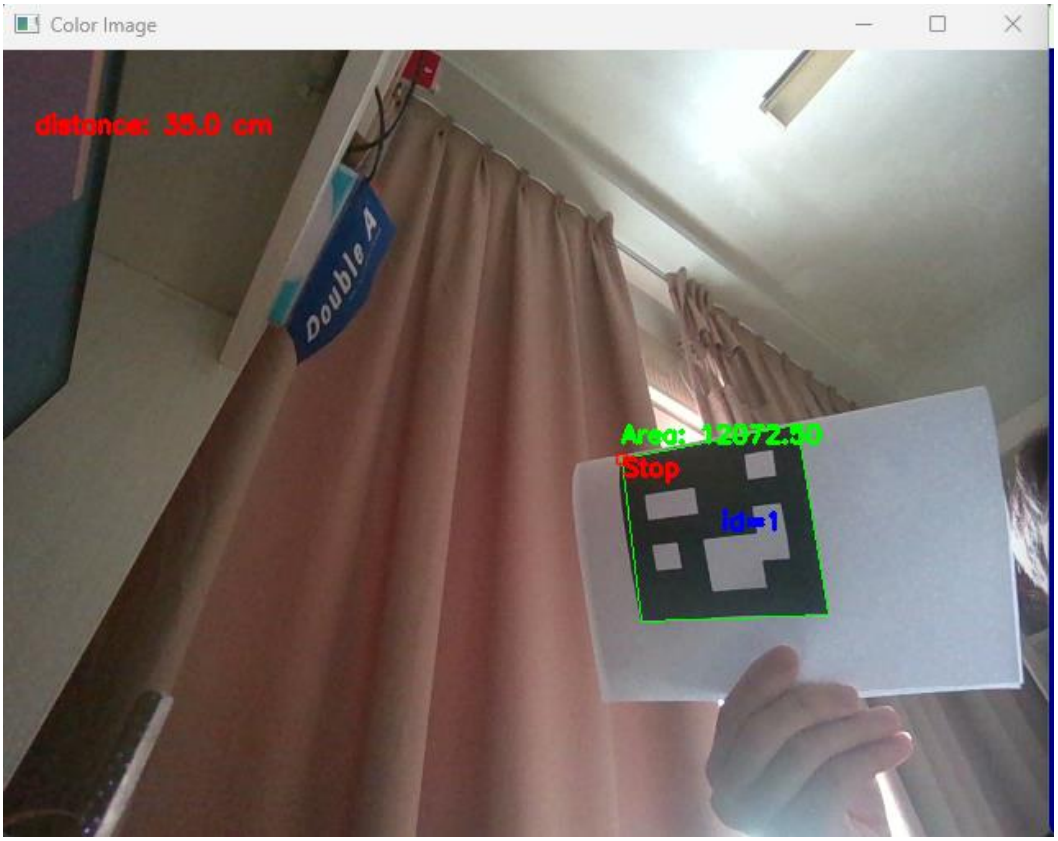

Figure 68. Area and Command through Marker Detection

The method of continuously polling the Arduino for distance data introduces a slight delay in camera update rate. This affects obstacle detection accuracy. An alternative of having the Arduino send a specific command upon detecting an obstacle within a threshold was also explored. However, this method also resulted in lagging camera frame.

Therefore, to address these limitations, the current design adopts an independent processing approach on the Arduino. The distance data from the ultrasonic sensor is processed in the Arduino. If the detected obstacle falls within a pre-defined threshold (below 5 cm), the Arduino triggers the robot to stop and wait until the obstacle is clear. This approach prioritizes immediate obstacle detection and reaction for safe navigation in cluttered environments.

4.3 ROS Implementation

ROS (Robot Operating System) framework provides a collection of tools and libraries to integrate and control hardware and software of robots. Ubuntu 20.04 Noetic is used for software development on my laptop.

On Raspberry Pi, Ubuntu Mate 20.04 is installed. SSH access was then activated on the Raspberry Pi, allowing for secure remote access through a dedicated SSH client on the PC. This established a command-line interface for interacting with the Raspberry Pi, granting the ability to operate the robot remotely.

4.3.1 ROS Serial

ROS serial package provides a communication bridge between ROS and Arduino boards. Raspberry Pi deals with high-level processing and control while the Arduino board collects the data from sensors in real time. The sensor data collected by the Arduino board is then transmitted to the ROS environment through ROS Serial, where it is converted into ROS messages and published to relevant topics for other ROS nodes to access and utilize.

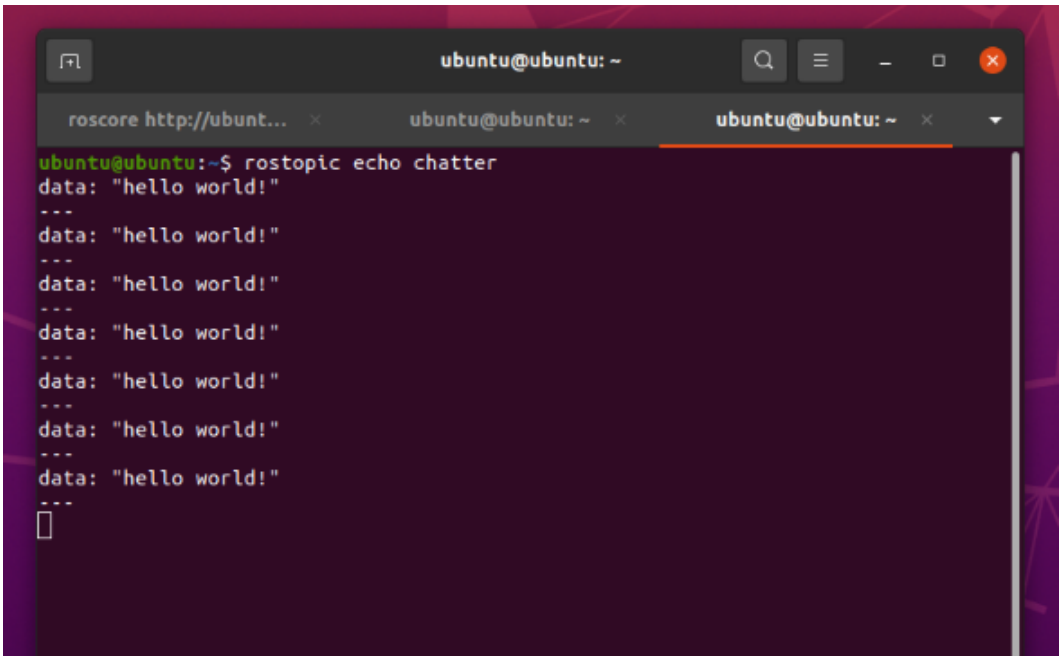

Figure 69. Hello World Testing in ROS

In this section, a simple program that prints “Hello World” is used to test ROS Serial functionality. This basic test verifies that data can be successfully exchanged between the ROS environment and the Arduino board. It allows easier integration of sensors and actuators controlled by Arduino.

5.3.2 Autonomous Navigation

LiDAR (Light Detection and Ranging) generates three-dimensional information to build a detailed 3D map of objects and the surrounding environment. The LiDAR sensor emits pulsed laser light to its surroundings and some of the lights are bounced back from an object to the sensor. Then, the sensor can calculate the distance of the object through measuring the time taken of a pulse return to the sensor. With the point cloud that represents the 3-D shape of objects and terrains, the 3D map of the environment can be created. The 3D map can be updated continuously as it moves forward. Based on the map, the robot can identify and avoid obstacles, and plan its path with various algorithms.

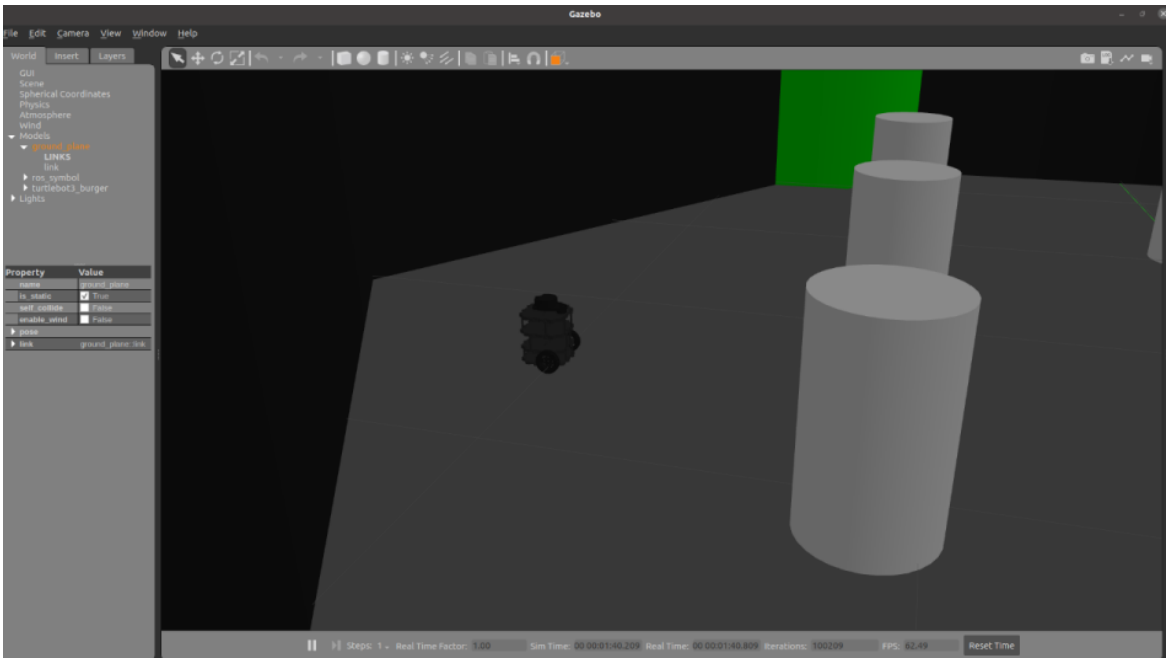

Figure 70. Simulation of TurtleBot3

TurtleBot3 is used to test the navigation and SLAM on Gazebo simulation. It involves building a map and interpreting data to make decisions and plan paths. A Lidar gives range data, and the depth camera gives visual data. SLAM packages in ROS use the data from these sensors to build a map of the environment and localize the robot within the map.

For path planning, the global and local planner are used. The global planner creates a path from the current position to the destination while local planner controls the velocity of the robot to avoid obstacles. The best path would be the shortest path while avoiding obstacles.

Although SLAM is not implemented in our robot, it could be a valuable addition for future iterations of the robot. Integrating SLAM would enable the robot to navigate more complex environments.

6. Result and Test

6.1 Climbing Ability Test

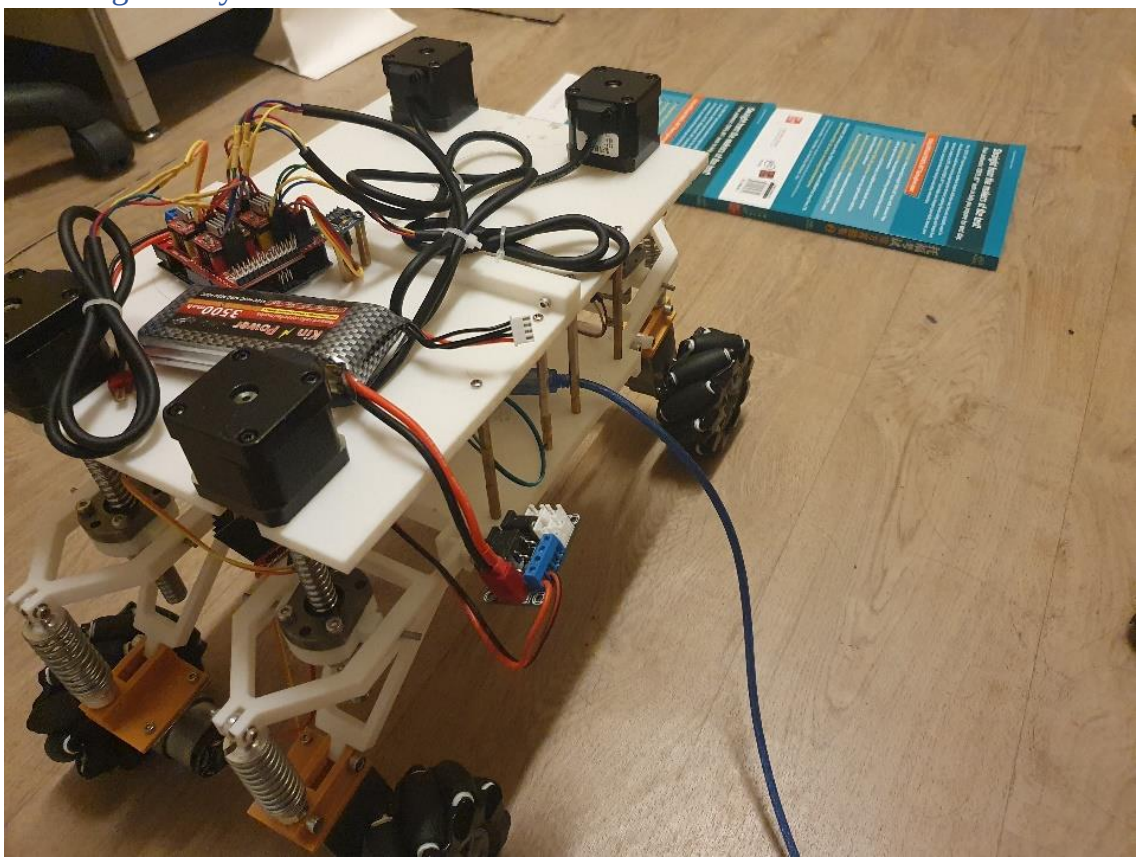

Figure 71. Climbing Test

The mobile robot's climbing ability was assessed using a series of books stacked on the ground to simulate obstacles of varying heights. The robot successfully climbed over obstacles up to 2cm tall. It demonstrates its ability to handle minor bumps and inclines. However, it was unable to overcome obstacles exceeding 2.5 centimeters in height.

6.2 Active Suspension Angles Achieved

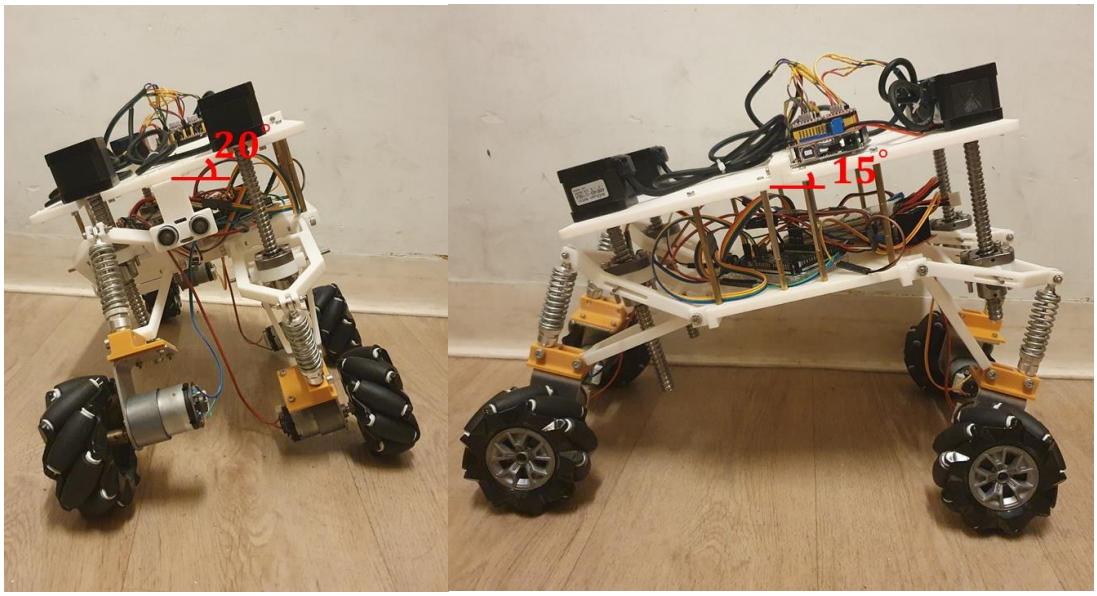

Figure 72. Angles Achieved

- X-axis (Roll): ± 15 degrees
- Y-axis (Pitch: ± 20 degrees

The active suspension system is capable of achieving a maximum angle of 15 degrees in the X direction and 20 degrees in the Y direction. This range of motion allows the mobile robot to adapt to various terrains and obstacles. By accommodating such angles, the active suspension system enhances the robot's ability to maintain balance and maneuver effectively. The system can adjust the height and orientation to optimize stability and traction.

6.3 Active suspension test

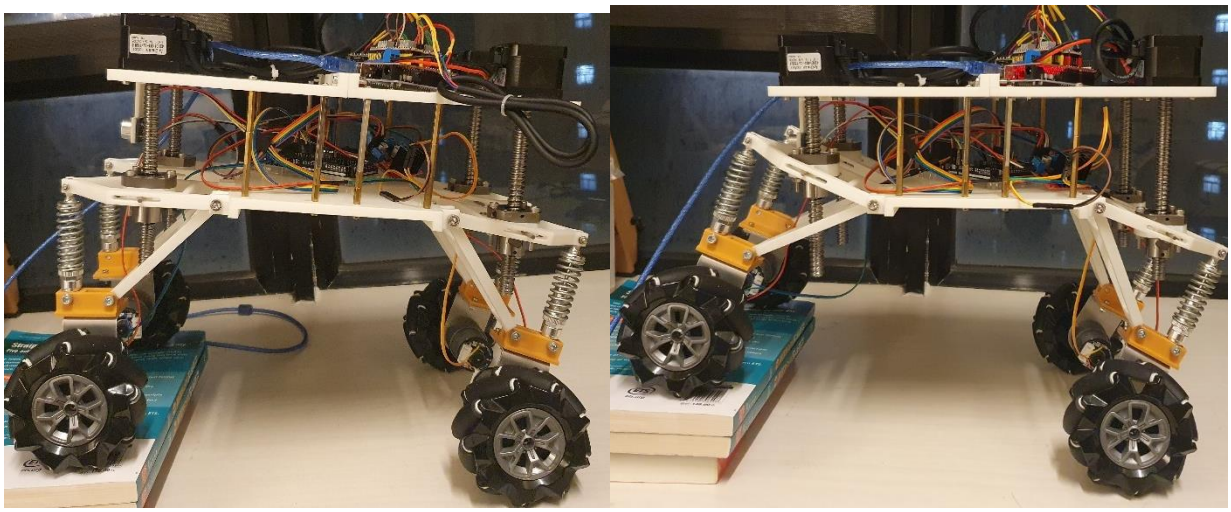

Figure 73. Adjusting X Angle Before and After Comparison

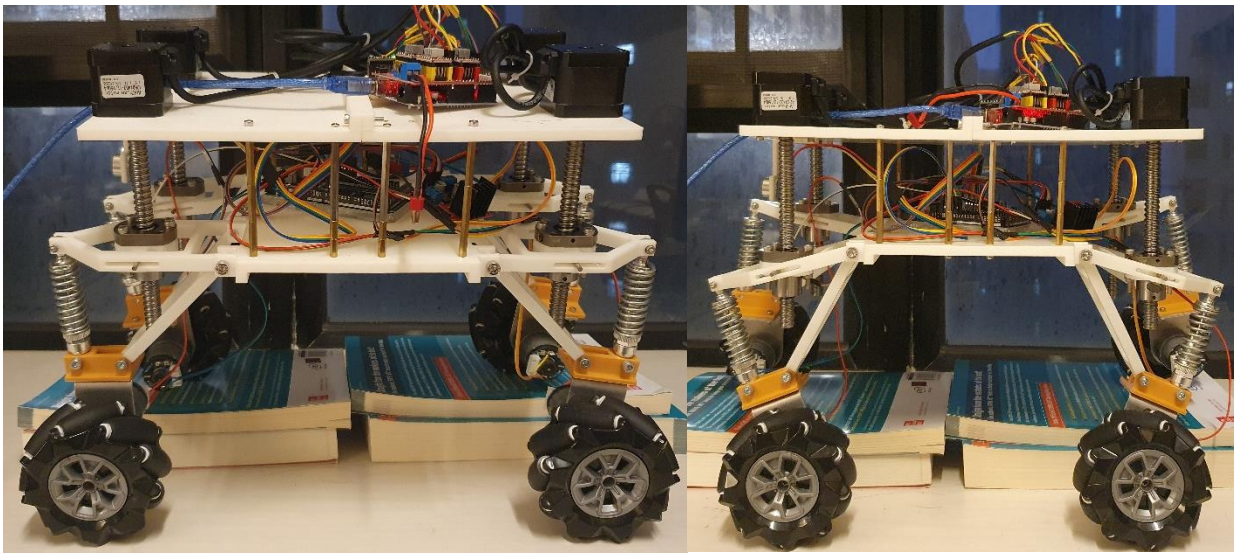

Figure 74. Adjusting Y Angle Before and After Comparison

The active suspension system adjusts the robot's orientation. This precise control allows the robot to maintain a level posture during operation.

6.3.1 Active Suspension Angle Control

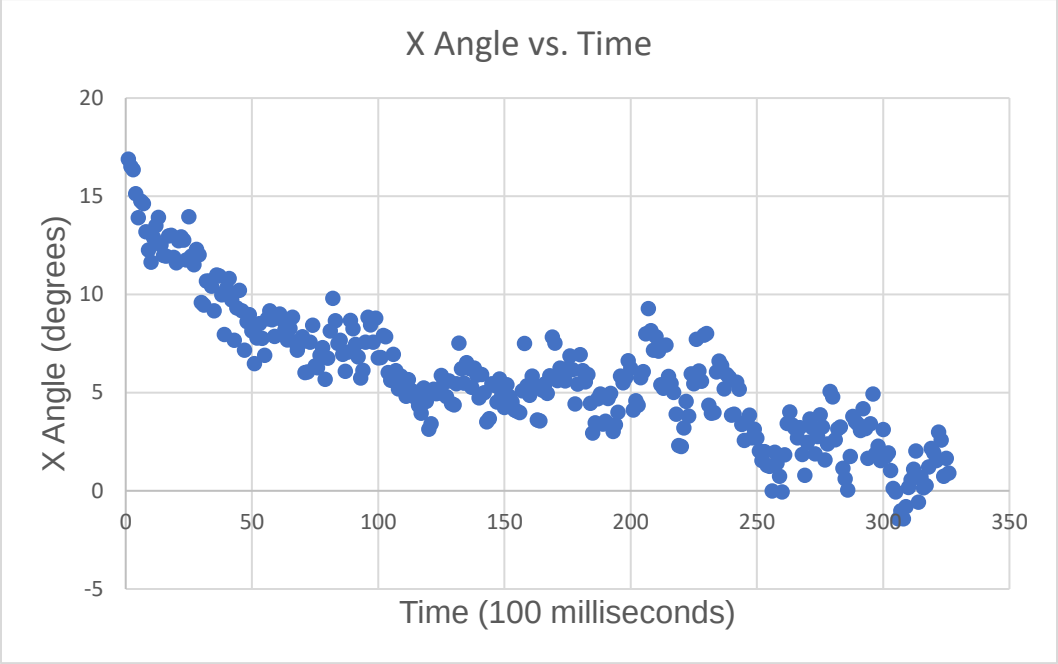

Figure 75. Stepper Motor X-Axis Angle Control (Slow Speed)

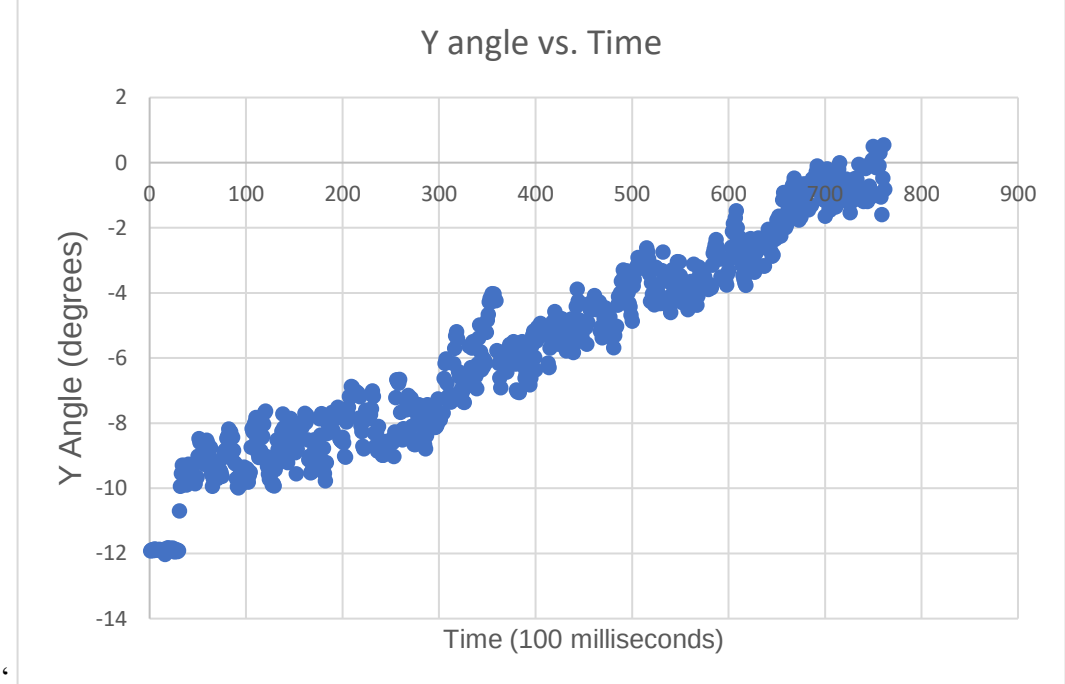

Figure 76. Stepper Motor Y-Axis Angle Control (Slow Speed)

The robot was positioned at a specific angle to evaluate its ability to adjust the upper plate. Based on the angle data, the stepper motors demonstrate their ability to precisely adjust the platform at slow speeds. The angle data shows that the stepper motors are able to adjust and go make the plate parellel.

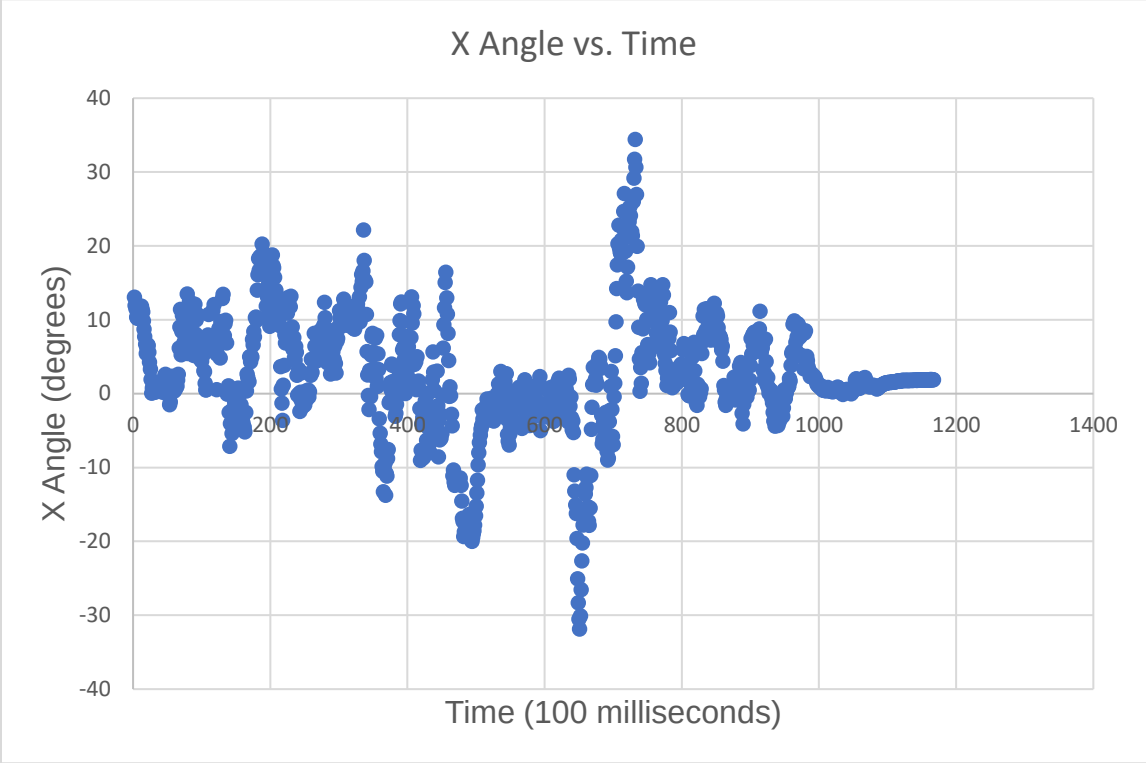

Figure 77. Stepper Motor X-Axis Angle Control (Fast Speed)

When the speed is increased, the vibration from the stepper motor introduces noise into the sensor leadings and leads to variations. This vibration introduces high-frequency noise into the sensor readings. It causes them to fluctuate significantly.

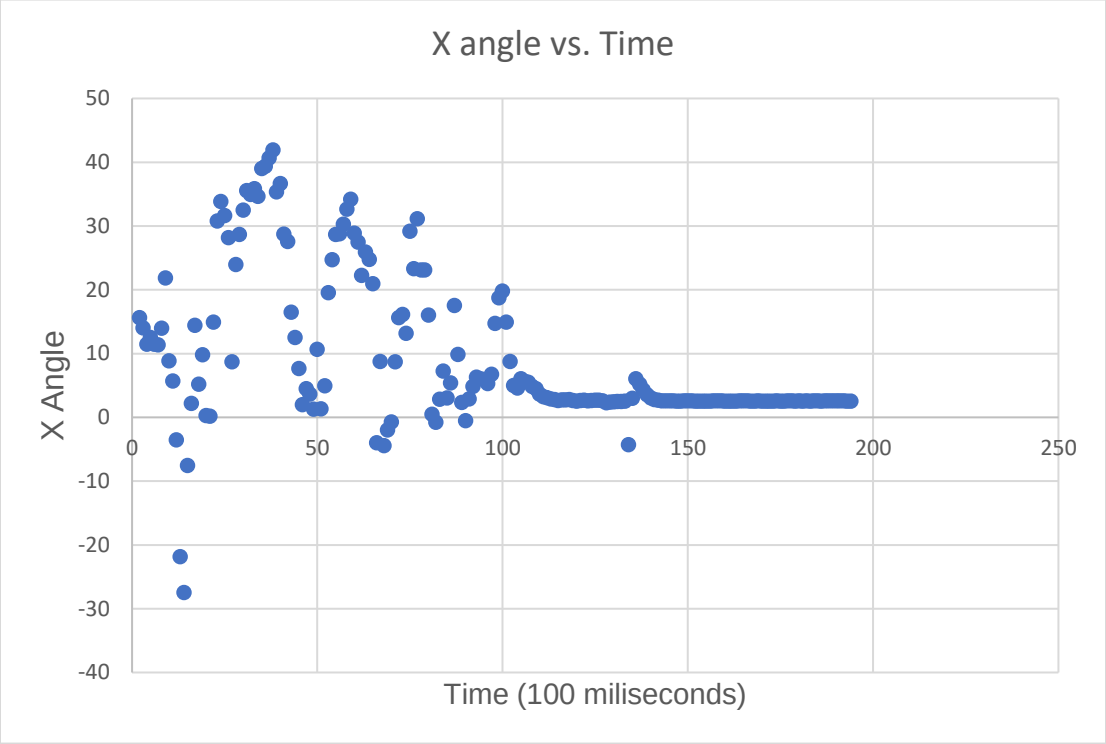

Figure 78. Low Pass Filtering with Averaging

A low pass filtering with averaging is introduced to reduce high-frequency noise in the angle measurements. The equation used for filtering is:

angleX = alpha * angleX + (1.0 - alpha) * prevAngleX

The alpha variable, ranging from 0 to 1, controls the filtering strength. A lower alpha value gives less weight to the new angle reading and more weight to the previous filtered angle value. This results in stronger filtering and gives smoother readings. However, it could be slower to react to changes. Different alpha values are tested to find the optimal balance between filtering strength and responsiveness.

6.3.2 Active Suspension Performance on Slopes and Obstacles

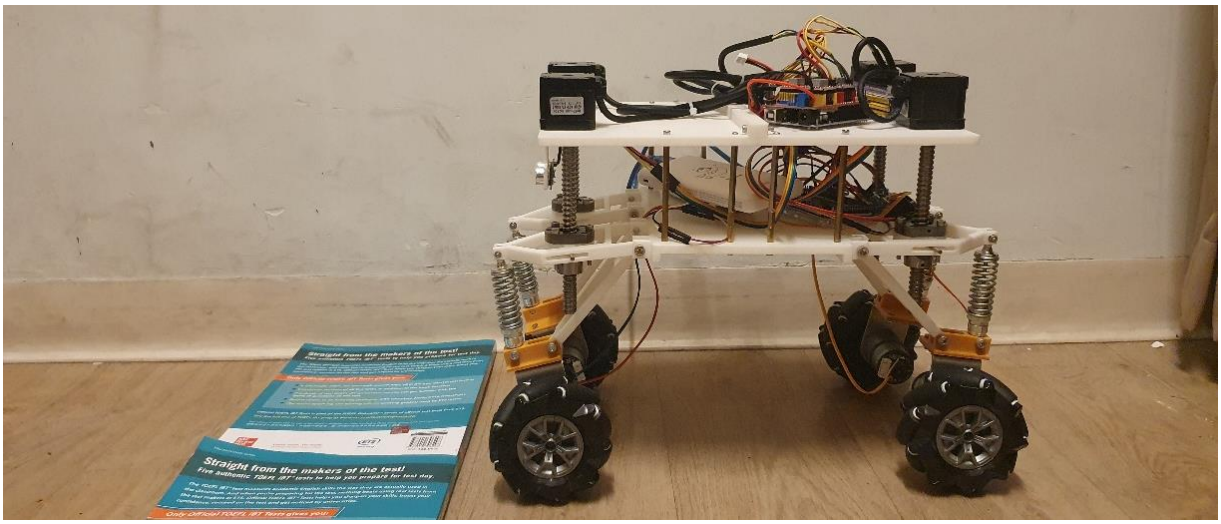

Figure 79. Climbing Obstacle Test

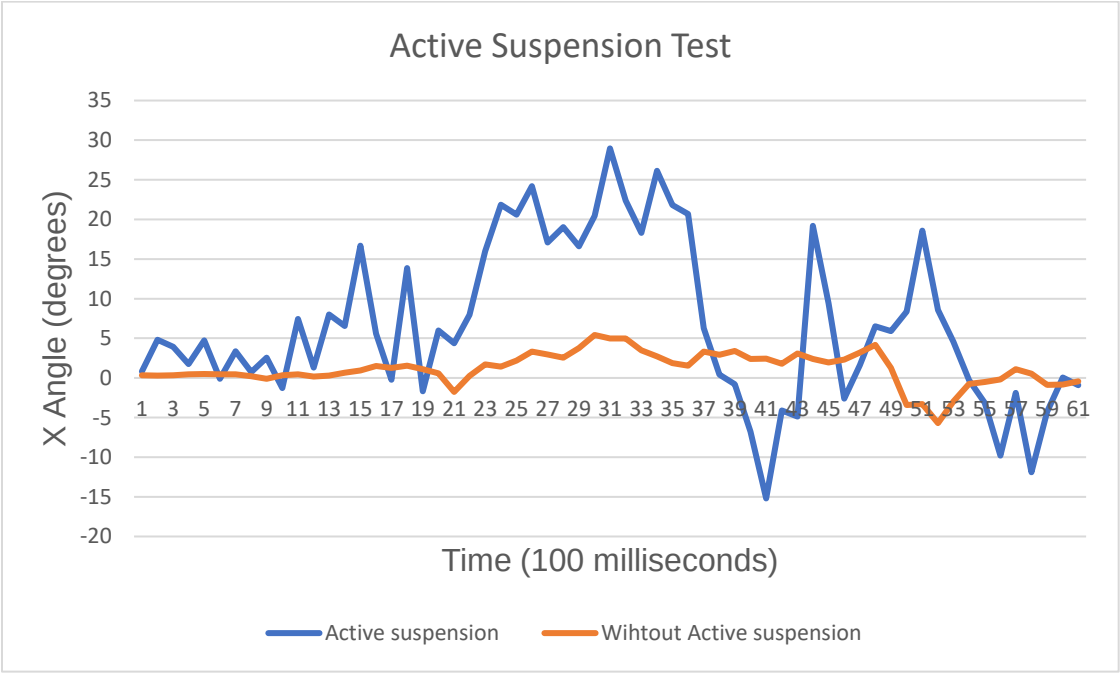

Figure 80. Performance on Obstacle Obstacle (Fast Speed)

Despite a moderate slope change of approximately 5 degrees, high-frequency noise in the sensor readings affected the measured angle.

Figure 81. Climbing Slope Test

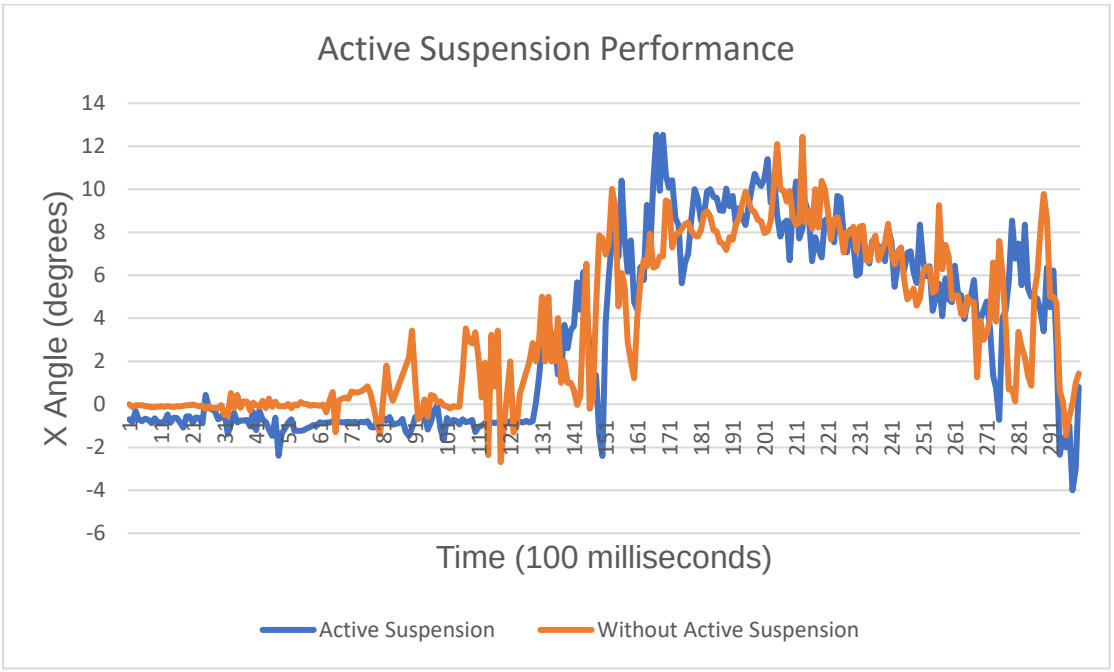

Figure 82. Active Suspension Performance on Slope (Slow Speed)

The angle data indicates a slight improvement in angle control with the active suspension system engaged. While the angle difference may appear small, this is due to the slow speed chosen for the test. This slower speed minimizes high-frequency noise from the stepper motors, allowing for more accurate angle measurements.

6.3.3 Summary on active suspension test

The experimental results demonstrate the ability of the active suspension system to precisely adjust the robot's orientation based on angle data. However, a trade-off exists between speed and measurement accuracy: At higher motor speeds, vibrations introduce noise into the angle measurements, impacting the effectiveness of the active suspension system. Conversely, operating at very slow speeds may hinder the robot’s performance, affecting stability. These findings highlight the importance of balancing speed with sensor accuracy for optimal performance.

6.4 Navigation Test

Figure 83. Detection of Markers

The robot navigation test employed a pre-defined set of markers placed within the environment. The markers were identified and the marker's area within the image analyzed. The robot continuously searched for the marker in its field of view. Upon detecting a maker with the desired ID, the area of the marker was calculated. If the area was small, it means that the robot is far away and has to move forward until the marker area reaches a specific threshold. After it is reached, the robot rotates until it searches for the next marker.

Figure 84. Corrective Turns

The robot would perform corrective turns (left or right) to align itself with the marker's center. This ensures that it is heading directly towards the next marker. It repeated the detection, approach, and action cycle until it reached its destination.

6.4.1 Summary of Navigation Test

This marker-based navigation approach demonstrated its potential as a reliable and efficient method for robots to navigate known environments. During the test, the robot successfully reached all pre-located markers. Under bright sunlight conditions, the marker detection was well performed.

The marker-based navigation test serves as a valuable foundation for developing more robust and adaptable robot navigation systems. The system can be improved by integrating additional sensors like LiDAR or ultrasonic sensors that could enhance obstacle detection and path refinement in more complex environments. Path planning algorithms might optimize robot movement and reduce navigation time.

7. Future Development

7.1 Implementation of advanced filtering algorithms

Implementing advanced filtering algorithms could potentially mitigate the effects of high-speed vibrations on sensor readings. While traditional filtering methods are effective in removing low-frequency noise, they are ineffective in removing high-frequency vibrations. Advanced algorithms, such as Kalman filters or Wavelet denoising techniques can significantly reduce these high frequencies.

7.2 Advanced Slope Compensation Techniques

LiDAR or stereo cameras could be integrated for more comprehensive terrain analysis, enabling real-time adjustments to the robot's tilt based on the surrounding slope variations. This can develop predictive algorithms that can predict upcoming slope changes based on sensor data. This would allow the robot to proactively adjust its suspension before encountering a significant incline, improving overall efficiency and stability.

A path planning algorithm that considers slope information during route selection can be developed. This would allow the robot to prioritize paths with less steep inclines.

7.3 Advanced Perception System

LiDAR could be integrated into the perception system to build a 3D map of the environment. This 3D map can be valuable obstacle detection and navigation in complex environments, especially when visibility is limited. Stereo camera systems can provide accurate depth information. Implementing AI algorithms for object recognition would further expand the robot’s capabilities, allowing it to identify and interact with specific objects in its environment.

8. Project Plan and Development Process

8.1 Project Schedule

Table 5. Gantt Chart

Months		Sep				Oct				Nov				Dec				Jan				Feb				Mar				April			
Weeks		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Initial stage	Research																																
	Conceptualization																																
	Purchase																																
Mechanical Design	Suspension System																																
	Chassis Design																																
	Component placement																																
	Fabrication																																
Electronic Design	Power System																																
	Sensor integration																																
	Wiring																																
Software Development	Sensor Data Processing																																
	Path Planning Algorithm																																
	Detection Algorithm																																
	Control System																																
Test	Hardware Test																																
	Software Test																																
	Refine & Improve																																

We have been working on hardware development and software development in parallel. After the fabrication of the robot, electronic and software components were integrated onto the robot. Then, the robot was tested in a controlled environment and real-world, uneven terrain. Based on the results, necessary improvements were made for better performance.

The proposed design of the robot was fabricated through 3D printing. The mechanism of the suspension system has gone through rigorous testing for validation with 3D printed components before finalization of the design.

Electronic components including sensors, motors, and control boards were integrated onto the mobile platform. Various tests were performed to ensure basic functionalities including adjusting its suspension. The software was tested to accurately control the hardware from sensor data. Then, it was tested outdoors in more dynamic environments to test its ability to adapt to different terrains and obstacles. The performance of the active suspension system, sensors, and control algorithms

in real-world conditions was tested. Through the testing process, issues and limitations were identified for further improvement.

8.2 Resource Planning and Availability

Table 6 Resource Table

Major Component	**Quantity**	**Unit Cost (HKD)**	**Total Cost (HKD)**	**Availability**
DC Motor (JGB37-520)	4	0	0	Borrow from ME Lab GH033 (Professor)
Lead Screw Stepper Motor	4	160	640	Purchase on Taobao
Omni Wheel	4	0	0	Borrow from ME Lab GH033 (Professor)
Arduino Mega	1	280	280	Purchase on Taobao
Arduino Uno	1	0	0	Borrow from ME Lab GH033 (Professor)
Raspberry Pi 4	1	600	600	Purchase on Taobao
RGBD Camera	1	0	0	Borrow from ME Lab GH033 (Professor)
IMU	1	0	0	Borrow from ME Lab GH033 (Professor)
Motor Shield	6	2	12	Purchase on Taobao
Ultrasonic Sensor	1	3	3	Purchase on Taobao

Stepper motor(42-step)	4	151.62	606.48	Purchase on Taobao
Hexagonal Copper Column	1	74.73	74.73	Purchase on Taobao
Nylon Rolled Belt	1	7.04	7.04	Purchase on Taobao
Hexagon socket head cap screws and nuts set	1	31.41	31.41	Purchase on Taobao
Iron bar-14 pair	1	219	219	Purchase on Taobao
M3-M5 screws and nuts set	1	27.54	27.54	Purchase on Taobao
The 12v battery	2	72.78	145.56	Purchase on Taobao
The 3d printed Robot frame	1	0	0	Printed from IC 3D print center and ME Lab GH033
Replacement cost of spare parts	2	160	320	Purchase on Taobao
Total cost			2966.76	

Conclusion

In this project, we have designed, prototyped, and tested a mobile robot with an active suspension system module of a mobile robot. The module includes the robot frame, wheels, motors, and connecting parts. The DC motors and stepper motors have been chosen carefully to meet the torque requirements of our design. Mecanum wheels were selected for the ability to provide smooth turning. The simulations on SolidWorks were instrumental in evaluating the structural integrity of components under extreme loading conditions.

The system utilizes a ball screw stepper motor design to effectively adjust the robot's height based on terrain variations. The IMU sensor data is processed to determine the robot's orientation and maintain balance on uneven surfaces. A balancing method was implemented that controls the four stepper motors in response to changes in terrain height and slope. Furthermore, the robot incorporates autonomous navigation using ArUco marker detection, enabling it to follow a pre-defined path.

This project represents a comprehensive approach to designing a mobile robot with an active suspension system capable of navigation autonomously in known environments. This project demonstrates the successful integration of an active ball screw suspension system with real-time IMU-based balancing and ArUco marker navigation, opening doors for future advancements in mobile robot design.

Reference

[1] T. R. Rao and P. Anusha, "Active suspension system of a 3 DOF quarter car using fuzzy logic control for ride comfort," *2013 International Conference on Control, Automation, Robotics and Embedded Systems (CARE)*, pp. 1-6, 2013. doi: 10.1109/CARE.2013.6733771.

[2] R. R. Murphy et al., "Search and rescue robotics," Springer Handbook of Robotics, pp. 1151–1173, 2008. doi:10.1007/978-3-540-30301-5_51

[3] H. Soltani, "Study a vehicle rollover maneuvers by using LQR active suspension control system," International Journal of Electrical and Electronics Research, vol. 4, no. 3, pp. 23–41, Jul. 2019.

[4] E. Esmailzadeh and H. Taghirad, Active Vehicle Suspensions with Optimal State-Feedback Control, Int. Journal of Modeling and Simulation, Vol.18, No. 3, 1998.

[5] Mohammad Biglarbegiana, William Meleka & Farid Golnaraghi, Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility Volume 46, Issue 8, pp.691-711, 2008

[6] T. Sabu1, "OVERVIEW OF ACTIVE SUSPENSION SYSTEM IN AUTOMOBILES," International Research Journal of Engineering and Technology (IRJET), vol. 08, no. 07, Jul. 2021.

[7] D. Karnopp, "Theoretical Limitations in Active Vehicle Suspensions", Vehicle System Dynamics, vol. 15(1986), no.1, pp. 41-54, 2007. doi: 10.1080/00423118608968839(1986).

[8] S. Kumar, A. Medhavi, and R. Kumar, "Active and passive suspension system performance under Random road profile excitations," The International Journal of Acoustics and Vibration, vol. 25, no. 4, pp. 532–541, Dec. 2020. doi:10.20855/ijav.2020.25.41702

[9] S. Jiang, Z. Li, S. Lin, W. Shi, Z. Zheng, H. Che, S. Yin, C. Zhang and Z. Jia, "Design, Control and Experiments of An Agile Omnidirectional Mobile Robot with Active Suspension,"2022 IEEE 18th International Conference on Automation Science and Engineering (CASE), pp. 913-918, 2022. doi:10.1109/CASE49997.2022.9926462.

[10] H. Jiang, G. Xu, W. Zeng, F. Gao, and K. Chong, "Lateral Stability of a Mobile Robot Utilizing an Active Adjustable Suspension" *Applied Sciences,* vol. 9, no. 20: 4410, 2019. Available: https://doi.org/10.3390/app9204410.

[11] "Principles of Robot Motion: Theory, Algorithms, and Implementations [Book Review]," *IEEE Robotics & Automation Magazine,* vol. 12, no. 3, pp. 110–110, Sep. 2005, doi: https://doi.org/10.1109/mra.2005.1511878.

[12] M. Elbanhawi and M. Simic, "Sampling-Based Robot Motion Planning: A Review," IEEE Access, vol. 2, pp. 56–77, 2014, doi: https://doi.org/10.1109/access.2014.2302442.

[13] A. Loganathan and N. S. Ahmad, "A systematic review on recent advances in autonomous mobile robot navigation," Engineering Science and Technology, an International Journal, vol. 40, p. 101343, Apr. 2023, doi: https://doi.org/10.1016/j.jestch.2023.101343.

[14] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik,* vol. 1, no. 1, pp. 269–271, Dec. 1959, doi: https://doi.org/10.1007/bf01386390.

[15] L. C. Wang, L. S. Yong, and M. H. Ang, "Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment," *Proceedings of the IEEE Internatinal Symposium on Intelligent Control,* doi: https://doi.org/10.1109/isic.2002.1157868.

[16] J. Zheng, S. Bi, B. Cao, and D. Yang, "Visual localization of inspection robot using extended Kalman filter and ARUCO markers," 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dec. 2018. doi:10.1109/robio.2018.8664777

[17] A. Babinec, L. Jurišica, P. Hubinský, and F. Duchoň, “Visual localization of mobile robot using artificial markers,” Procedia Engineering, vol. 96, pp. 1–9, 2014. doi:10.1016/j.proeng.2014.12.091

[18] R. Zeng, Y. Kang, J. Yang, W. Zhang, and W. Qi, “Terrain Parameters Identification of Kinematic and Dynamic Models for a Tracked Mobile Robot,” IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, May 2018, doi: https://doi.org/10.1109/imcec.2018.8469223.

[19] R. Ventura and P.U. Lima, "Search and Rescue Robots: The Civil Protection Teams of the Future", Emerging Security Technologies (EST), vol. 12, no. 19, pp. 5-7, Sept. 2012.

[20] “Comparative Analysis of Mobile Robot Wheels Design,” 2018 11th International Conference on Developments in eSystems Engineering (DeSE), 2018. DOI: 10.1109/DeSE.2018.00041

[21] A. Kholil, E. Asyaefudin, N. Pinto, and S. Syaripuddin, “Compression strength characteristics of ABS and PLA materials affected by layer thickness on FDM,” Journal of Physics: Conference Series, vol. 2377, no. 1, p. 012008, Nov. 2022. doi:10.1088/1742-6596/2377/1/012008

Appendices

Appendix A: Arduino Code for Stepper Motor Control

```
#include "Wire.h"
#include <MPU6050_light.h>
#include <AccelStepper.h>

#define MOTOR_X_ENABLE_PIN 8
#define MOTOR_X_STEP_PIN 2
#define MOTOR_X_DIR_PIN 5

#define MOTOR_Y_ENABLE_PIN 8
#define MOTOR_Y_STEP_PIN 3
#define MOTOR_Y_DIR_PIN 6

#define MOTOR_Z_ENABLE_PIN 8
#define MOTOR_Z_STEP_PIN 4
#define MOTOR_Z_DIR_PIN 7

#define MOTOR_A_ENABLE_PIN 8
#define MOTOR_A_STEP_PIN 12
#define MOTOR_A_DIR_PIN 13

MPU6050 mpu(Wire);
unsigned long timer = 0;

const float TILT_THRESHOLD = 2.5;

float prevAngleX = 0.0;
float prevAngleY = 0.0;
float alpha = 0.1; // filtering strength


long startPosition[4];
long endPosition[4];

AccelStepper motorX(1, MOTOR_X_STEP_PIN, MOTOR_X_DIR_PIN);
AccelStepper motorY(1, MOTOR_Y_STEP_PIN, MOTOR_Y_DIR_PIN);
AccelStepper motorZ(1, MOTOR_Z_STEP_PIN, MOTOR_Z_DIR_PIN);
AccelStepper motorA(1, MOTOR_A_STEP_PIN, MOTOR_A_DIR_PIN);

void setup() {
  Serial.begin(9600);
```

```
Wire.begin();

byte status = mpu.begin();
Serial.print(F("MPU6050 status: "));
Serial.println(status);
while(status!=0){ } // stop everything if could not connect to MPU6050

/*Serial.println(F("Calculating offsets, do not move MPU6050"));
delay(1000);
mpu.calcOffsets(); // offset gyro and accelero
Serial.println("Done!\n");*/

  // Initialize stepper motors
pinMode(MOTOR_X_ENABLE_PIN, OUTPUT);
pinMode(MOTOR_Y_ENABLE_PIN, OUTPUT);
pinMode(MOTOR_Z_ENABLE_PIN, OUTPUT);
pinMode(MOTOR_A_ENABLE_PIN, OUTPUT);

motorX.setEnablePin(MOTOR_X_ENABLE_PIN);
motorX.setPinsInverted(false, false, true);
motorX.setAcceleration(100);
motorX.setMaxSpeed(300);
motorX.setSpeed(200);
motorX.enableOutputs();

motorY.setEnablePin(MOTOR_Y_ENABLE_PIN);
motorY.setPinsInverted(false, false, true);
motorY.setAcceleration(100);
motorY.setMaxSpeed(300);
motorY.setSpeed(200);
motorY.enableOutputs();

motorZ.setEnablePin(MOTOR_Z_ENABLE_PIN);
motorZ.setPinsInverted(false, false, true);
motorZ.setAcceleration(100);
motorZ.setMaxSpeed(300);
motorZ.setSpeed(200);
motorZ.enableOutputs();

motorA.setEnablePin(MOTOR_A_ENABLE_PIN);
motorA.setPinsInverted(false, false, true);
motorA.setAcceleration(100);
motorA.setMaxSpeed(300);
motorA.setSpeed(200);
```

```
  motorA.enableOutputs();

  // Set the range with respect to the current position
  startPosition[0] = motorX.currentPosition() + 1000; //500 = 1.5cm
  startPosition[1] = motorY.currentPosition() + 1000;
  startPosition[2] = motorZ.currentPosition() + 1000;
  startPosition[3] = motorA.currentPosition() + 1000;
  endPosition[0] = motorX.currentPosition() - 500;
  endPosition[1] = motorY.currentPosition() - 500;
  endPosition[2] = motorZ.currentPosition() - 500;
  endPosition[3] = motorA.currentPosition() - 500;
}

void loop() {
  mpu.update();
  float angleX = mpu.getAngleX();
  float angleY = mpu.getAngleY();

  // Apply low-pass filtering with averaging
  angleX = alpha * angleX + (1.0 - alpha) * prevAngleX;
  angleY = alpha * angleY + (1.0 - alpha) * prevAngleY;

  prevAngleX = angleX;
  prevAngleY = angleY;

  if((millis()-timer)>100){ // print AngleX and AngleY every 100ms
    Serial.print("\tX : ");
    Serial.print(mpu.getAngleX());
    Serial.print("\tY : ");
    Serial.print(',');
    Serial.println(mpu.getAngleY());
    timer = millis(); }

  if(abs(angleX)>TILT_THRESHOLD || abs(angleY)>TILT_THRESHOLD){

    if (angleX > TILT_THRESHOLD && angleY > TILT_THRESHOLD) { // positive X
angle, positive Y angle
      motorX.moveTo(startPosition[0]);
      motorX.run();
      }
    else if (angleX > TILT_THRESHOLD && angleY < -TILT_THRESHOLD) { // positive X
angle, negative Y angle
      motorY.moveTo(startPosition[1]);
      motorY.run();
```

```
    }
    else if (angleX < -TILT_THRESHOLD && angleX > TILT_THRESHOLD) { // negative X
angle, positive Y angle
      motorZ.moveTo(startPosition[2]);
      motorZ.run();
    }
    else if (angleX < -TILT_THRESHOLD && angleY < -TILT_THRESHOLD) { // negative
X angle, negative Y angle
      motorA.moveTo(startPosition[3]);
      motorA.run();
    }
    else if (angleX > TILT_THRESHOLD) { // positive X angle
      motorX.moveTo(startPosition[0]);
      motorX.run();
      motorY.moveTo(startPosition[1]);
      motorY.run();
      motorZ.moveTo(endPosition[2]);
      motorZ.run();
      motorA.moveTo(endPosition[3]);
      motorA.run();
    }
    else if (angleX < -TILT_THRESHOLD) { // negative X angle
      motorX.moveTo(endPosition[0]);
      motorX.run();
      motorY.moveTo(endPosition[1]);
      motorY.run();
      motorZ.moveTo(startPosition[2]);
      motorZ.run();
      motorA.moveTo(startPosition[3]);
      motorA.run();
    }
    else if (angleY > TILT_THRESHOLD) { // positive Y angle
      motorX.moveTo(startPosition[0]);
      motorX.run();
      motorY.moveTo(endPosition[1]);
      motorY.run();
      motorZ.moveTo(startPosition[2]);
      motorZ.run();
      motorA.moveTo(endPosition[3]);
      motorA.run();
      }
    else if (angleY < -TILT_THRESHOLD) { // negative Y angle
      motorX.moveTo(endPosition[0]);
      motorX.run();
```

```
      motorY.moveTo(startPosition[1]);
      motorY.run();
      motorZ.moveTo(endPosition[2]);
      motorZ.run();
      motorA.moveTo(startPosition[3]);
      motorA.run();
    }
  }
  else{
    motorX.stop();
    motorY.stop();
    motorZ.stop();
    motorA.stop();
    }
}
```

Appendix B: DC Motor Control, Ultrasonic Ranging, and Python Serial Communication

```
#define TRIG_PIN A1
#define ECHO_PIN A2

#define IN1_PIN 9
#define IN2_PIN 8
#define ENA_PIN 10
#define IN3_PIN 7
#define IN4_PIN 6
#define ENB_PIN 5

unsigned long timer = 0;

void setup() {
    // Initialize serial communication
    Serial.begin(9600);

    // Set trigger pin as output, echo pin as input
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);

    // Set motor control pins as outputs
    pinMode(IN1_PIN, OUTPUT);
    pinMode(IN2_PIN, OUTPUT);
    pinMode(ENA_PIN, OUTPUT);
    pinMode(IN3_PIN, OUTPUT);
    pinMode(IN4_PIN, OUTPUT);
    pinMode(ENB_PIN, OUTPUT);
}

// Function to move both motors forward at a set speed
void moveForward(int speed) {
    digitalWrite(IN1_PIN, HIGH);
    digitalWrite(IN2_PIN, LOW);
    analogWrite(ENA_PIN, speed); // Set speed

    digitalWrite(IN3_PIN, HIGH);
    digitalWrite(IN4_PIN, LOW);
    analogWrite(ENB_PIN, speed); // Set speed
}

// Function to move both motors backward at a set speed
void moveBackward(int speed) {
```

```
    digitalWrite(IN1_PIN, LOW);
    digitalWrite(IN2_PIN, HIGH);
    analogWrite(ENA_PIN, speed); // Set speed

    digitalWrite(IN3_PIN, LOW);
    digitalWrite(IN4_PIN, HIGH);
    analogWrite(ENB_PIN, speed); // Set speed
}

// Function to rotate left (both motors turn in opposite directions)
void rotateAnticlockwise(int speed) {
    digitalWrite(IN1_PIN, HIGH);
    digitalWrite(IN2_PIN, LOW);
    analogWrite(ENA_PIN, speed); // Set speed

    digitalWrite(IN3_PIN, LOW);
    digitalWrite(IN4_PIN, HIGH);
    analogWrite(ENB_PIN, speed); // Set speed
}

// Function to rotate right (both motors turn in opposite directions)
void rotateClockwise(int speed) {
    digitalWrite(IN1_PIN, LOW);
    digitalWrite(IN2_PIN, HIGH);
    analogWrite(ENA_PIN, speed); // Set speed

    digitalWrite(IN3_PIN, HIGH);
    digitalWrite(IN4_PIN, LOW);
    analogWrite(ENB_PIN, speed); // Set speed
}
// Function to move left (right wheels move slower than left wheels)
void moveLeft(int speed) {
  digitalWrite(IN1_PIN, HIGH);
  digitalWrite(IN2_PIN, LOW);
  analogWrite(ENA_PIN, speed); // Set speed

  digitalWrite(IN3_PIN, HIGH);
  digitalWrite(IN4_PIN, LOW);
  analogWrite(ENB_PIN, speed*0.8); // Set speed
}

// Function to move right (left wheels move slower than right wheels)
void moveRight(int speed) {
  digitalWrite(IN1_PIN, HIGH);
```

```
  digitalWrite(IN2_PIN, LOW);
  analogWrite(ENA_PIN, speed*0.8); // Set speed

  digitalWrite(IN3_PIN, HIGH);
  digitalWrite(IN4_PIN, LOW);
  analogWrite(ENB_PIN, speed); // Set speed
}
void loop() {

  // Ensure trigger pin is low
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);

  // Generate the ultrasound wave
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Measure the time it takes for the wave to return
  long duration = pulseIn(ECHO_PIN, HIGH);

  // Calculate the distance in centimeters
  long distance = duration / 29 / 2;

  if (distance < 10)
    moveForward(0);
    Serial.println("stop");

  if (Serial.available() > 0) {
      char command = Serial.read();
      switch (command) {
          case 'F':
              moveForward(80); // Adjust speed (0-255)
              break;
          case 'B':
              moveBackward(80); // Adjust speed (0-255)
              break;
          case 'L':
              moveLeft(80); // Adjust speed (0-255)
              break;
          case 'R':
              moveRight(80); // Adjust speed (0-255)
              break;
          case 'A':
```

```
                rotateAnticlockwise(80); // Adjust speed (0-255)
                break;
            case 'C':
                rotateClockwise(80); // Adjust speed (0-255)
                break;
            case 'S':
                // Stop motors
                moveForward(0);
                break;
        }
    }
}
```

Appendix C: Python Code for Marker-Based Robot Navigation

```
import cv2
import numpy as np
import pyrealsense2 as rs
import serial
import time

current_marker_id = 0  # Initialize the current marker to search for
start_rotation_time = time.time()  # Initialize rotation timer
threshold = 60
direction = 1  # Direction of movement: 1 for forward, -1 for backward

# Serial communication settings
SERIAL_PORT = 'COM8'  # Serial port
BAUD_RATE = 9600
ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)

# Initialize the RealSense pipeline
pipe = rs.pipeline()
cfg = rs.config()
cfg.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
cfg.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
pipe.start(cfg)

# Initialize the ArUco dictionary
aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_5X5_250)

# Function to send movement commands to Arduino
def send_command(command):
    ser.write(command.encode())

# Function to read distance from Arduino
def read_distance():
    try:
        distance_str = ser.readline().decode().strip()
        distance = float(distance_str)
        time.sleep(0.1)
        return distance
    except ValueError:
        return None

# Function to draw a center line on the image
def draw_center_line(image):
```

```
    height, width, _ = image.shape
    center_x = width // 2
    cv2.line(image, (center_x, 0), (center_x, height), (255, 255, 255), 2)

while True:
    # Wait for a new frame
    frames = pipe.wait_for_frames()
    color_frame = frames.get_color_frame()
    depth_frame = frames.get_depth_frame()

    if not color_frame or not depth_frame:
        continue

    # Convert RealSense frames to numpy arrays
    color_image = np.asanyarray(color_frame.get_data())
    depth_image = np.asanyarray(depth_frame.get_data())

    # Convert depth image to color map
    depth_colormap = cv2.applyColorMap(cv2.convertScaleAbs(depth_image,
alpha=0.5), cv2.COLORMAP_JET)

    # Detect markers in the color image
    corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(color_image,
aruco_dict)

    # Draw center line on a copy of the color image to avoid affecting marker
detection
    image_with_center_line = color_image.copy()
    draw_center_line(image_with_center_line)

    # Draw detected markers on the color image and compute area
    if ids is not None and len(ids) > 0:
        start_rotation_time = time.time()  # Reset rotation timer
        # Search for the marker with the ID matching current_marker_id
        for i in range(len(ids)):
            # Check if the ID matches the current_marker_id
            if ids[i] == current_marker_id:
                # Compute area of the marker
                area = cv2.contourArea(corners[i])
                # Display area
                org_x = int(corners[i][0][0][0])
                org_y = int(corners[i][0][0][1]) - 10
                cv2.putText(color_image, f'Area: {area:.2f}', (org_x, org_y),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

```
                cv2.putText(image_with_center_line, f'Area: {area:.2f}', (org_x,
org_y),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
                # Check area condition and handle movement
                if 0 <= ids[i] <= 5:  # Handle markers 0 to 5
                    if area < 20000:
                        # Determine left or right movement based on marker
position
                        marker_center_x = (corners[i][0][0][0] +
corners[i][0][2][0]) / 2
                        position = marker_center_x - (color_image.shape[1] // 2)
                        if position < -threshold:  # Left region
                            send_command('L')  # Move left
                            cv2.putText(color_image, 'Going Left', (org_x, org_y
+ 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                        (0, 0, 255), 2)
                        elif position > threshold:  # Right region
                            send_command('R')  # Move right
                            cv2.putText(color_image, 'Going Right', (org_x, org_y
+ 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                        (0, 0, 255), 2)
                        elif -threshold <= position <= threshold:
                            send_command('F')  # Move forward
                            cv2.putText(color_image, 'Going Forward', (org_x,
org_y + 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                        (0, 0, 255), 2)
                    else:
                        current_marker_id += 1
                        # this is for going back to original location.
                        # if current_marker_id == 1:
                        #   direction = -1  # Change direction to backward if
marker 5 is reached
                        #current_marker_id += direction  # Move on to the next
marker
                        cv2.putText(color_image, f'Reached Marker
{current_marker_id}', (org_x, org_y + 40),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0),
2)
            else:
                # Rotate towards the next marker
                send_command('C')  # Rotate clockwise
                # Display rotating message
                cv2.putText(color_image, f'Rotating to find {current_marker_id}',
(50, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
```

```
    else:
        # If the desired marker is not found, rotate until the next marker is
found
        if start_rotation_time and (time.time() - start_rotation_time < 8):
            send_command('C')  # Rotate clockwise
            cv2.putText(color_image, f'Rotating to find {current_marker_id}',
(50, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
        else:
            break

    # Check for distance value from Arduino
    #distance = read_distance()
    #if distance is not None and distance < 10:
    #    send_command('S')
    #    cv2.putText(color_image, f'Stop for a while (distance: {distance} cm)',
(50, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
    #                (0, 0, 255), 2)
    #    break

    #break the loop if current_marker_id becomes 6
    if current_marker_id == 6:
        send_command('S')
        cv2.putText(color_image, 'Destination Reached!', (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                (0, 0, 255), 2)
        break

    # Display the color and depth images
    cv2.imshow('Color Image', color_image)
    #cv2.imshow('Depth Image', depth_colormap)

    # Exit if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Stop the RealSense pipeline and close all windows
pipe.stop()
cv2.destroyAllWindows()
```