Tree Pruning/Inspection Robot Climbing Mechanism Design, Kinematics Study and Intelligent Control

A thesis presented in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

In

Mechatronics

at Massey University, Manawatu Campus,

New Zealand

Massey University

Pengfei Gui

2018

Abstract

Forestry plays an important role in New Zealand's economy as its third largest export earner. To achieve New Zealand Wood Council's export target of $12 billion by 2022 in forest and improve the current situation that is the reduction of wood harvesting area, the unit value and volume of lumber must be increased.

Pruning is essential and critical for obtaining high-quality timber during plantation growing. Powerful tools and robotic systems have great potential for sustainable forest management. Up to now, only a few tree-pruning robotic systems are available on the market. Unlike normal robotic manipulators or mobile robots, tree pruning robot has its unique requirements and features. The challenges include climbing pattern control, anti-free falling, and jamming on the tree trunk etc. Through the research on the available pole and tree climbing robots, this thesis presents a novel mechanism of tree climbing robotic system that could serve as a climbing platform for applications in the forest industry like tree pruning, inspection etc. that requires the installation of powerful or heavy tools. The unique features of this robotic system include the passive and active anti-falling mechanisms that prevent the robot falling to the ground under either static or dynamic situations, the capability to vertically or spirally climb up a tree trunk and the flexibility to suit different sizes of tree trunk. Furthermore, for the convenience of tree pruning and the fulfilment of robot anti-jamming feature, the robot platform while the robot climbs up should move up without tilting. An intelligent platform balance control system with real-time sensing integration was developed to overcome the climbing tilting problem. The thesis also presents the detail kinematic and dynamic study, simulation, testing and analysis.

A physical testing model of this proposed robotic system was built and tested on a cylindrical rod. The mass of the prototype model is 6.8 Kg and can take 2.1 Kg load moving at the speed of 42 mm/s. The trunk diameter that the robot can climb up ranges from 120 to 160 mm. The experiment results have good matches with the simulations and analysis.

This research established a basis for developing wheel-driven tree or pole climbing robots. The design and simulation method, robotic leg mechanism and the control methodologies could be easily applied for other wheeled tree/pole climbing robots. This research has produced 6 publications, two ASME journal papers and 4 IEEE international conference papers that are available on IEEE Xplore. The published content ranges from robotic mechanism design, signal processing, platform balance control, and robot climbing behavior optimization. This research also brought interesting topics for further research such as the integration with artificial intelligent module and mobile robot for remote tree/forest inspection after pruning or for pest control.

Acknowledgements

Many people contributed to the success of this research. Firstly, I would like to thank my supervisors, Dr. Liqiong Tang and Prof. Subhas Mukhopadhyay, for their advice and guidance throughout this project. I am grateful to them for giving me the opportunity to complete this Ph.D.

I would also like to thank the SEAT workshop technicians Mr. Kerry Griffiths, Mr. Ian Thomas, Mr. Clive Bardell, Mr. Morio Fukuoka, and Mr. Anthony Wade, for their precious time and great assistance on the mechanical and electronics part design and making for my projects. I have learnt a lot of electronic and mechanical skills from these people. I really appreciate their help.

I am thankful to my parents and parents-in-law for their support. A special note of thanks to my wife Shuti Hou for her support, patience, and encouragement throughout this time.

Contents

List of Figures

List of Tables

List of Abbreviations

Chapter 1 Introduction

1.1 Background

The timber industry is a significant contributor to New Zealand economy. According to a report from New Zealand Plantation Forest Industry, the total New Zealand forest product exports up to $4.8 billion for 2015 and accounts for 1.6% of the country’s GDP [1]. Although harvesting has increased significantly over the last five years, it is still hard to meet the growing overseas demand, especially the demand from China [2]- [4]. To achieve New Zealand Wood Council’s export target of $12 billion by 2022 and cope with the forestry workforce shortage and aging, both the unit value and volume of timber must be increased.

Pruning is an essential method to help obtain high-quality timber during plantation growing because the pruned lumber could produce beautiful surface without gnarl and has homogeneous quality with well-formed annual growth ring [5]. Additionally, in terms of wood quantity, under the same growth conditions, the well-pruned tree produces 2.3 m^3 of wood while the unpruned tree only produces 1.61 m^3 of wood [4]. Furthermore, high quality pruned stands, well located to the market can sell for as much as $50,000 per hectare net to the owner, while unpruned stands may net less than $10,000 particularly if logging and cartage cost is high [4]. Currently, almost all the timber pruning work is conducted by lumberjacks. The worker climbs up the tall trunk to cut off branches and gnarl with axe or chainsaw. This kind of timber pruning method is not only high labor intensity and costly, but also dangerous and low efficient. Moreover, the future shortage of forestry workforce to prune trees will certainly exacerbate this situation [4].

A solution to ease such a situation is powerful pruning tools and automatic systems. Nowadays, the fast development in robotics and automation has brought significant changes to many industries and benefit to the society [6] - [9]. However, the applications in the forestry industry have been left behind, especially in automatic tree pruning, inspection and harvest. Research work in tree pruning/inspection robot is still in the early stage which means this field worth intensive study.

This research proposed a novel wheel-driven robotic system that has potential to be used as a platform for tree pruning/inspection and developed a tree climbing mechanism for tree pruning. The unique features of this novel system include the passive and active anti-falling mechanisms which prevent the robot from falling to the ground under either static or dynamic situations, the capability to vertically or spirally climb up a tree trunk, the flexibility to suit different trunk sizes, lightweight and anti-jamming during the climbing process. The 3D CAD model of the proposed tree pruning robotic mechanism was first created using SolidWorks and then modified according to the model obtained from the 3D printer. Following were the static and dynamic climbing analysis. Robot climbing simulation was carried out in MATLAB SimMechanics based on the SolidWorks model. Meanwhile, the robot hardware and control system were designed for the platform anti-tilting control to prevent jamming on the tree trunk. A testing prototype was built to justify and evaluate the system design, control methods, and robot climbing behavior. The tests made on the testing prototype match with the simulation and the analysis.

1.2 Research Topic

The aim of this research is to make a study on climbing robotic systems including the climbing mechanism design, kinematics behavior simulation, pruning robot hardware design and control system development for the applications in the forest industry. The main objectives are:

1. Systematically study mechanical systems suitable for tree climbing robot and identify the key features, functions, and movements for tree climbing robot.
2. Analyzing the kinematic and dynamic characteristics of typical climbing mechanisms and establish a base for climbing robot design evaluation.
3. Study tree climbing and pruning process then develop the control methodologies.
4. Propose and design a tree climbing robotic system with anti-falling, anti-jamming, low climbing slippage, suitable to a certain range of trunk diameter and lightweight characters.
5. Build a physical testing system to demonstrate the features and functionalities of the proposed tree climbing robotic system and evaluate the control strategy.

6. Produce quality academic outcomes that are of value for further research in robot design for the forestry industry especially for tree pruning/inspection.

1.3 Scope of Research

Based on the research topic and the consideration of Ph.D. project time frame, the scope of this research mainly focuses on the following aspects:

- Climbing method study and analysis: simple mechanism and high climbing speed
- Anti-falling mechanism design
- Tree pruning robot kinematics study and simulation: feasibility and robustness
- Tree pruning robot mechanism optimization
- Robot hardware and software design
- Tree pruning robot tilting measurement and control for anti-jamming during the climbing procedure
- Tree pruning robot wheel slip control
- Build a physical testing prototype to verify and optimize the algorithms and control methodologies

1.4 Organization of Dissertation

This dissertation contains nine chapters and five appendices. The development of tree climbing mechanism, robot mechanical system, control and communication system. Robot hardware and software design are also presented in detail in the main chapters. The appendices include supporting information such as the robot mechanical drawing, hardware and software drawing, some of the develop code and simulation results.

Chapter 1 provides the background to the research along with the aims, objectives, and scope of the research.

Chapter 2 is a review of tree or pole climbing robots' mechanical systems and illustrations of their advantages and disadvantages.

Chapter 3 explores different design aspects and identify the essential features and functions for tree pruning robot.

Chapter 4 discusses the design and optimization of climbing mechanical systems of tree pruning robot according to kinematic study and simulation to fulfill the key features and functions of pruning robot.

Chapter 5 focuses on robot control system design including the hardware and software design.

Chapter 6 is primarily concerned with the robot platform tilting measurement methods in static and dynamic situations.

Chapter 7 provides details on the robot tilting control using fuzzy logic.

Chapter 8 discusses the robot longitudinal wheel slip control based on dynamic neural networks during the climbing process.

Chapter 9 is the discussions, the contributions of this research and the recommendations for future improvements.

Appendix A contains some of the programming codes of the climbing robot.

Appendix B presents the MATLAB SimMechanics simulations.

Appendix C contains some mechanical drawings of this climbing robot.

Appendix D presents some hardware design of the proposed robot.

Appendix E is the robot 3D CAD Model in SolidWorks.

Chapter 2 Literature Review

An extensive research was conducted to locate as much information as possible on robot design that could climb trees, poles and other surfaces. The following sections contain a review of these robot designs. Each design of the climbing robotic systems was analyzed and evaluated based on the climbing mechanism, types of climbing surface, moving speed and climbing reliability. Also, some commonly used robot climbing control methodologies like fuzzy logic and neural networks are also discussed in this chapter.

2.1 Currently Available Tree or Pole Climbing Robots

2.1.1 RiSE

The RiSE project was funded by the U.S. Defence Advanced Research Project Agency (DARPA) for the purpose of surveillance, retrieval, and inspection. Boston Dynamics Inc., in collaboration with several universities, has at this point created three versions of the RiSE robot which can climb straight up trees and wooden poles [10].

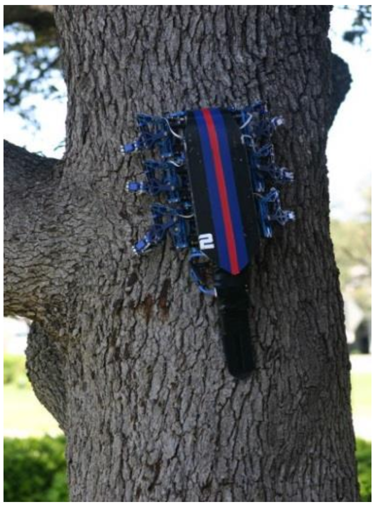


Figure 2.1 RiSE V1

RiSE V1 was first announced in 2005 [11]. Each of its six legs is actuated by two electric motors, giving each leg two degrees of freedom. The robot was tested mainly on carpeted walls to analyze and enhance its climbing ability. This robot maintains stability while climbing by using a tripod gait, meaning that at least three legs are in contact with the climbing surface at any given time. The robot maintains that grip by using a tail, which is attached to the rear of the chassis and able to push the robot. Fig. 2.1 shows how the tail works by pushing towards the climbing surface, which allows the front of the robot to remain contact with the tree.

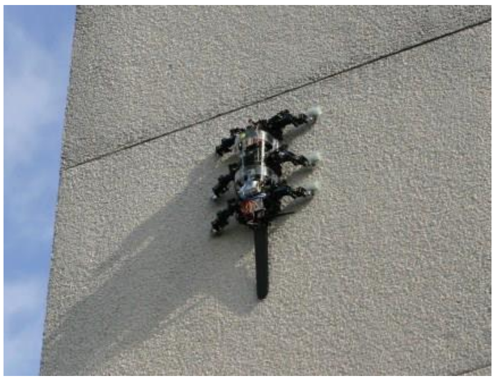


Figure 2.2 RiSE V2

RiSE V2 was the second generation which was very similar in structure to the original version [12]. It uses the same six-legged configuration and each leg is powered by two actuators. It reuses the tripod gait for climbing, in which three legs always maintain contact with the surface. This robot also has several end effector modules, which allow it to climb a variety of surfaces including outdoor walls and trees, as shown in Figure 2.2.

The gripping method for this robot takes advantage of a novel inspired gripper which includes spines made from modified medical needles installed at the end of each leg. These micro-spines covered feet can penetrate the climbing surface with minimal damage. With two degrees of actuated freedom on each leg, RiSE V2 can determine and utilize the best direction in which to apply force through the spiny feet for maximum gripping.

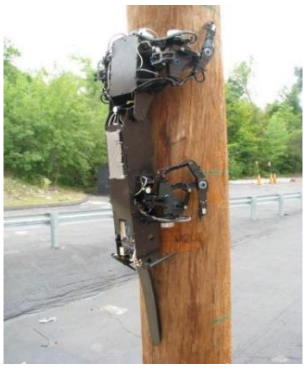


Figure 2.3 RiSE V3

RiSE V3 has some major changes from the previous versions [13]. This robot employs a Quadrupedal configuration, which means it only has four legs instead of the original six. Different brushless DC motors are used in this version to increase power. Coupled with a dramatically different leg mechanism and unique gaited behavior, this robot can climb up the telephone pole with a speed of 210 mm/s. The chassis offers another degree of freedom over the old design. A pivoting joint in the backbone of the robot allows it to adjust its upper body toward or away from the climbing surface. This gives it even more ability to adjust to the optimal gripping position during climbing as shown in Figure 2.3. Although the robot has a high climbing speed, the mechanism is not flexible and robust enough to do tree pruning work due to its complex mechanism.

2.1.2 WOODY

The WOODY project began in 2004 in the Sugano Lab at Waseda University in Japan, and since then there have been three generations of prototypes [14]. Unlike the RiSE project, the only desired application for WOODY is in forest preservation. If installing an electric saw on the top of the robot, it has the potential to do some tree pruning work.

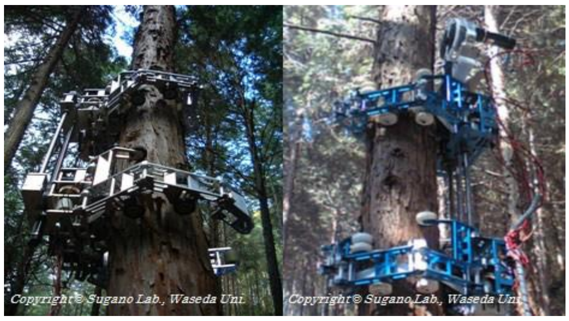

Copyright© Sugano Lab., Waseda Uni.
Copyright© Sugano Lab., Waseda Uni.

Figure 2.4 WOODY

WOODY is a manually controlled robot. It can hang on a tree trunk by wrapping its two arms around the tree trunk which is illustrated in Figure 2.4. The robot climbs vertically by extending and contracting its body using threaded rod mechanism and at the same time releases and encloses its upper and lower arms alternatively. The tree side of the arm has wheels mounted to it which allow for rotational motion. Though this robot can do some tree pruning work, the heavyweight (13.8 Kg) and wheel rotational slippery issue are the main drawbacks of this robot. Moreover, because the size of the robot has to be proportional to the circumference of the tree trunk, this feature greatly limits the application of this robot.

2.1.3 TREPA

The TREPA robot was developed at Miguel Hernandez University in 2006. This robot uses a Gough-Stewart platform with 6-DOF as the parallel climbing platform. It consists of two hexagonal rings that are linked with six linear actuators through universal and spherical joints at each end [15-16].

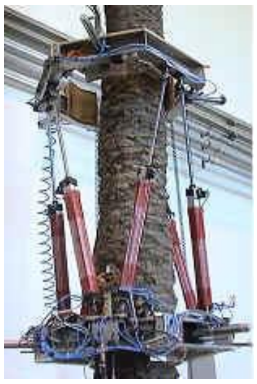


Figure 2.5 TREPA

The climbing procedure of TREPA can be generalized as a repeated four-step which is illustrated in Figure 2.6. First, the bottom hexagonal ring grips the tree and the top hexagonal ring releases its grippers. The linear actuators then extend to move the top ring to an upper position. Next, the top ring uses its grippers to grasp the tree trunk and the lower ring releases its grippers. Finally, the linear actuators contract to raise the lower ring to a higher position and grip the tree. This process is repeated to make the robot climb the trunk. One potential task of the robot is tree pruning, but the robot's heavyweight (D350-31Kg) and low climbing speed (5.5 mm/s) are concerns for such work.

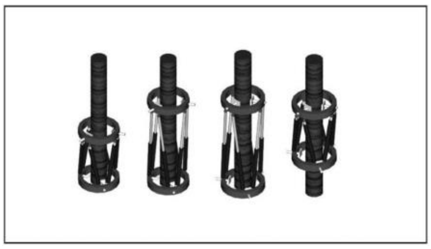


Figure 2.6 Four-step Climbing Process

2.1.4 Treebot

T. L. Lam and Y. S. Xu developed a tree climbing robot inspired by the inchworm [17-19]. This robot aims to assist or replace humans in tree-related tasks. For instance, it is able to do tree inspection tasks by climbing from tree trunk to branches with the gripper.

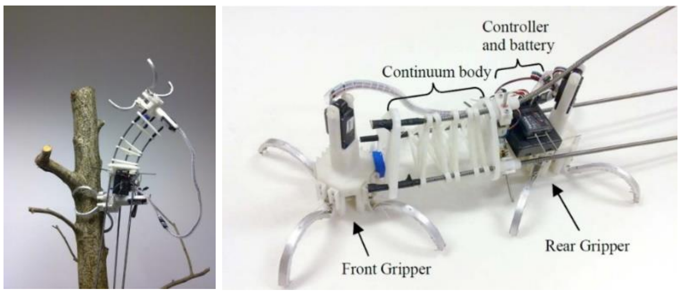


Figure 2.7 Treebot

It can be seen in Figure 2.7 Treebot is composed of three main assemblies: the controller and battery, the two grippers and the continuum body. The front and rear grippers offer the whole holding forces for the robot to strongly adhere on the tree trunk during the climbing process. The continuum body of Treebot can extend up ten times longer than its contracted length and has three degrees of freedom. It uses three mechanical springs connected in parallel, separated by 120 degrees as a rack, and combines a pinion gear attached to a DC motor to provide bendable movement.

Treebot climbs up the tree in an inchworm style motion. First, the robot anchors its rear gripper to the tree and extends its front gripper up the tree. Then the front gripper is engaged, and the rear gripper released. The continuum body contracts and raises the rear gripper to a higher position before it is reengaged with the tree. Once this process is complete it begins again and continues to move up the tree.

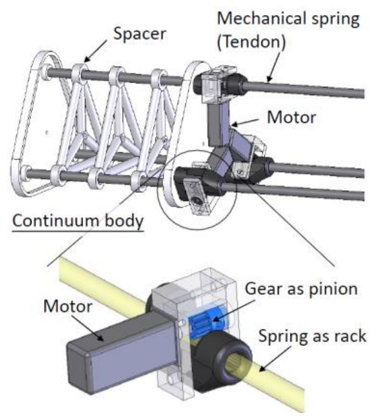


Figure 2.8 Treebot Continuum Body

Even though Treebot can afford a payload (1.75 Kg) nearly three times of its own weight, it seems the robot is not ideal to do tree pruning tasks due to its low climbing speed (12.2 mm/s) and unique mechanical structure.

2.1.5 Slider-Crank based Pole Climbing Robot

This robot was made by S.C. Lau in University Sains Malaysia in 2013 [20]. It consists of 2 main modules, which are climbing module and gripping modules. The climbing module utilizes the slider-crank mechanism for the ascending and descending motion. A DC motor is connected to the crank in order to produce rotation. When the crank rotates, the rotational motion is converted into linear motion and the input torque of the DC motor is used to overcome the friction between the slider and the end bar. One gripper is attached to the end bar and another is fixed at the end of the slider.

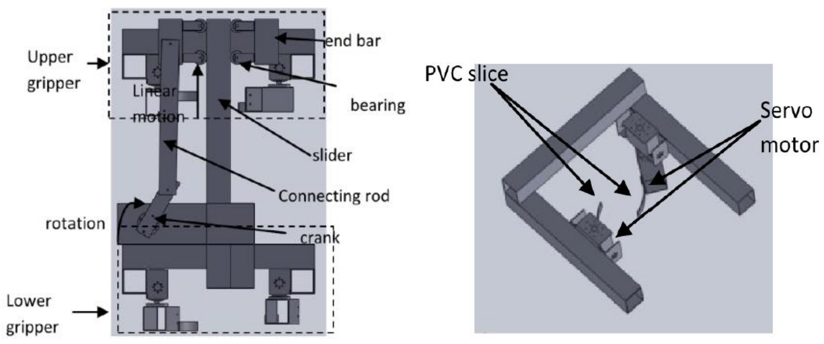


Figure 2.9 Climbing Module and Gripping Module

The gripping module consists of 2 servo motors which attached on a frame. The PVC slice is attached to an aluminum bar which is connected to the horn of servo motors as shown in Figure 2.9. If the 2 servo motors rotate in clockwise direction, the gripper will grasp the pole tightly. Otherwise, the gripper will release. To increase the friction between the gripper and pole, rubber is attached to the PVC slice. The average climbing speed of this robot is 3.7 mm/s. Obviously, such a low climbing speed is not suitable for tree pruning robot.

2.1.6 Snake Robot

Carnegie Mellon University developed a modular hyper-redundant robot that mimics the motion of a snake. Using universal joints with 3-DOF, the robot is able to move in many different ways including rolling, wiggling, and sidewinding depending on the terrain being encountered [21]. Not only can this robot move on the land, it can also climb up a tree.

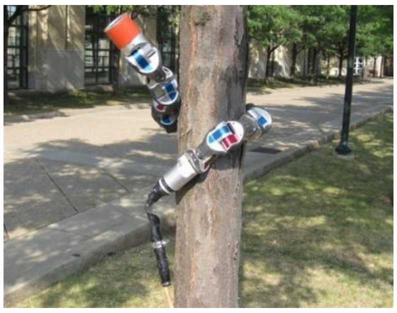


Figure 2.10 Snake Robot

The robot wraps around the tree and applies inward pressure while rolling its body to generate vertical motion up the tree. This climbing method is effective in certain situations but also has some inherent limitations. The main drawback of this robot is that the robot body has to be long enough to wrap around the tree trunk. It is not possible to do the tree pruning work.

2.1.7 Biped Wall-Climbing Robot

The biped wall-climbing robot is designed by Yisheng Guan et al. [22] to perform some high-rise tasks such as cleaning, painting, inspection and maintenance on walls of large buildings or other structures require robot with climbing and manipulating skills. The climbing pattern of this robot is inspired by the climbing motion of inchworms.

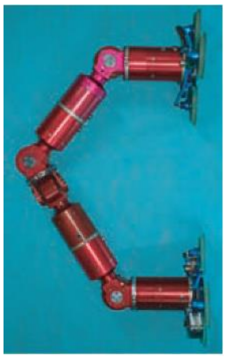


Figure 2.11 Biped Wall-Climbing Robot

From Figure 2.11, built with a modular approach, the robot consists of five joint modules connected in series and two suction modules mounted at the two ends. The advantage of this robot is that it has higher maneuverability than tree or pole climbing robot since it has more degree of freedoms. The weight of this robot is 16.1 Kg and it can climb up a flat wall at the speed of 36.7 mm/s with the payload of 1.5 Kg. However, this robot is not suitable for tree pruning task since it utilizes a vacuum system as the holding approach which cannot work efficiently on curved surfaces.

2.1.8 3D Climber

3D Climber was designed by M. Tavakoli et al. [23-24] in the University of Coimbra for developing a climbing robot with the capability of manipulating over 3D human-made structures.

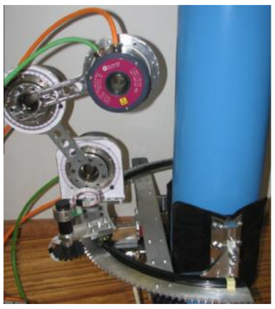


Figure 2.12 3DCLIMBER

It can be seen from Figure 2.12 that 3DCLIMBER consists of a 4-DOF serial climbing mechanism and two grippers. Unlike other developed pole climbing robots, it can overcome bends, T-junctions, flanges, and sharp changes on the pole’s diameter. The weight of 3DCLIMBER is 42 Kg and climbing speed is 16.7 mm/s. Though this robot has climbing maneuverability, it is not suitable for tree pruning work due to its heavy weight and low climbing speed.

2.1.9 UT-PCR

M. N. Ahmadabadi et al. [25-27] developed a non-holonomic wheel-based pole climbing robot to climb up a lamppost and clean the bulb. This robot employs six 1-DOF un-actuated arms with ordinary 1-DOF wheels at their tips to grasp and climb the cylindrical or near cylindrical poles. Three lower wheels are active with the upper ones being utilized to increase the robot stability. It uses the preloaded springs to produce large enough normal components to bring the lower wheels in good contact with the surface of the pole so that the wheels do not slip. This method uses a spring mechanism to generate non-automatically adjustable holding force. In addition, employing 1-DOF wheels prevents the robot from climbing the pole in a spiral pattern for the convenience of tree pruning.

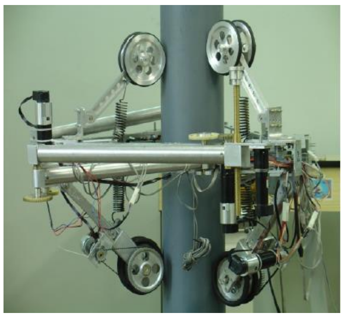


Figure 2.13 Pole Climbing Robot UT-PCR

2.1.10 Kawasaki’s Pruning Robot

A tree-pruning robot prototype was developed at the Kawasaki & Mouri Lab of Gifu University in Japan [5] [28-30]. This prototype can climb up cylindrical objects such as trees or poles. It has four wheels in contact with the trunk during climbing. Two of the wheels are located below the robot’s center of gravity and adjacent to each other. The other two are installed above the robot’s center of gravity. The mechanical structure of this robot is shown in Figure 2.14.

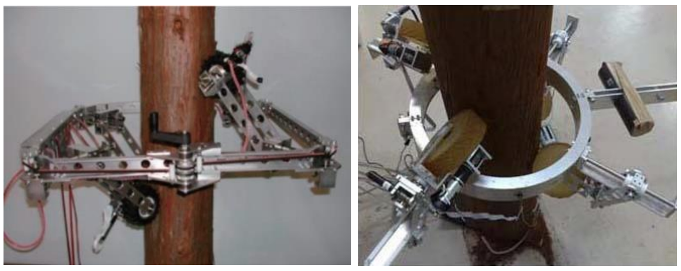


Figure 2.14 Kawasaki's Tree Pruning Robot

The biggest advantage of this robot is that there is no energy consumption when the robot keeps still on the trunk. It utilizes the friction forces generated by placing the mass center of the robot on one side of the tree to achieve this goal. The average climbing speed of this robot is 20 mm/s. Though the new version of this robot can climb a cylindrical trunk to do pruning work, the wheel slippery and flexibility to cope with different trunk diameters still need to be improved.

2.1.11 Seirei Industry's Automatic Pruning Machine

Seirei Industry's Co.'s AB232R Automatic Pruning Machine is the only available commercial tree climbing robot [31]. Because its wheels are mounted at fixed angles, the robot can only climb up a tree in a fixed spiral pattern.

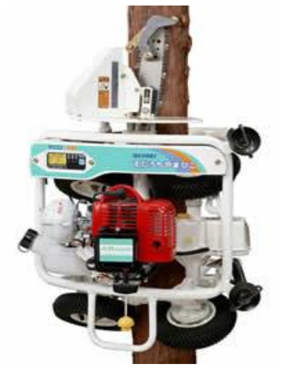


Figure 2.15 Seirei Industry's Automatic Pruning Machine

This robot generates its gripping force on the tree from preloaded springs. This type of robot gripping approach restricts the domain of tree diameters that the robot can climb. Moreover, due to its heavy weight (32.8kg), this robot is not used widely.

2.1.12 Pobot

J.C. Fauroux et al. developed a rolling self-locking robot in 2009 [32]. This robot is capable to climb up a cylindrical-conic pole to do the surveillance and inspection tasks. The mechanical structure of Pobot is illustrated in Figure 2.16.

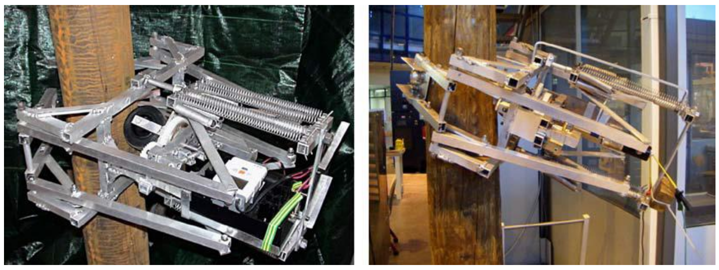


Figure 2.16 Pobot

Pobot has the similar feature as Kawasaki's robot design, which can maintain itself at a given height on the pole without energy consumption. In order to realize this feature, one necessary condition is that the center of mass of Pobot is sufficiently shifted laterally respect to the contact points between the wheel and the pole, depending on the friction conditions. Another feature for Pobot is that it can avoid obstacles on the pole by horizontal rolling.

Pobot uses six preloaded strings to produce the holding forces, but this force cannot be adjusted when the robot climbs. Therefore, in order to climb up the pole successfully, sometimes the holding force can reach to 300N and could even create grooves on the wood surface. The average climbing speed of Pobot is around 30 mm/s. The low climbing speed, complex mechanism, heavyweight (10.5 Kg) and un-adjustable holding force are the main disadvantages of the Pobot.

2.2 Tree Climbing Robot Control Methodologies

During robot climbing process, due to the complicated surface conditions, it is inevitable that the robot platform suffers tilting. To tackle this issue, many approaches have been proposed and studied in both robotics and automotive fields. These methods are based on different techniques and technologies, e.g., torque control, dynamic-model-based or vision-based control, and fuzzy logic. Each method is best suitable for its own specific system due to the characteristic of the system and the sensors used.

Another critical problem for wheel driven climbing robot is the wheel slip. When the wheel slip is not in its optimally stable zone, for tree climbing robots, the desired tire-trunk longitudinal force cannot be achieved, which not only causes distance error but also increases overall energy consumption. As a result, it directly affected the climbing performance. There are numerous research approaches in wheel slip control field. However most of them are in the automotive industry, only a handful of approaches could be found in research in forestry applications especially tree climbing robot development.

For the convenience to closely link with the control methodologies proposed in this research, the detail information on control methodologies in literature and approaches are respectively presented in chapter 7 and chapter 8.

2.3 Conclusions of Literature Study

The literature review on existing climbing robots reveals that there are only a few prototypes of climbing robotic systems available. In terms of timber pruning robot, the research and development work in literature is even less. The main reason for this status quo is that, due to the gravitational pull and the unsmooth and irregular shape of the tree surface, it is quite difficult to design a climbing mechanism to climb up a tree fast and stable to do the pruning work.

Thus, the robotic climbing mechanism design for tree or pole requires extensive and in-depth study. Chapter 3 summarizes the key features and functions that a tree pruning robot should have and illustrates a novel robotic tree climbing mechanism for tree pruning in detail.

Chapter 3 Features and Functions of Tree Pruning Robot

In terms of robot application field, climbing robot is one significant branch. However, the research and development of climbing robots is relatively more complicated and difficult comparison to other robot application areas [5]. Comparing with other robots, for tree pruning robots, the robots need to carry the power tools and execute the pruning task. At the same time, it must have the climbing capability. To realise such functions, the robot platform must keep it horizontal under unexpected dynamic situation that may cause by wheel slippage, trunk surface condition. During the last two decades, most of the research in the area of climbing robots focused on wall climbing robots (WCRs) and in-pipe robots and only limited number of pole climbing robots (PCRs) were designed and developed. Generally, the design and implementation of PCRs face more problems than those of WCRs and in-pipe robots. For instance, most WCRs [22] [33-35] could use vacuum grippers or suction cups to climb up the flat wall. However, such climbing mechanism employed on poles or tree trunks is not a desirable choice because the vacuum system cannot work efficiently on curved surfaces.

The research and study for tree climbing robots is even less as the climbing environment is more complicated such as the unsmooth surface and irregular shape of the tree trunk. Therefore, designing a safe, fast and stable tree climbing mechanism is a challenging topic that hinges on anti-falling, trunk jamming and robot platform tilting control.

3.1 Comparison and Analysis of Different Types of Tree or Pole Climbing Robots

A comparison of previously discussed robots in terms of climbing speed, weight, climbing surface, advantages and disadvantages is shown in Table 3.1.

Table 3.1 Comparison of different tree or pole climbing robots

Robot	**Climbing style**	**Pruning**	**Speed (mm/s)**	**Weight (Kg)**	**Climbing Surface**	**advantages**	**disadvantages**
RiSE V3	Step-by-step	No	210	5.4	Straight trunk	Fast locomotion	Not stable, complex
WOODY	Step-by-step	Yes	19.4	13.8	Straight trunk	Stable grip	Limited gripping range, heavyweight

TREPA	Step-by-step	Yes	5.5	31	Slightly curved trunk	High maneuverability	Heavyweight, low climbing speed
Treebot	Step-by-step	No	12.2	0.65	Curved trunk and branches	Large payload	Non-robust mechanism
Slider-crank	Step-by-step	No	3.7	_	Straight pole	Simple mechanism	Low climbing speed
Snake robot	Continuous	No	_	_	Ground, straight pole and trunk	Adaptive to different terrains	Complex, multiple actuators
Biped WCR	Step-by-step	No	36.7	16.1	Flat surface	High maneuverability	Complex, heavy weight
3D Climber	Step-by-step	No	16.7	42	Curved pipes and T type pipes	High maneuverability	Complex, heavyweight
UT-PCR	Continuous	No	_	_	Straight pole	Simple mechanism	Low maneuverability
Kawasaki	Continuous	Yes	40	15	Straight trunk and pole	Self-locking	Wheel slippery
Seirei	Continuous	Yes	_	32.8	Straight trunk	Commercial tree pruning machine	Heavyweight, wheel slippery
Pobot	Continuous	No	30	10.5	Straight pole	Self-locking	Complex, heavyweight

Table 3.1 reveals that the available robotic climbing mechanisms can be classified into two types: continuous climbing and step-by-step climbing. Continuous climbing means the robot climbs a tree trunk continuously such as the wheel-driven mechanism while the step-by-step climbing indicates the robot climbs a tree trunk in an extend-contract way. Continuous climbing robots usually take advantages of a simple structure and are faster than step-by-step based robots, but the step-by-step climbing mechanism has higher stability and flexibility. Wheel mechanism is the most typical style of continuous climbing. The advantages and disadvantages of these two climbing styles are shown in Table 3.2.

Table 3.2 Advantages and disadvantages between continuous and step-by-step climbing method

Climbing style	Advantages	Disadvantages
Continuous climbing	• High climbing speed • Simplicity mechanism	• Wheel slippery • Low maneuverability from trunk to branch
Step-by-step climbing	• Robust and high stability • High flexibility	• Low climbing speed • Complex mechanism

Taking the advantages and disadvantages of these two climbing mechanisms into consideration, the continuous climbing method is more suitable for tree pruning robot as high climbing speed is preferred.

3.2 Requirements for Tree Pruning Robot Design

From the literature review on the existing climbing mechanism and robot structure analysis, the basic requirements for tree pruning robot design become clear. This research aims to develop a novel tree climbing robotic system that is able to be a base for tree pruning power tools, climb vertically or in a spiral pattern, be used on a range of trees with a certain range of diameters, climb in different speed and never fall freely to the ground. While the tree grows the diameter is increased. It is hard to use one robot with a fixed size to suit a whole range of trees with different diameters. For this research, the aim focuses on the pruning of the 3 to 5 years young pine trees. The diameters are mainly in the range of 120 to 160 mm. Based on such desired functionalities and requirements, the novel tree climbing robotic mechanism must have the following features.

- Ability to carry a payload
- Climb in different patterns
- Cope with certain range size of tree trunk
- Able to climb tree trunk with a relatively high speed and the speed is adjustable
- Anti-falling for static and dynamic
- Lightweight as to facilitate transportation

The overall mechanical design of the tree pruning robot and its optimization are discussed in the next chapter.

Chapter 4 Mechanical Design and Optimization of Tree Pruning Robot

Design is the process by which the needs of the customer or the marketplace are transformed into a product satisfying these needs. Figure 4.1 displays general systematic approaches of the design process and Figure 4.2 shows the approach used in the design process of tree pruning robot. All 3D models demonstrated in this thesis were modelled in SolidWorks.

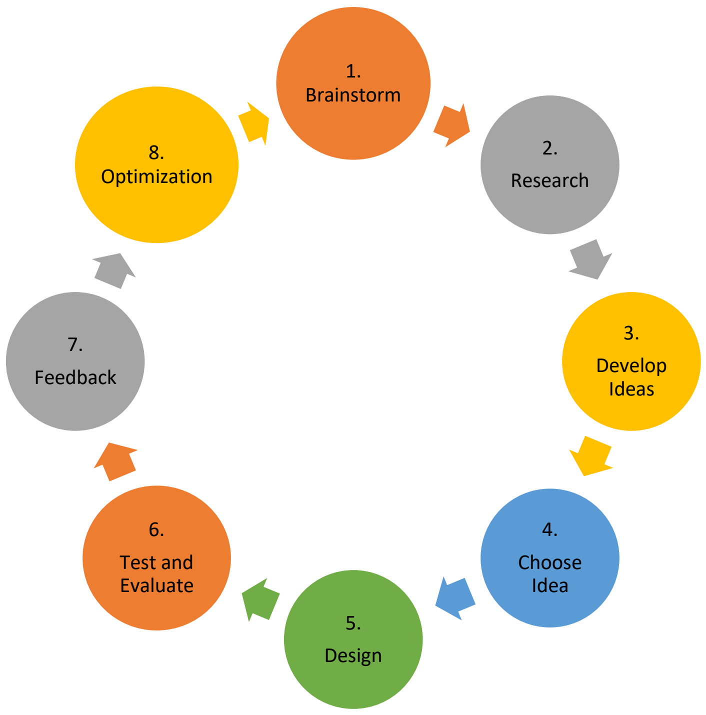
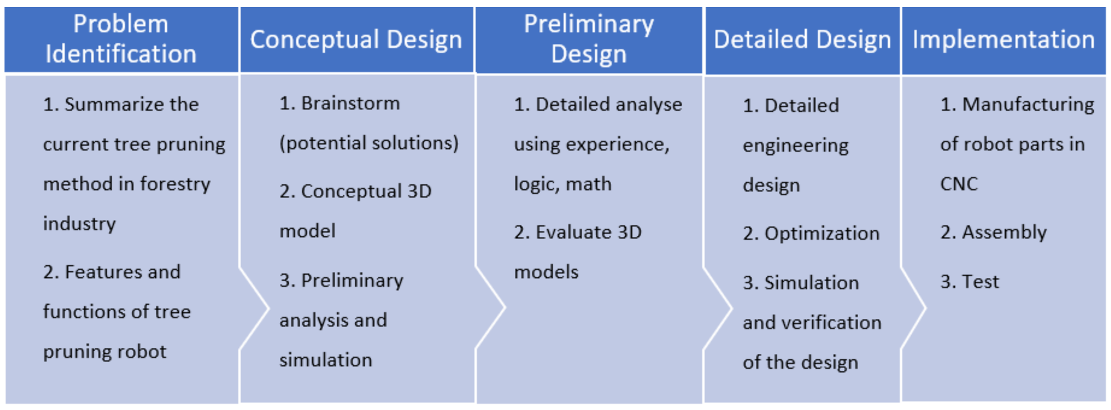


Figure 4.1 Systematic approaches of design process



Problem Identification	Conceptual Design	Preliminary Design	Detailed Design	Implementation
1. Summarize the current tree pruning method in forestry industry 2. Features and functions of tree pruning robot	1. Brainstorm (potential solutions) 2. Conceptual 3D model 3. Preliminary analysis and simulation	1. Detailed analyse using experience, logic, math 2. Evaluate 3D models	1. Detailed engineering design 2. Optimization 3. Simulation and verification of the design	1. Manufacturing of robot parts in CNC 2. Assembly 3. Test

Figure 4.2 Systematic approach utilized in the design process of tree pruning robot

In this chapter the conceptual design is firstly presented. Following is its static analysis in three-dimensional space. Then the conceptual model is simulated in MATLAB SimMechanics[1]. Based on the outcomes of the simulations, the conceptual model is redesigned and optimized. After that the static and kinematic analysis of the optimized model is presented in detail and verified in SimMechanics. Finally, a comparison of the conceptual and optimized model is conducted.

4.1 Conceptual Design of Tree Pruning Robot

4.1.1 Mechanical Construction of Tree Pruning Robot Conceptual Design

Base on the analysis of existing tree or pole robotic climbing mechanisms and the basic requirements for tree pruning robot, a conceptual design was proposed as shown in Figure 4.3. In order to simplify the mechanical design and obtain relatively high climbing speed, the wheel-driven climbing method was adopted.

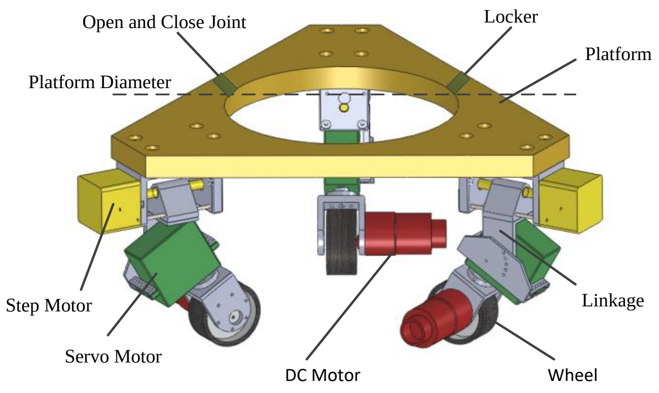


Figure 4.3 Mechanical Construction of conceptual tree pruning robot design

[1] "SimMechanics 2 User's Guide," MathWorks Inc, http://www.mathworks.com.au/products/simmechanics/.

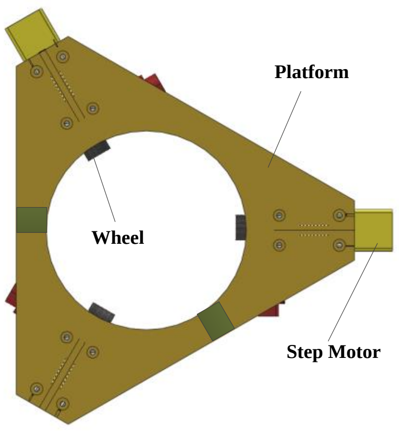


Figure 4.4 Top view of tree pruning robot conceptual design

This conceptual model consists of two parts: the platform and three legs. The design consideration is that, in the future, the chainsaw and control system could be easily installed on the platform. The platform plays a carrier role which also connects the robot driving system with the chainsaw execution system in the process of tree pruning. The platform is constructed by two parts. The two parts are joined through an open-close joint. Once the robot is set up on the tree trunk, the platform is closed and locked by a locker as shown in Figure 4.3. Under the platform, there are three legs evenly distributed around the platform and these legs support the entire robot. The robot legs are also the tree pruning robot’s driving mechanism which is composed of five major units: stepper, servo motor, DC motor, linkage and wheel unit.

Two step-motors are installed on two legs of the robot under the platform as shown in Figure 4.3. These two step motors together with the two nut and screw units are the major mechanism to adjust the diameter constructed by the robot wheels to suit a certain range of different sizes of tree trunk. The robot leg without installing step motor is a fixed leg with a wheel against the tree trunk.

The function of the three servo motors is to change the robot climbing morphology. Except for vertical climb, the robot can also use these three servo motors to climb the trunk spirally at different spiral angles. The three DC motors provide the power to drive the robot up and down along the tree trunk.

4.1.2 Static Analysis of Conceptual Model in Three-dimensional Space

Because there are only two step motors installed under the platform, this uneven mass distribution leads to the robot center of mass not coinciding with the center of the platform. Thus, under static conditions, the conceptual model has the capability to hang itself on the trunk without consuming any energy and prevent free fall to the ground.

The two step-motors can also do fine adjustment to tune the distance between the trunk center and the center of the robot wheel, which controls the normal forces applied on the three robot wheels. When the stepper motor moves towards to the trunk, the tire contact area with the trunk and the corresponding normal force increase. As a result, the normal forces provide the frictional force by multiplying the friction coefficient, which overcomes the robot gravity force and other resistance to make the robot climbing up.

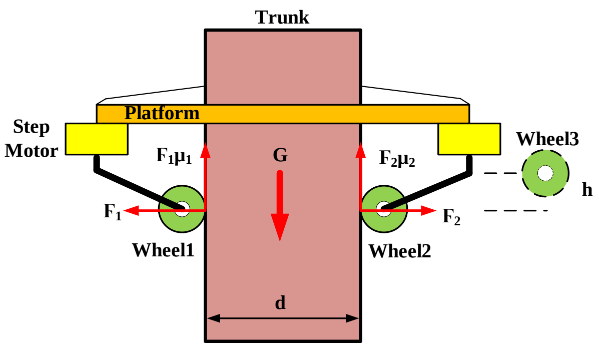


Figure 4.5 Lateral view of conceptual model force analysis

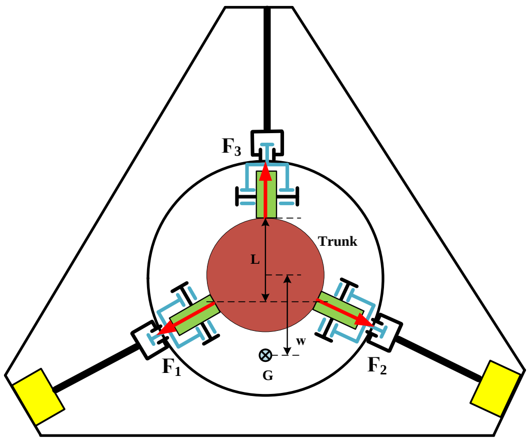


Figure 4.6 Top view of conceptual model force analysis

Figure 4.5 and Figure 4.6 illustrate the lateral and top view of the conceptual model force analysis. They represent the situation when the robot is in static status and the platform keeps still on the trunk by gravity force without consuming energy. Then the equilibrium status of the forces applied on the robot meet the following conditions based on the fundamental principle of mechanics (assuming three wheels distributed evenly around the trunk).

$$F_1\mu_1 + F_2\mu_2 + F_3\mu_3 = G \tag{4.1}$$

$$\frac{1}{2}F_1 + \frac{1}{2}F_2 = F_3 \tag{4.2}$$

Where F_1, F_2 and F_3 are normal forces applied on the three wheels at the contact points between the wheels and the tree trunk, μ_1, μ_2 and μ_3 are the friction coefficients, G represents the gravity force of the robot.

Assuming that when the robot keeps still on the trunk, as wheel 1 and 2 each has a step motor on the robot leg, it makes the gravity center of the robot has an offset from the trunk center as shown in Figure 4.6. Under equilibrium condition, assuming wheel 1 and 2 stay at the same level which is lower than the position of wheel 3. And the equilibrium of the moment at the upper wheel (wheel 3) contact point gives:

$$G\left(w + \frac{d}{2}\right) = \frac{1}{2}F_1h + \frac{1}{2}F_2h + F_1\mu_1L + F_2\mu_2L \tag{4.3}$$

Where d is the diameter of the trunk, L is the horizontal distance from the contact point between wheel 3 and the trunk to the line defined by the similar contact points of wheel 1 and 2, h is the vertical distance between wheel 3 and wheel 1, 2. w is the robot gravity force offset from the center of the platform as shown in Figure 4.6.

For tree climbing, assuming that the working condition for three wheels is similar which means $\mu_1 = \mu_2 = \mu_3 = \mu$, according to Eqn. (4.1) - (4.3),

$$L = \frac{1}{2}d + \frac{1}{4}d = \frac{3}{4}d \tag{4.4}$$

$$F_3 = \frac{G(2\mathrm{w}+\mathrm{d})}{3d} = \frac{G}{3\mu} \tag{4.5}$$

$$\mu = \frac{h}{3w+1.5d-2L} \qquad (4.6)$$

This indicates that to keep the robot still on the trunk by using the frictional force (increasing friction coefficient) without consuming the robot power, a larger value of h is more desirable.

4.1.3 Simulation of Conceptual Design Model

The mechanical system of the conceptual model is designed in SolidWorks 2015. Every part is assigned with its material property and geometric dimensions. Some of the part and assembly property information is summarized in table 4.1.

Table 4.1 Specifications of tree pruning robot conceptual design

Parameters	**Values**
Platform Parts Mass Density	2700 kg/m^3
Wheel Mass Density	1246.5 kg/m^3
Robot Mass	8760.81 g
Volume	2013051.28 cubic mm^3
Surface area	431292.27 mm^2
Center of mass	[X Y Z] = [3.81 6.60 -36.01] mm

Table 4.1 shows that the mass of the pruning robot is 8.76 Kg. That means its gravity force is 85.94 N (the acceleration of gravity takes 9.81N/kg). Therefore, in the climbing process, in order to prevent the robot from falling down, the total tractive force offered by the three wheels of the robot has to be greater than the robot gravity force. In the vertical climbing situation, the tractive force is the sum of the frictional forces applied to the wheels. Under the ideal circumstance, assuming that the frictional force is uniformly distributed on the three wheels. For the conceptual model, the value of the friction force on each wheel should be 28.65 N.

The dynamic simulation and analysis of the tree pruning robot conceptual model are conducted using MATLAB SimMechanics. Figure 4.7 shows the four views of the pruning robot in SimMechanics.

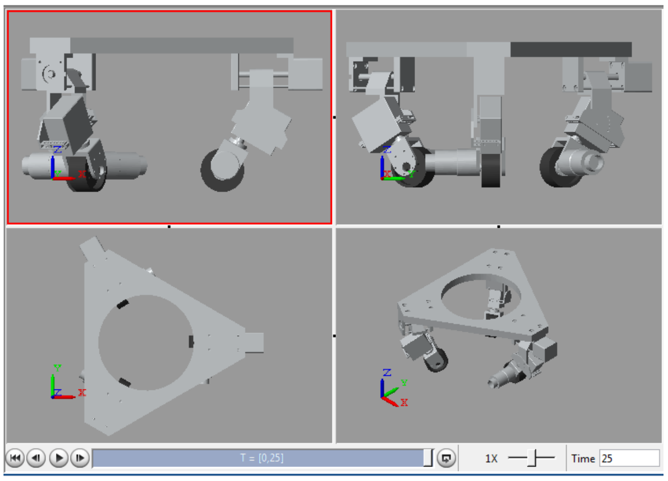


Figure 4.7 Four views of tree pruning robot conceptual design in Simmechanics

In order to make the simulation more realistic, the frictional force is set as simple Gaussian noise so that the wheel slippage can be considered as disturbance. The frictional force parameters are shown in Table 4.2.

Table 4.2 Parameters of frictional force

Parameters	**Values**
Frequency	25 Hz
Mean	28.66 N
Standard Deviation	0.2

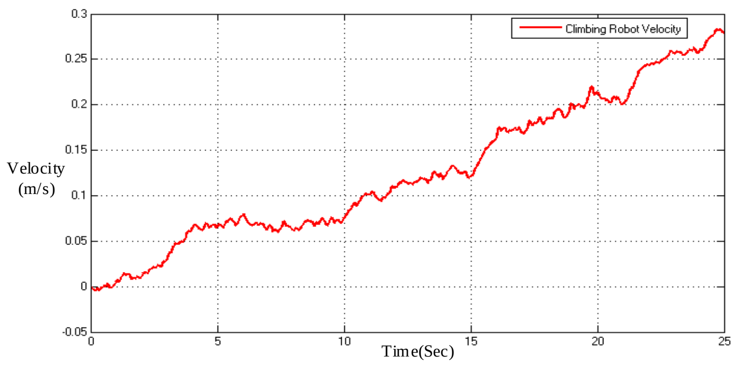


Figure 4.8 Conceptual model vertical climbing velocity

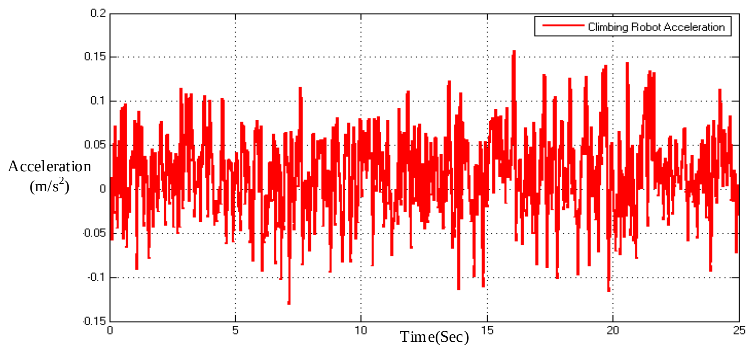


Figure 4.9 Conceptual model vertical climbing acceleration

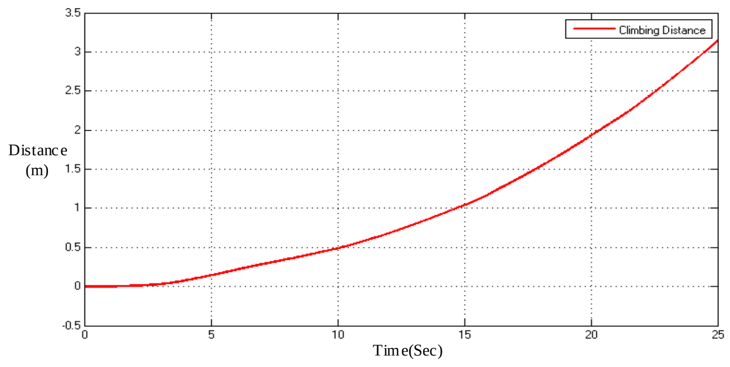


Figure 4.10 Conceptual model vertical climbing distance

At simulation time 0, three frictional forces were applied on three wheels respectively. Figure 4.8, 4.9 and 4.10 are the outcome of the simulation of the conceptual model's vertical climbing velocity, acceleration and distance according to the artificial frictional forces. It can be seen that the velocity and distance are increasing over time. This is because the total average frictional force is slightly larger than the robot gravity force. During the simulation process, the maximum of the velocity is 0.2795 m/s and the relative distance is 3.1383 m. The average velocity of the pruning robot is 0.126 m/s in 25 seconds.

Though this conceptual design has some tree pruning robot required features like anti-falling, cope with a certain range of trunk sizes and climbing in different patterns, it also has some drawbacks. For instance, the mechanical design of robot leg is not robust enough to support the whole robot when climbing up the trunk. It is more vulnerable to suffer wheel slippage which greatly impacts on the robot climbing performance. Another issue is the point-to-point connection method between the servo motor and DC motor.

The servo motor can easily get stuck when the normal force is large, or the robot suffers disturbances. Furthermore, aiming to design a lightweight tree pruning robot, the robot weight can be reduced by reconstructing the robot platform frame.

4.2 Tree Pruning Robot Design Model Optimization

4.2.1 Anti-falling Mechanism Optimization

In reality, for any tree climbing robots, the most important feature is the robot never falls to the ground in any situation: not only during the climbing process, but also in the extreme conditions such as robot power cut off, wheel malfunction, etc. Based on such considerations, a conceptual model was proposed (Figure 4.3), which uses two steppers and screw-nut units to fulfill this active anti-falling function.

The working principle is that, during the robot climbing process, when the travel nut moves towards the tree trunk, the friction forces between the wheel tread and the tree trunk increases. The adjustable friction forces guarantee the robot not falling. However, there are several shortages applying two-stepper systems. Firstly, the two-stepper climbing method increases the total weight of the robot comparing to one stepper system. Furthermore, the two-stepper climbing approach requires not only the synchronization of the two steppers which means complicated control and more MCU computational resource occupation but also more battery power consumption.

Therefore, to improve this situation, one-stepper climbing strategy is considered. In this approach, only one robot leg is equipped with step motor system which could make adjustment on the robot wheel. The other two wheels are stationary.

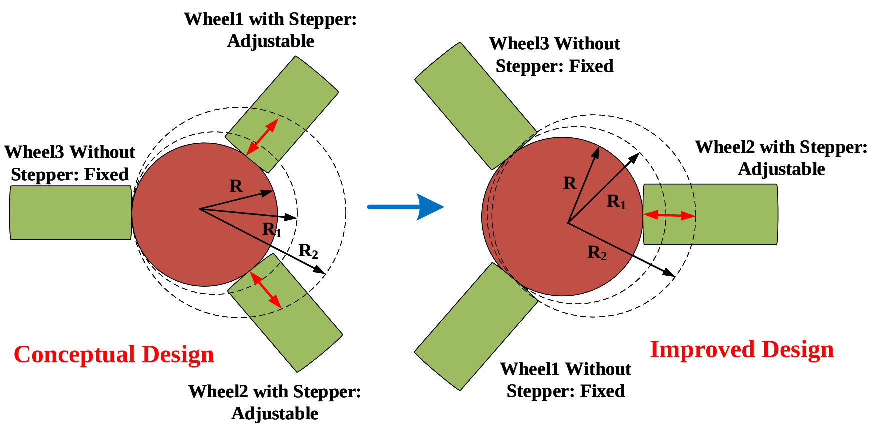

Wheel1 with Stepper: Adjustable
Wheel3 Without Stepper: Fixed
R
R_1
R_2
Conceptual Design
Wheel2 with Stepper: Adjustable
Wheel3 Without Stepper: Fixed
Wheel2 with Stepper: Adjustable
R
R_1
R_2
Improved Design
Wheel1 Without Stepper: Fixed

Figure 4.11 Improvement of tree pruning robot mechanical design

Figure 4.11 illustrates the improvement of the climbing robot mechanical design. R, R1 and R2 stand for different radii of tree trunk while the dash circles represent the different sizes of tree trunk. In conceptual design, two-step motors together with the two nut and screw units adjust the diameter of the circle constructed by robot wheels to suit a range of trunk sizes while this is simplified by one stepper in the optimized mechanical design. The stepper's screw-nut unit is shown in figure 4.12. During the robot climbing process, the stepper adjusts the normal forces between the robot wheels and the tree trunk through the screw and nut mechanism. The other two legs are fixed legs with the wheel against the tree trunk without installing step motors.

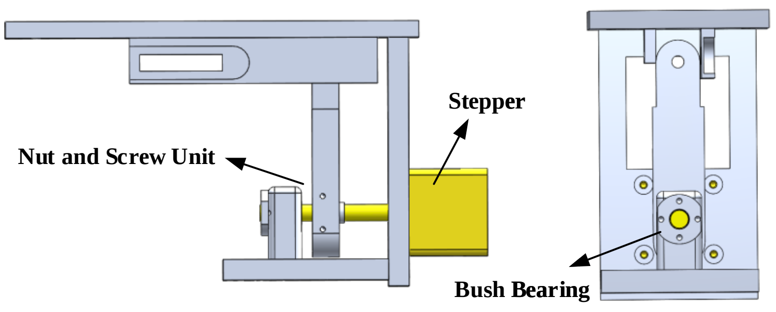


Figure 4.12 Stepper and Nut-Screw Unit

4.2.2 Servomotor Module Optimization

To perform the tree pruning work, each robot wheel is installed a servomotor module. The robot uses these modules to change the climbing morphology. Except for vertical climbing, the robot can also climb the trunk in a spiral pattern by turning the wheels at given angles. However, during the spiral or vertical climbing process, sometimes the servomotor gets stuck due to the high normal force and other disturbing forces between the wheels and the trunk in the conceptual design.

To solve this issue, one servomotor module and one bearing support mechanism are designed which is illustrated in Figure 4.13. This servomotor module consists of three parts: rotary plate, support plate and lock plate. When the servo motor rotates, the rotary plate connected with servomotor shaft drives the wheel to rotate. During the servo motor rotate process, the normal force is absorbed by the support plate and the servomotor holding bracket. Hence, the servomotor only needs to overcome the friction force between the rotate plate and support plate to swing. The friction force is relatively small compared to the normal force since the contact area between rotary plate and support plate is small and well lubricated.

The bearing support mechanism consists of one ball bearing, bearing shaft, bearing linkage and guide rail. Its function is to overcome the robot wheel module gravity force and other disturbing forces that affect the servomotor module and stepper's nut and screw unit. This guarantees the lead screw efficiency and control accuracy of the step motor.

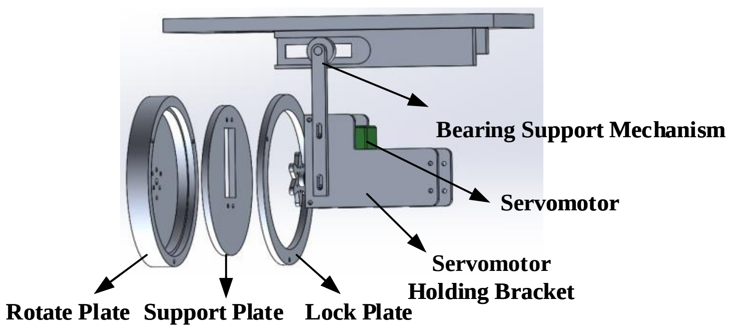


Figure 4.13 Servo Motor Module and Bearing Support Mechanism

4.3 Optimized Tree Pruning Robot Model

The modified climbing robot design is displayed in Figure 4.14 and one of the robot legs is illustrated in Figure 4.15. It only relies on one stepper system to adjust robot holding force and suit different tree diameters. Three servomotor modules are installed to change the robot climbing morphology. The three geared DC motors provide the power to drive the robot up and down along the tree trunk. Each of the geared motors has an encoder to control the climbing speed. To increase the robot wheel friction force, the improved robot uses 120×60 mm size rubber tire to replace the original 56×25 mm tire.

To achieve the lightweight feature without sacrificing the strength of the robot mechanical construction, the platform is also redesigned by cutting off the unnecessary materials on the top plate. The mass of the improved robot model in SolidWorks is now around 6.42 Kg while the conceptual model was 8.76 Kg.

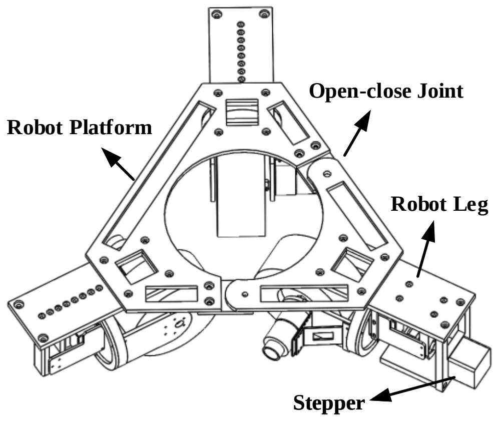


Figure 4.14 Mechanical Construction of the Optimized Tree Pruning Robot Design

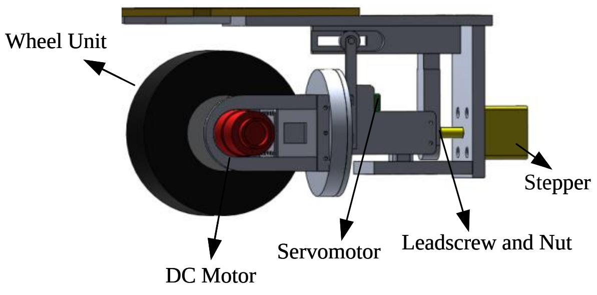


Figure 4.15 Climbing Robot Leg Module

4.3.1 Optimized Tree Pruning Robot Static Analysis in Three-dimensional Space

Because the purposely installed one step motor under the platform, the uneven mass distribution leads to the robot center of mass not coincides with the center of the platform. Thus, under static situations, the platform tilts from the horizontal position. This feature makes the climbing robot has the potential to hang itself on the tree trunk without consuming any power and prevent free falling to the ground when power is cut off. This means the optimized model also has the passive anti-falling feature.

The step motor can also do fine adjustment to tune the distance between the tree trunk and the tread of the robot wheel, which controls the values of the normal forces applied on the three robot wheels. When the stepper motor moves closer to the tree trunk, the tire contact area with the trunk and the corresponding normal force increases. When the frictional forces obtained by multiplying the normal forces and the friction coefficient overcome the robot gravity force, the robot starts to climb up.

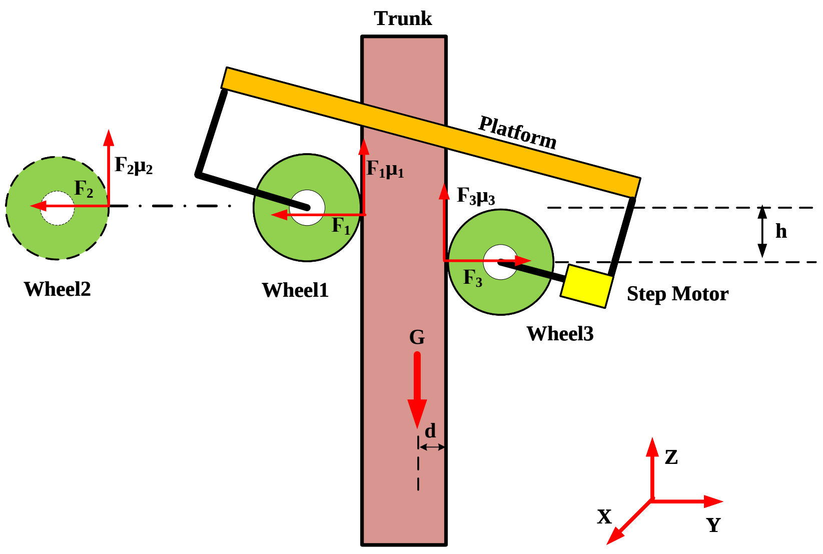


Figure 4.16 Lateral View of Climbing Robot Force and Moment Illustration

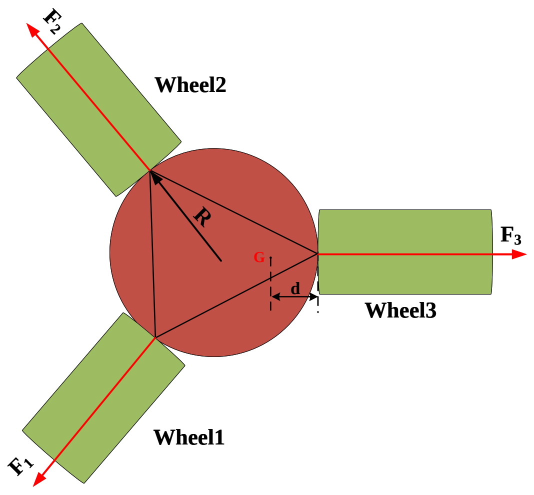


Figure 4.17 Top View of Climbing Robot Force and Moment Illustration

Figure 4.16 and Figure 4.17 illustrate the lateral and top view of the forces and moments applied on the optimized robot when it is in static status and the platform is hung on the trunk by gravity force without consuming power. Then the equilibrium status of the forces applied to the robot meet the following conditions based on the fundamental principle of mechanics.

$$\sum F_H = \vec{F}_1 cos60° + \vec{F}_2 cos60° + \vec{F}_3 = 0 \qquad (4.7)$$

$$\sum F_V = \vec{F}_1 \mu_1 + \vec{F}_2 \mu_2 + \vec{F}_3 \mu_3 = mg \qquad (4.8)$$

Where $\vec{F}_1$, $\vec{F}_2$and $\vec{F}_3$ are normal forces applied on the three wheels at the contact points between the wheels and the tree trunk, μ_1, $\mu_2$and μ_3 are the static friction coefficients, F_H and F_V are the resultant horizontal and vertical forces, m represents the mass of the robot.

When the robot hangs on the tree trunk, as wheel 3 has a step motor on the robot leg, it makes the gravity center of the robot has an offset from the trunk center (tilting) as shown in Figure 4.16. Under the equilibrium condition, wheel 1 and 2 stay at the same level which is higher than the position of wheel 3. And the equilibrium of the moment at the lower wheel (wheel 3) contact point (in the case of the wheel not rolling) gives:

$$\overrightarrow{F_1} cos60°h + \overrightarrow{F_2} cos60°h - \overrightarrow{F_1} \mu_1 \frac{3}{2} R - \overrightarrow{F_2} \mu_2 \frac{3}{2} R + mgd = 0 \qquad (4.9)$$

Where d is the length from the surface of the trunk to the center of mass of the robot, R is the radius of the trunk, h is the vertical distance between the upper and lower wheels.

For tree climbing, the working condition between the tree trunk and the three robot wheels is almost similar. Therefore, assuming that $\mu_1 = \mu_2 = \mu_3 = \mu$, according to Eqn. (8),

$$(F_1 + F_2 + F_3)\mu = \mathrm{mg} \qquad (4.10)$$

From Eqn. (7),

$$\frac{1}{2} F_1 + \frac{1}{2} F_2 = F_3 \text{ or } F_1 + F_2 = 2F_3 \qquad (4.11)$$

From Eqn. (4.10) and (4.11),

$$F_3 = \frac{mg}{3\mu} \qquad (4.12)$$

From the moment equation,

$$(F_1\cos60° + F_2 cos60°)\mathrm{h} - (F_1 + F_2)\frac{3}{2}R\mu + mgd = 0 \qquad (4.13)$$

From Eqn. (4.11), (4.12) and (4.13),

$$F_3 h - 3R\mu F_3 + mgd = 0 \qquad (4.14)$$

$$\therefore\ \mu = \frac{h}{3(R-d)} \qquad (4.15)$$

This means the static friction coefficient can be represented as a function of robot mass center location and the vertical distance between the upper and lower wheels. In this case, to stay on the tree trunk by robot’s own weight without consuming any power, the larger distance between upper and lower wheels is more desirable.

However, for wheel climbing mechanism, there is another issue. When the robot is tilted on the tree trunk, if the wheels can still roll because of the gravity force, then the robot will fall to the ground. How to stop the wheel rolling must be considered. Taking the lower wheel in Figure 4.16 as an example to illustrate this problem, the force and moment analysis is shown in Figure 4.18. Due to the robot gravity G, static frictional force F_f and normal force F, the wheel contact surface was changed. That means the normal pressure in the upper half of the contact patch (shaded area) is higher than that in the lower half. Therefore, the resultant force of the pressure distribution is offset towards the upper of the contact patch. The offset force produces the rolling moment M that gives the robot a rolling downtrend.

$$\mathrm{M} = \mathrm{G} \cdot r_s + F \cdot \lambda \qquad (4.16)$$

r_0 is the wheel free radius (60 mm) while r_s is the static radius of the robot wheel under loaded condition, the detail of wheel radius will be discussed in the next section; F_f is the static friction force; N is the applied horizontal force; λ is the vertical offset distance of the resultant normal force F due to the tire deformation.

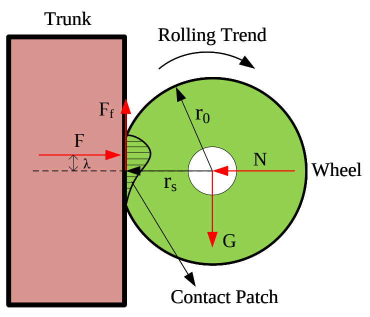


Figure 4.18 Static Rolling Moment Analysis

To offset this rolling moment, the three servo motors turn the three DC motors certain degrees to stop the wheel rolling. The exact servomotor rotating angles depends on the size of the tree trunk, robot wheel dimension, condition of trunk condition, etc. While in extreme situation, the servomotor can turn the DC motor together with the wheel 90 degrees, which means the axis of the trunk is parallel to the axis of the wheel as shown in figure 4.19.

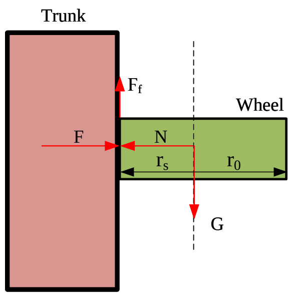


Figure 4.19 Anti-wheel-rolling Approach

4.3.2 Optimized Tree Pruning Robot Dynamic Analysis of Vertical Climbing

For vertical climbing, before the robot begins to climb up, the three servomotors first turn the three wheels' axes horizontal to allow the robot to climb the tree vertically. Assuming the stepper shaft is at a proper position, which means no adjustment is needed and the normal forces are big enough to drive the robot up, also assuming the friction coefficient is the same for the three wheels during the climbing process, then the three DC motors that driving the three wheels need to overcome the rolling resistance, robot gravity, acceleration resistance, etc.

1) Rolling Resistance

Rolling resistance, sometimes called rolling friction or rolling drag, is the force resisting the motion when a body roll on a surface. In this case, the rolling resistance is mainly caused by the hysteresis losses. Another cause of rolling resistance lies in the slippage between the wheel and the surface, which dissipates energy[2].

Figure 4.20 is a simplified wheel tire model. Considering an elastic wheel which is composed of many spring-damper units. When the wheel tries to rotate on the tree surface, the frictional force F_f and resultant normal force F will occur on the contact patch (slash area in Fig. 4.20). Additionally, the tire deforms since the stiffness of the trunk is much larger than the tire. When the tire enters the contact patch, the spring-damper unit is initially compressed and then released. This compress and release process dissipate internal energy in the way of frictional heat and is defined as hysteresis losses.

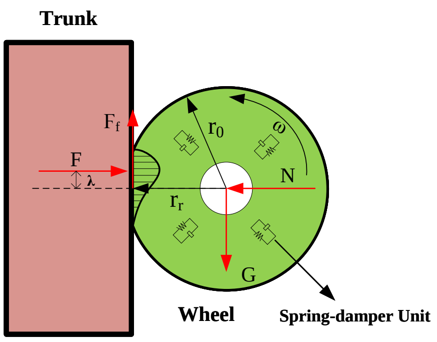


Figure 4.20 Simplified Climbing Robot Wheel Tire Model

Figure 4.21 illustrates the hysteresis losses. Curve A refers to the compress process while curve B is the release process. The area between the curve A and B is the hysteresis losses [36][37].

[2] "Rolling Resistance," Wikipedia, last modified December 15, 2016. https://en.wikipedia.org/wiki/Rolling_resistance.

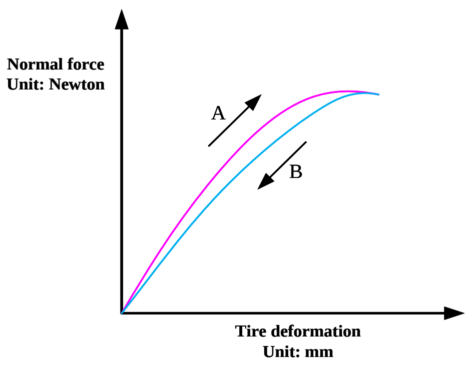


Figure 4.21 Climbing Robot Wheel Rolling Resistance: Hysteresis Losses

2) Rolling Radius

There are three types of wheel radii: free radius r_0, static radius r_s and rolling radius r_r. The radius, when the robot wheel is under unloaded condition, is the free radius r_0. In static condition, when the robot wheel deforms under loaded situation, the radius is called static radius r_s. Obviously, the value of the static radius r_s depends on the static load. In kinematic situation, the ratio of the robot wheel displacement velocity v and wheel rotational velocity ω is called rolling radius r_r [37].

$$r_r = \frac{v}{\omega} = \frac{v \times \Delta t}{\omega \times \Delta t} = \frac{s}{2 \times \pi \times n_\omega} \qquad (4.17)$$

Where n_ω and s are the robot wheel rotational cycles and the distance in time Δt respectively.

There is another method to calculate the wheel rolling radius r_r. Assuming the wheel rotates angle Δφ and moves distance Δx, then the rolling radius r_r can be obtained from Figure 4.22.

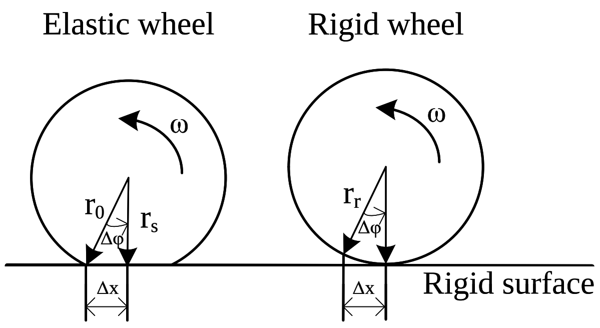


Figure 4.22 Climbing Robot Wheel Rolling Radius

$$r_0 sin\Delta\varphi = \Delta x \qquad (4.18)$$

$$r_0 \cos\Delta\varphi = r_s \qquad (4.19)$$

Comparing the elastic wheel with the rigid wheel in the above figure, it can get:

$$\Delta\mathrm{x} = r_r\Delta\varphi \qquad (4.20)$$

$$r_r = \frac{r_0 sin\Delta\varphi}{\Delta\varphi} \qquad (4.21)$$

When $\Delta\varphi \to 0$, then $r_r = r_0$. If $\Delta\varphi$ is small enough, the sine function can be expanded using series.

$$r_r = \frac{r_0(\Delta\varphi - \frac{1}{6}\Delta\varphi^3)}{\Delta\varphi} = r_0(1 - \frac{1}{6}\Delta\varphi^2) \qquad (4.22)$$

For cosine function Eqn. (4.19), expanded in series:

$$r_s = r_0\left(1 - \frac{1}{2}\Delta\varphi^2\right) \text{ or } \Delta\varphi^2 = 2(1 - \frac{r_s}{r_0}) \qquad (4.23)$$

From Eqn. (4.22) and (4.23), it can get [38]:

$$r_r = \frac{2}{3}r_0 + \frac{1}{3}r_s \qquad (4.24)$$

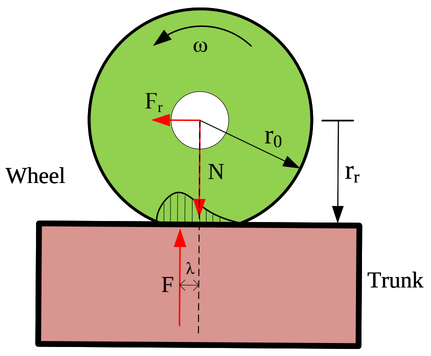


Figure 4.23 Free Rolling at Horizontal Trunk

In Figure 4.23, suppose a robot wheel is free rolling on a horizontal surface, from the moment equilibrium, it can get:

$$F_r r_r = F\lambda \qquad (4.25)$$

From the Eqn. (4.25), the rolling resistance F_r is proportional to the wheel normal force F.

Therefore, when the robot stays on the tree trunk the robot wheel radius uses r_s while it uses the r_r for the kinematic analysis during the climbing process.

3) Robot Gravity Force

Robot mass is a critical factor in climbing robot design as it directly affects the value of the robot gravity force. Under different situation, this gravity force can accelerate or decelerate the robot speed and the mechanical behavior of the mechanism. The total mass of the presented tree climbing robot is approximately 6.42 Kg in CAD design, while the actual total robot mass is 6.8 Kg (about 66.7 Newton) because of the contribution from PCB boards and other electrical components.

4) Acceleration Resistance

At the beginning of the tree climbing from a static condition, the three DC motors need to overcome the inertia force caused by the robot mass during the accelerated movement. As shown in Figure 4.24, the inertia force is deemed the acceleration resistance F_j.

$$F_j = m\frac{dv}{dt} \qquad (4.26)$$

The inertia torque T_j caused by the acceleration resistance is:

$$T_j = I\alpha = F_j r_r \qquad (4.27)$$

Where I is the wheel rotational inertia; α is the wheel angular acceleration and r_r is the wheel rolling radius.

As the robot climbs up, the DC motor's torque should overcome the rolling resistance, robot gravity and acceleration resistance.

$$F_M = F_r + G + F_j \qquad (4.28)$$

$$\mathrm{T}_M = T_r + T_G + T_j = F \cdot \lambda + \mathrm{G} \cdot r_r + F_j \cdot r_r \qquad (4.29)$$

F_M and T_M are the force and torque required from the DC motor respectively. T_r is the rolling resistance torque; T_G is the robot gravity torque; F_r is rolling resistance; G is robot gravity force and λ is vertical offset distance of the resultant normal force F due to the tire deformation.

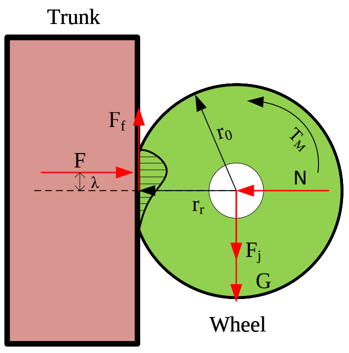


Figure 4.24 Force and Torque Analysis on one robot leg

As the tree trunk is not smooth so the friction coefficient in fact varies and the wheel may be slipped. The climbing speed of each robot leg cannot always be kept the same. As a result, the platform is constantly tilted in different directions. The stepper therefore must be engaged to adjust the normal forces on the three wheels to keep the platform horizontal. The platform tiling control and robot wheel slippage control will be illustrated in the robot control section.

4.3.3 Active and Passive Anti-Falling Mechanism

This novel tree climbing robot design has a special characteristic: the anti-falling mechanism, both active and passive. As previously discussed， the passive anti-falling mechanism uses only friction forces and the gravity force of the robot to maintain a hold on the trunk like Kawasaki’s and Faurouxs’ designs in static or sudden power cut off situation. The primary point of achieving this feature is to let the center of the mass of the robot offset from the center of the tree. The lateral and top views of passive anti-falling mechanism are illustrated in Figure 4.25.

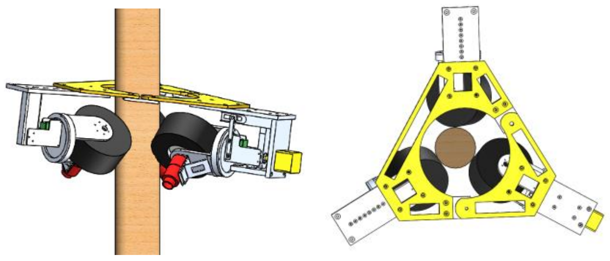


Figure 4.25 Passive Anti-Falling Mechanism: Lateral View and Top View

During the climbing process, due to the diverse and complicated tree surface, the robot may slip or fall in extreme conditions. Therefore, an active anti-falling mechanism is proposed to overcome this problem. As shown in Figure 4.16, when the normal forces increase, with the unchanging friction coefficient μ, the total friction force of the robot increases correspondingly. This process is executed by the stepper and screw-nut unit. If the stepper motor moves toward to the trunk, the normal forces on the wheel-trunk contact areas increase. So, as the related friction forces. With such an active anti-falling mechanism, it guarantees the robot can climb up the trunk safely and steadily without slip or falling.

The entire implementation of the active anti-falling mechanism is illustrated by the following flowchart. During the climbing process, the active anti-falling mechanism plays the leading role while the passive anti-falling mechanism acts as a complement. If the robot system loses its power, the active anti-falling mechanism is mal-functional, and the passive anti-falling mechanism takes over the control to hang the robot on the tree. This doubles the guarantee to resist robot falling. Compare to passive anti-falling mechanism, the main drawback of active anti-falling mechanism is that it consumes the power and requires a dynamic control system.

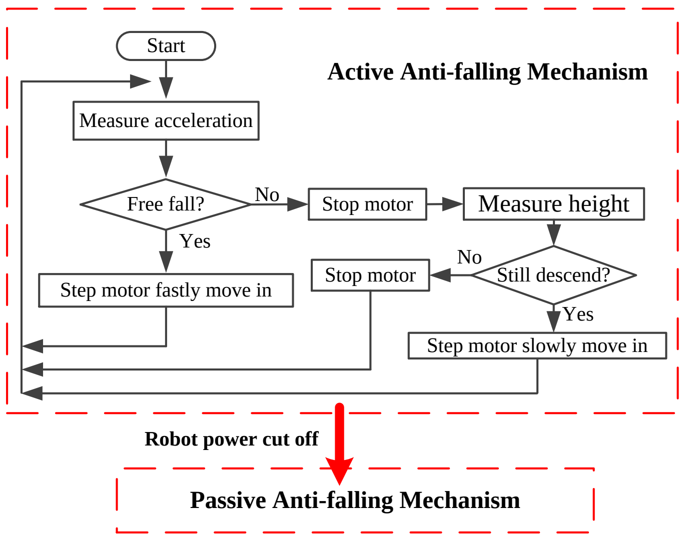


Figure 4.26 Anti-falling Mechanism Flowchart

4.3.4 Simulation and Experiment of Optimized Tree Pruning Robot

The assembly property information of optimized pruning robot is summarized in table 4.3. The mass is 6.42 Kg while the conceptual model is 8.76 Kg.

Table 4.3 Specifications of pruning robot

Parameters	**Values**
Platform Parts Mass Density	2700 kg/m^3
Wheel Mass Density	960 kg/m^3
Robot Mass	6419.74 g
Volume	3079209.1 cubic mm^3
Surface area	1081348.83 mm^2

As discussed previously, during the vertical climbing process the wheel tractive force overcomes the rolling resistance, robot gravity, acceleration resistance, motor bearing heat losses, etc. To confirm the previous analysis and the behavior of the climbing robot and help to select the correct motors, a dynamic simulation of the proposed robot climbing system was carried out. The simulation and analysis of the climbing process are also conducted in MATLAB SimMechanics like the conceptual model. Figure 4.27 shows the four views of the climbing robot in SimMechanics.

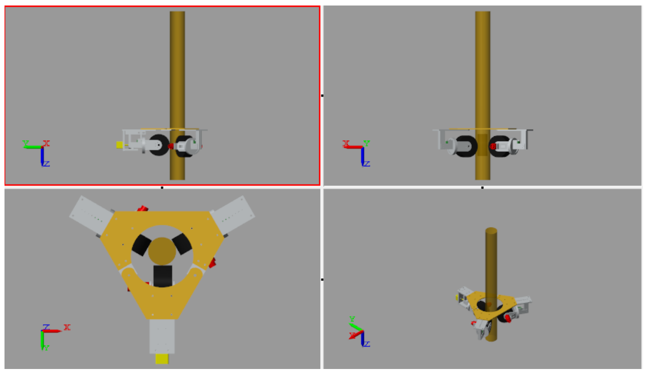


Figure 4.27 Four Views of Pruning Robot in Simmechanics

In the dynamic simulation, the inverse kinematic method was used. The wheel rotation angle and rotation speed were set to 4 pi rad and 12 rpm respectively. To make the simulation more realistic, the slippage and wheel deformation were taken into consideration. Figure 4.28 showcases the results of required DC motor torque which is around 1.15 N-m. The corresponding robot vertical climbing speed is shown in Figure 4.29. Due to the wheel slippage and deformation, the average robot vertical climbing speed is 62.9 mm/s.

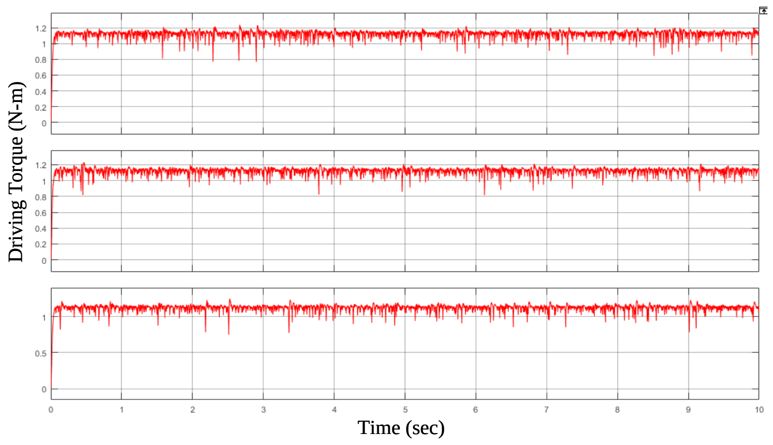


Figure 4.28 DC Motor Driving Torque

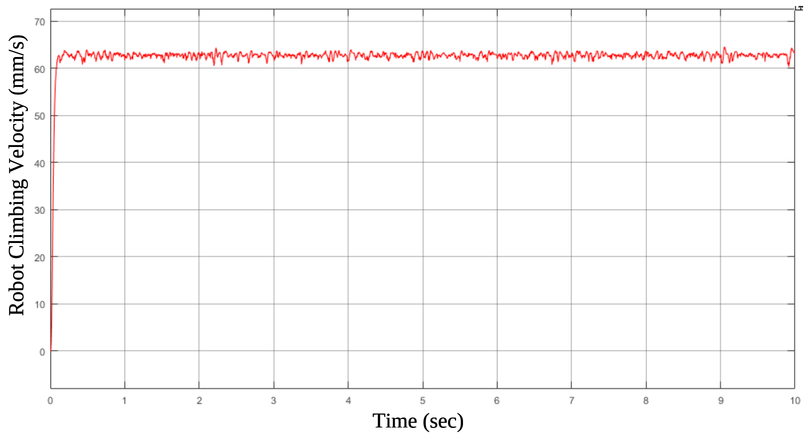


Figure 4.29 Robot Climbing Velocity

Based on the simulated DC torque and previously dynamic analysis, the DC motor selected in this design is a powerful 12V brushed DC motor with an integrated quadrature encoder which can offer maximum 16 kg-cm torque.

The friction coefficient between rubber and wood is in the range of 0.75 to 1 [38]. Based on the robot gravity and previous force analysis, the type of stepper and servomotor was selected. The servomotor used in this design can produce 17 kg-cm torque at 6V. The stepper has 5 kg-cm holding torque. The dimension of the robot's rubber wheel is 120×60 mm. The first testing climbing robot is then built and shown in Figure 4.30 and 4.31. Figure 4.30 illustrates the structure of one robot leg while Figure 4.31 shows that the robot keeps stationary on the test rod using the passive anti-falling mechanism. The length of the test rod is 1 meter and the diameter is 122.5 millimeters. The total mass of this prototype is 6.8 kg.

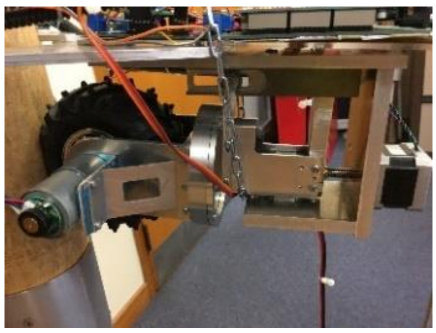


Figure 4.30 Tree Climbing Robot Leg

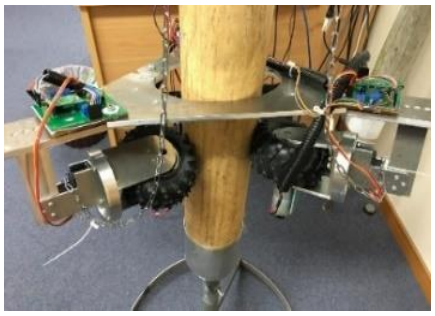


Figure 4.31 Tree Climbing Robot Physical Test Model

This chapter illustrates the mechanical design and optimization of tree pruning robot according to the requirements and functions of tree pruning. These CAD models were simulated and verified in the SolidWorks and MATLAB SimMechanics. Base on the outcome of simulation, the robot motor types are determined. The following chapter will discuss the robot hardware and software design.

Chapter 5 Tree Pruning Robot Control System Design

This chapter describes the tree pruning robot control methodologies and system design which utilized the modular design method. For the entire control system, it focuses on the design methodologies, intelligent control and PCB board design. The control system consists of hardware and software design which is illustrated in Figure 5.1. In terms of hardware design, it covers power module, MCU module, motor driving module, communication module, etc. while the software aims at the development of control algorithms such as filtering techniques, climbing robot tilting measurement, fuzzy control, and wheel slip control. The software development is specifically discussed later chapters with Chapter 6 focuses on filtering and tilting measurement, Chapter 7 and 8 on fuzzy and neural network control respectively.

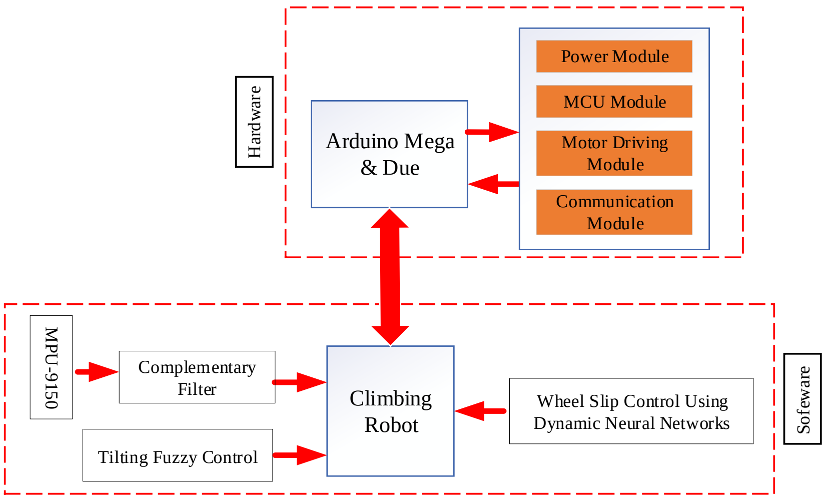


Figure 5.1 Robot control system diagram

5.1 Tree Pruning Robot Control System Schematic Design

The essential part of the control system is the design of tree pruning robot PCB. It plays a key role in the climbing robot control system since the entire system control is via the PCB board. Furthermore, a professional PCB layout enables the control system more efficient, concise and stable. While designing a professional PCB board is not an easy work, there are many factors and aspects needed to be taken into consideration, such as heat dissipation, electromagnetic noise, voltage regulation, current adjustment, etc.

The control system PCB acts as a bridge between the robot electronics and mechanics. It uses the MCU to control the robot mechanical actuators to finish certain tasks. And the PCB board is also a carrier for the hardware and software of the robot control system. In terms of hardware, the robot motor driving system, tilting measurement system, robot communication system and power conversion system are all designed and placed on the PCB board. In the design of software, the DC motor, servomotor and stepper control algorithm, IMU data fusion algorithm, tilting control algorithm and robot wheel slippage control are all applied in the MCU. Therefore, the design of tree pruning robot PCB is significant to the robot control system, and also to the entire robot system.

In order to design a professional climbing robot PCB, the first thing is to design the schematic. PCB layout is the resulting design from taking a schematic with specific components and determining how they will physically be laid out on a printed circuit board. To produce a PCB layout, the connections of components, components sizes (footprints), and a myriad of other properties (such as current, frequencies, emissions, reflections, high voltage gaps, safety considerations, manufacturing tolerances, etc.) should be studied and researched in advance. Such work is mainly done in the schematic. It is based on the principles of the PCB design.

A schematic shows the connection in a circuit in a way that is clear and standardized. It is a way of communicating to other engineers exactly what components are involved in a circuit as well as how they are connected. A good schematic shows component names and values, provide labels for sections or components to help communicate the intended purpose. In the climbing robot schematic design, the entire control system schematic can be divided into seven sections: Arduino Mega2560[3] Control Module, Arduino Due[4] Module, Communication Module, DC Motor Module, Servomotor Module, Stepper Module, and Power Module.

[3] Atmega2560 datasheet, https://cdn.sparkfun.com/datasheets/Components/General%20IC/2549S.pdf.

[4] Atmel SAM3X8E ARM Cortex-M3 datasheet, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf.

The entire climbing robot schematic and PCB are designed using Altium Designer[5]. It is an electronic design automation software package for printed circuit board, FPGA and embedded software design. For the climbing robot PCB design, the first thing needed to do is to design the schematic library and PCB library according to the electronic components size and footprint.

5.1.1 Power Module

The first part of control system schematic is power module design. In the climbing robot control system, there are five different voltage levels: 3.3V, 5V, 6V, 9V and 12V. The 12V is the battery voltage level which supplies power for the entire robot system. The 6V and 9V voltage levels are converted from the 12V voltage level, while the 5V and 3.3V voltage level are from the Arduino Mega and Due board respectively.

The 12V voltage level is used for the DC motor and stepper, while the 6V is the working voltage of three servo motors. The power supplied for Arduino Mega and Due boards are from 9V voltage level. The DC motor and stepper driver control logic voltage level, the encoder power and main current sensor power are all 5V, while the IMU power, the XBee power and logic level converter are 3.3V.

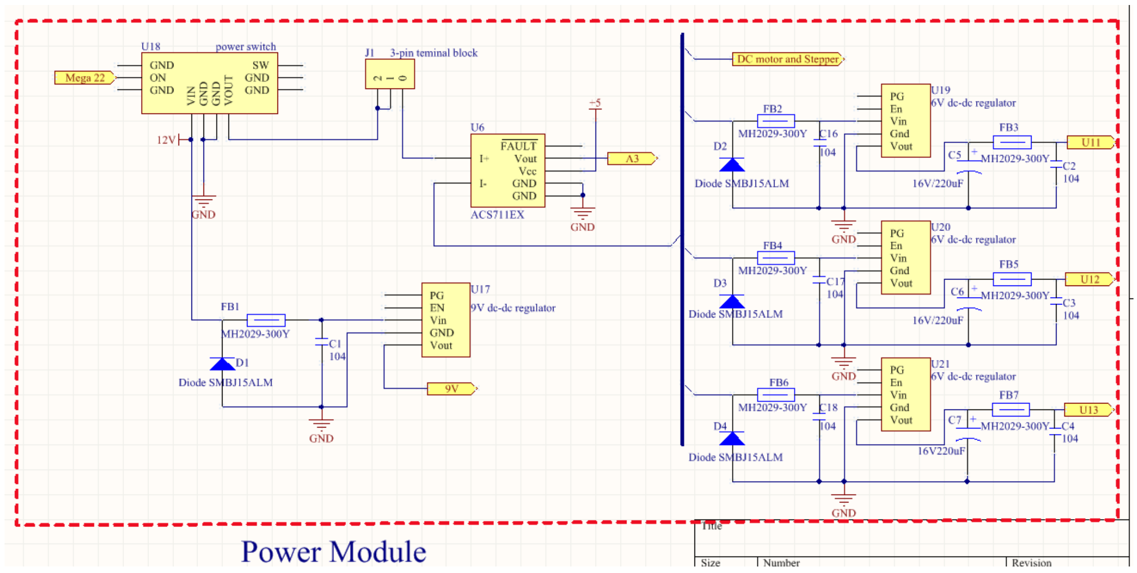


Figure 5.2 Power module schematic

[5] Altium Designer ver.13, http://www.altium.com/documentation/18.0/display/ADES/Altium+Designer+Documentation.

In order to hold high current for the entire robot system, the power switch[6] used is a big MOSFET slide switch with reverse voltage protection which can supply 6A continuous current at 55℃, 16A continuous current at 150℃. The current sensor used is Allegro's ACS711EX[7] (-15.5A ~ +15.5A) which is a hall effect-based linear current sensor with overcurrent fault output. The step-down voltage regulators used for 6V and 9V are Pololu's D24V22F6[8] and D24V22F9[9] respectively.

The robot power switch, current sensor, step-down voltage regulators and their typical efficiency, dropout voltage graphs are displayed in Figure 5.3, 5.4, 5.5, and 5.6.

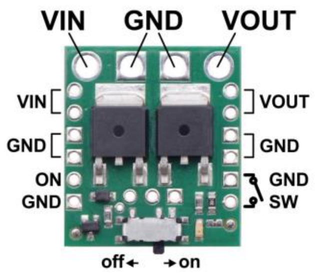


Figure 5.3 MOSFET Slide Power Switch

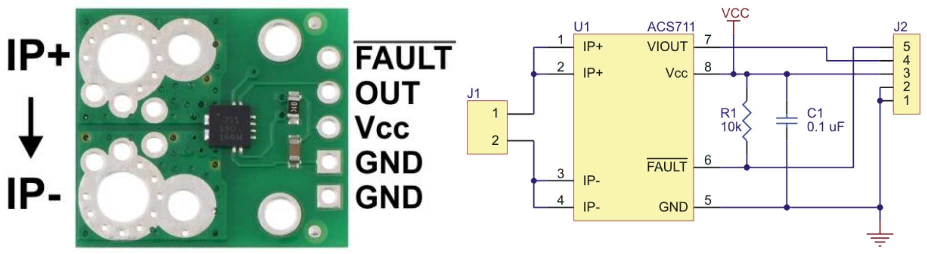


Figure 5.4 Current Sensor

[6] Big MOSFET Slide Switch with Reverse Voltage Protection, HP, https://www.pololu.com/product/2815.

[7] ACS711EX Current Sensor Carrier -15.5A to +15.5A, https://www.pololu.com/product/2452.

[8] Pololu 6V, 2.5A Step-Down Voltage Regulator D24V22F6, https://www.pololu.com/product/2859.

[9] Pololu 9V, 2.3A Step-Down Voltage Regulator D24V22F9, https://www.pololu.com/product/2861.

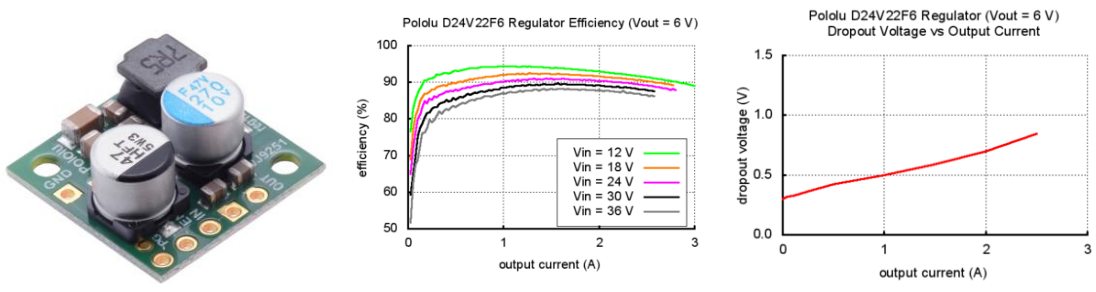


Figure 5.5 6V DC-DC Converter

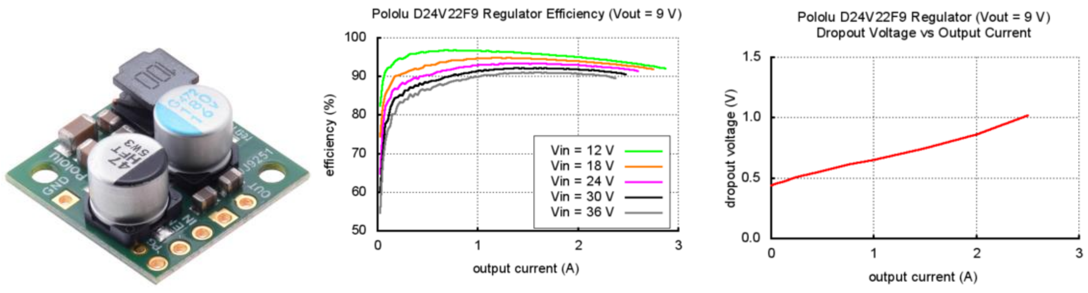


Figure 5.6 9V DC-DC Converter

The 9V DC-DC converter can still get power supply even the main power switch is turned off. Because the 9V voltage is used for two MCUs, the converter power supply is directly from the battery. That guarantees the control circuit power and actuator circuit power are separated so that the control system still can control the robot when the motor driving system power is turned off.

5.1.2 MCU Module (Arduino Mega and Due)

The climbing robot uses Arduino Mega 2560 and Arduino Due as the MCU. Arduino Mega is responsible for dealing with the control algorithm of motors (3 DC motors, 3 Servomotors, 1 stepper), monitoring the system current and voltage, transmitting control system information and receiving commands from Arduino Due through serial ports. And the Mega also supplies 5V voltage for the robot control system.

The Arduino Due plays a role as information processing and control algorithm operating. It collects the tilting information from IMU module and operates the data fusion algorithm to obtain the useful and precision tilting angles. Then according to these tilting (pitch, roll and yaw) angles and information from Mega, the Due calculates and gives commands to Mega to control the entire robot climbing speed and morphology. The tree pruning robot tilting control algorithm and wheel slippage control algorithm are all executing on Arduino Due. Besides, Due can also communicate with the PC using the XBee module to realize remote control. The system voltage level 3.3V is from Arduino Due module.

One thing that needs to be emphasized here is the voltage compatible. The voltage level of Arduino Mega system is 5V while the Due is 3.3V, so the logic level converter must be used to solve this issue.

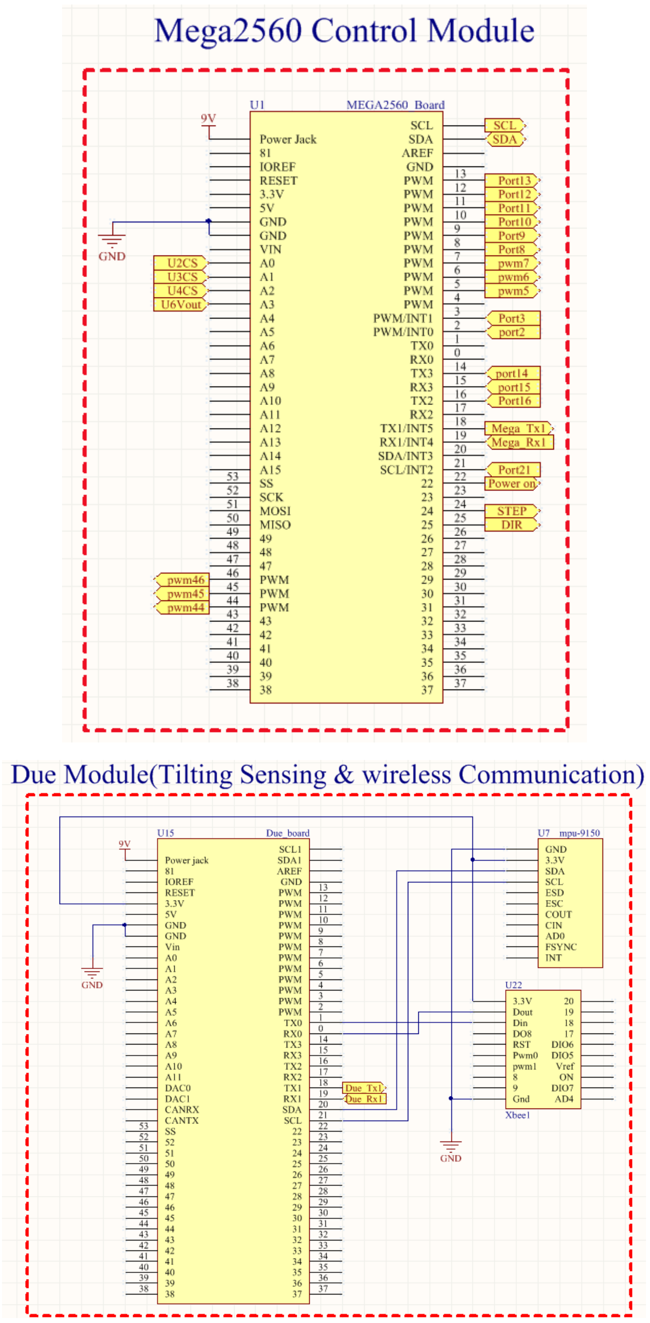
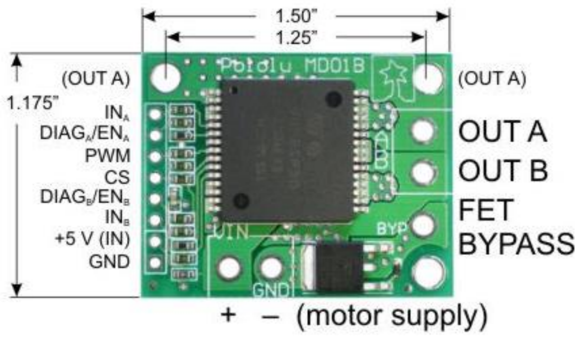


Figure 5.7 Tree pruning robot schematic design of Arduino Mega 2560 and Arduino Due

5.1.3 Motor Driving Module

This module contains three parts: DC motor, Servomotor and Stepper. The DC motor driver is ST's VNH2SP30[10] motor driver IC which operates from 5.5V to 16V and can deliver a continuous 14A (30A peak). It works with 5V logic levels and supports ultrasonic (up to 20 KHz) PWM. It features current sense feedback (an analog voltage proportional to the motor current) along with built-in protection against reverse-voltage, over-voltage, under-voltage, over-temperature, and over-current. This DC motor driver is a fully integrated H-bridge that can be used for bidirectional speed control of a single brushed DC motor. The breakout board and schematic of DC motor driver are shown in figure 5.8.



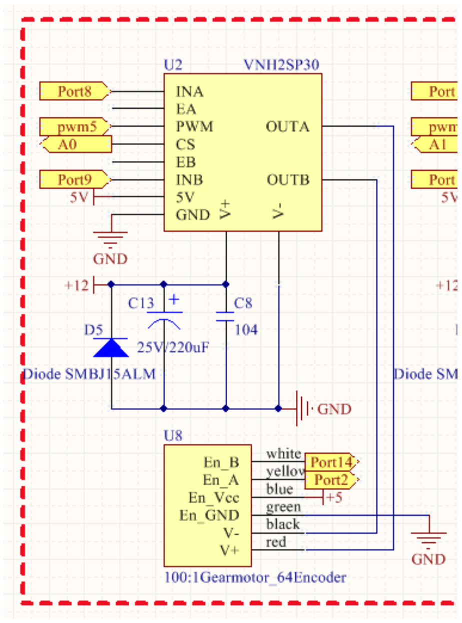


Figure 5.8 Breakout board and schematic design of DC motor driver – VNH2SP30

[10] VNH2SP30 Motor Driver Carrier MD01B, https://www.pololu.com/product/706.

The DC motor used in this robot is powerful 12V brushed gear motor (figure 5.9) with a 102.083:1 metal gearbox and an integrated quadrature encoder that provides a resolution of 64 counts per revolution of the motor shaft, which corresponds to 6533 counts per revolution of the gearbox's output shaft[11].

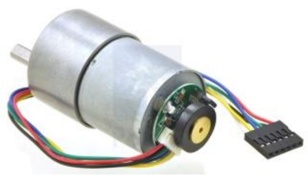


Figure 5.9 12V brushed gear DC Motor

The servo motor used in this robot is 1501MG[12] from Power HD which is a metal-geared analog servo that delivers extra-high torque. This servo weighs 60 grams and the stall torque is 17 Kg-cm at 6V. The stall current is 2.5A at 6V operating voltage. The limit angle is 180°±10°. Its schematic is shown in figure 5.10.

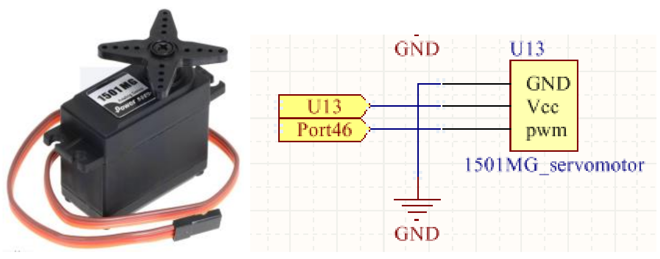


Figure 5.10 Servomotor and schematic

The stepper[13] used in the robot is a NEMA 17-size hybrid bipolar stepping motor (figure 5.11) which has an integrated 18 cm threaded rod as its output shaft, turning it into a linear actuator capable of precision open-loop positioning. The included traveling nut has four mounting holes and moves 40 μm per full step. Finer resolution can be achieved with micro-stepping. The stepper motor has a 1.8-degree step angle (200 steps/revolution) and each phase draws 1.7A at 2.8V, allowing for a holding torque of 3.7 Kg-cm.

[11] 100:1 Metal Gearmotor 37DX57L mm with 64 CPR Encoder, https://www.pololu.com/product/1446.

[12] Power HD High-Torque Servo 1501MG, https://www.pololu.com/product/1057.

[13] Stepper Motor with 28cm Lead Screw, https://www.pololu.com/product/2268.

Figure 5.11 NEMA 17-size stepper motor

The stepper motor driver used is TI's DRV8825[14]. This micro-stepping bipolar stepper motor driver features adjustable current limiting, over-current protection, over-temperature protection and six micro-step resolutions. It operates from 8.2V to 45V and can deliver up to approximately 1.5A per phase without a heat sink or forced air flow (rated for up to 2.2A per coil with sufficient additional cooling). The breakout board and schematic design of stepper motor are illustrated in Figure 5.12.

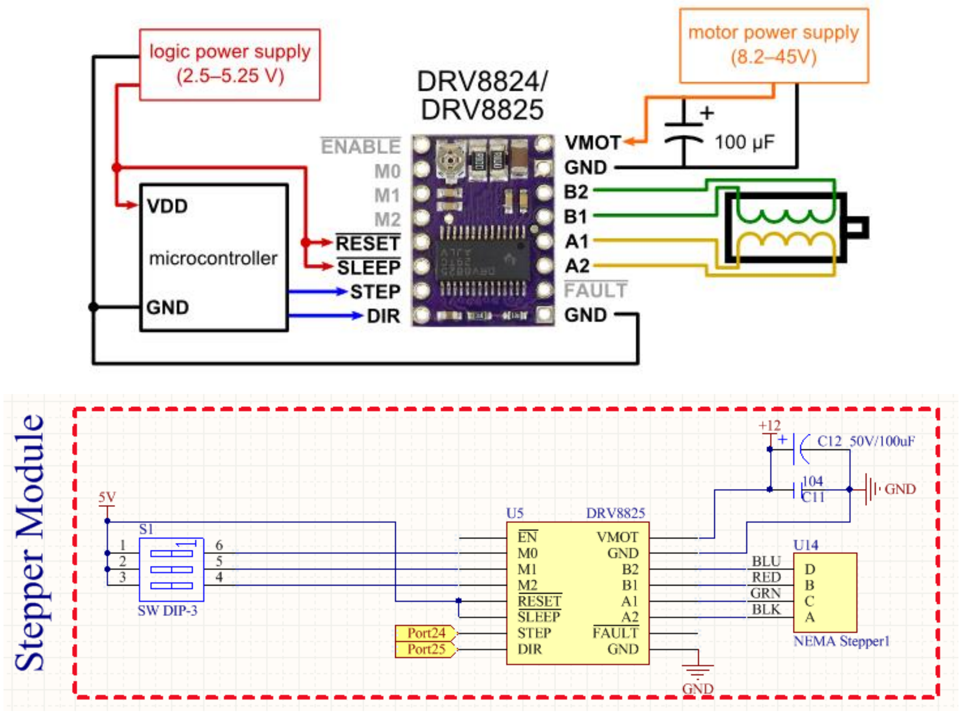


Figure 5.12 Breakout board and schematic design of stepper motor driver – DRV8825

These three types of motors make up the whole climbing robot driving system.

[14] DRV8825 Stepper Motor Driver Carrier, https://www.pololu.com/product/2133.

5.1.4 IMU Module

The IMU module uses MPU-9150[15] to acquire precision tilting angles for the tree pruning robot tilting control and wheel slippage control. The 9-DOF MPU-9150 is the world's first 9-axis motion tracking MEMS device designed for the low power, low cost, and high-performance requirements of customer electronics equipment including smartphones, tablets and wearable sensors.

The MPU-9150 breakout board provides I^{2}C pull-up resistors so that it is easy and simple to communicate with Arduino Due using SCL and SDA ports. What's more, the power of the MPU-9150 is supplied by Arduino Due directly since they have the same voltage level – 3.3V.

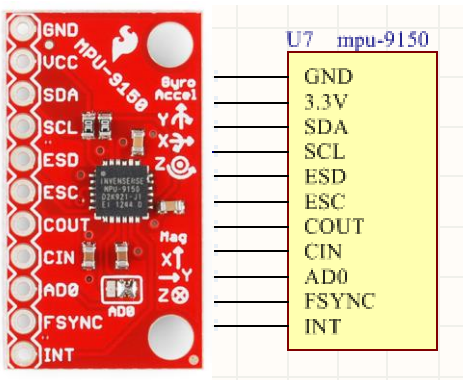


Figure 5.13 Breakout board and schematic design of MPU-9150

Main features of MPU-9150:

- Tri-axis angular rate gyroscope with a sensitivity up to 131 LSB/dps and a full-scale range of ±250, ±500, ±1000 and ±2000 dps
- Tri-axis accelerometer with a programmable full-scale range of ±2g, ±4g, ±8g and ±16g
- Tri-axis compass with a full-scale range of ±1200 uT
- 400KHz fast mode I^{2}C serial host interface
- 10000g shock tolerant
- I^{2}C pullup resistors populated on board

[15] "MPU-9150 Register Map and Descriptions Revision 4.0," InvenSense Ltd. 12, 2012. http://www.invensense.com.

- Full chip idle mode supply current: 8 uA
- All pins broken out to standard 0.1’’ spaced headers

5.1.5 Communication Module

The robot communication module consists of two parts: serial communication between Arduino Mega 2560 and Arduino Due; wireless communication between Arduino Due and PC. In terms of serial communication, the Arduino Mega can receive commands from Arduino Due to control the driving system and transmit robot system data to Due through serial communication ports. This is internal communication operated automatically between these two MCUs. For wireless communication, the robot operator can directly give commands to control the robot by PC using the XBee module[16]. On the other hand, the user can also monitor the robot climbing states and parameters through the PC using XBee. One thing should be noticed is the voltage compatible when using serial communication between Arduino Mega (5V) and Arduino Due (3.3V). Figure 5.14 and figure 5.15 display these two communication approaches.

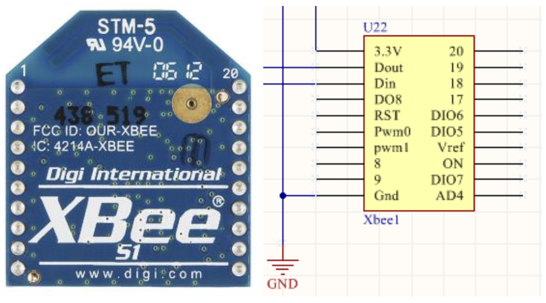


Figure 5.14 Breakout board and schematic design of wireless communication module – XBee

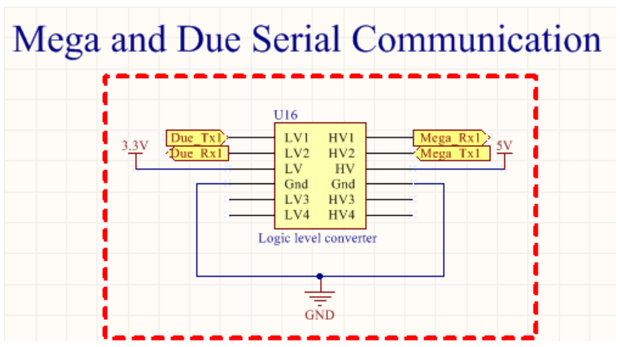


Figure 5.15 Serial communication between Arduino Mega and Arduino Due

[16] XBee Zigbee RF Module-PCB Antenna, https://nicegear.nz/product/xbee-zigbee-rf-module-chip-antenna.

5.2 Tree Pruning Robot Control System PCB Design

There are some issues needed to be taken into consideration when designing the robot control system PCB, like radiated interference, heat dissipation, transient voltage and current, etc. These issues all belong to the EMI/EMC range.

5.2.1 Radiated Interference

In terms of radiated interference, in this tree pruning robot PCB design the buck converter noise and motor noise are the two main noise sources. There are many ways to reduce the level of radiated interference, especially during the initial design of the circuit board. These techniques include proper routing of tracks, proper use of ground planes, power supply impedance matching, and reducing logic frequency to a minimum. While even with the most diligent employment of good EMI/EMC circuit design practices, not all interference or compatibility issues can be eliminated. At this point, additional components can be added, allowing the circuit to comply with design and regulation limits for EMI/EMC [39].

1) Buck Converter Interference

In the climbing robot power module, there are four step-down DC-DC converters to supply the 9v and 6V voltage level for MCU and servomotors respectively. These fast transient, compact, power management ICs offer significantly higher efficiency than comparable linear alternatives. Moreover, in order to allow designers to take advantage of smaller external inductors and ceramic capacitors, the power semiconductor designers have migrated to step-down converters using increasingly higher switching frequencies. The move to step-down converters using higher switching frequencies has generated another problem – input noise. If it isn’t filtered, DC-DC converter input ripple and noise can reach levels high enough to interfere with other devices powered from the same source.

Input noise in a step-down DC-DC converter has two components. The first occurs at the fundamental switching frequency commonly referred to as ripple. The second noise component is ringing [40]. “Ringing” is a common term referring to the undesired oscillation that occurs when a power semiconductor switch turns on or off in the presence of parasitic inductance and capacitance. Energy stored in the parasitic junction capacitance of the switch is released during the switching transition and rings with parasitic inductance coming from the stray fields of discrete power inductors and the wiring inductance of the PCB traces, component leads, connectors, etc. In real circuits, parasitic are always present, and hence all switching converters produce at least some ringing. This electromagnetic interference (EMI) is typically in the range of 50 to 200 MHz, and at these frequencies PCB traces and the input and output leads act as unwanted antennas, resulting in both conducted and radiated noise [41].

To solve this issue, one effective approach is to use the ferrite beads especially when the impedance of the circuit board trace is insufficient as a series element for the low pass noise filter. Designers can increase the series impedance with a small surface mount ferrite bead to improve noise rejection – especially the high-frequency noise.

2) *What is ferrite bead and how to select the Proper ferrite bead*

A ferrite bead is a passive electric component that suppresses high-frequency noise in electronic circuits. It employs high-frequency current dissipation in a ferrite ceramic to build high-frequency noise suppression devices[17].

The simplest form of a ferrite bead is a conducting wire inserted through a hollow piece of ceramic material known as ferrite. The electromagnetic properties of ferrite allow the material to influence the current flowing through the conductor. The precise nature of this influence depends on the type of ferrite (e.g., manganese-zinc vs. nickel-zinc), and the properties of a particular ferrite material can be further refined via the manufacturing process. In many surface-mount ferrite beads, the conductor is formed into a coil structure, with individual windings layered between ferrite sheets. Thus, the electrical characteristics depend also on the details of the winding construction.

Ferrite beads can be divided into two general categories: high-Q (resonant) beads and low-Q (non-resonant) beads. High-Q beads are intended for applications that require high levels of resonance, such as oscillators and specialized filters. In the context of

[17] Ferrite bead, Wikipedia, https://en.wikipedia.org/wiki/Ferrite_bead.

power-supply filtering, resonance needs to be minimized. Hence, in the tree pruning robot PCB design, the low-Q beads are used.

In order to better understand the ferrite beads, a first-order equivalent circuit is showing below.

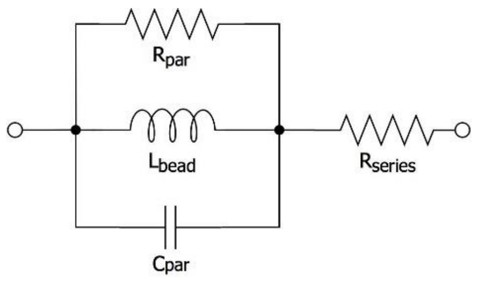


Figure 5.16 Ferrite Bead Equivalent Circuit

The inductor is placed in the center as a reminder that the predominant response of a ferrite bead is inductive, i.e., impedance increases with frequency. However, at some point (generally somewhere between 30 and 500 MHz), the parallel capacitance begins to dominate the inductance, and impedance then decreases with frequency. The relatively small parallel resistance reduces the resonance associated with the capacitor and inductor, such that the impedance levels off at the transition point instead of peaking in typical high-Q fashion. This response is evident in the following plot showing the measured impedance characteristics of a standard SMD ferrite bead [42].

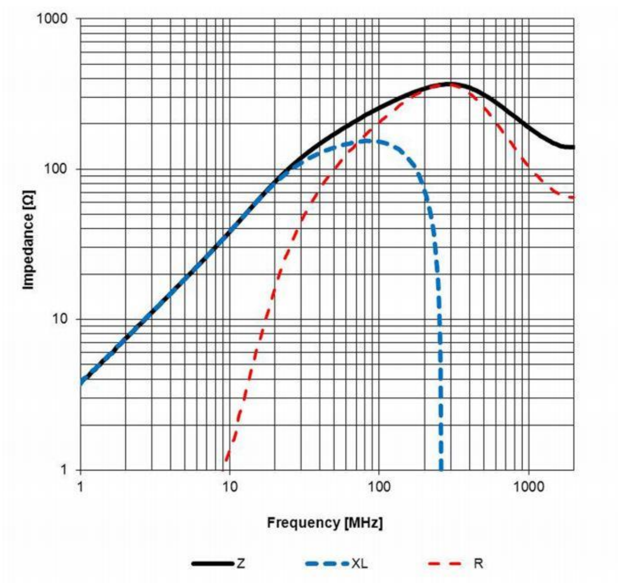


Figure 5.17 Ferrite Bead Impedance VS. Frequency Plot

In figure 5.17, the black line indicates the overall impedance, which starts at R_{series}, also known as the bead's DC resistance. It then increases linearly during the inductive frequency range and levels off at 300 MHz. After that, it begins to decrease before leveling off at 1.1 GHz.

The red and blue dotted lines indicate that the overall impedance is the result of two distinct elements, inductive reactance (XL) and frequency-dependent resistance (R). This brings up an important point: the equivalent circuit given above (figure 5.16) is designed to replicate the frequency response of the bead – it does not convey the bead's internal structure. The equivalent model is helpful for understanding how a ferrite bead's impedance changes with frequency and performing simulations, but primarily it is the ferrite material itself that determines the component's impedance properties. This is important to understand because the equivalent circuit might distract you from one of the defining characteristics of ferrite beads: they actually dissipate high-frequency energy.

The ideal inductors and capacitors do not dissipate any energy. They merely store energy, either in a magnetic field (inductors) or an electric field (capacitors). A resistor, on the other hand, takes energy out of the circuit and dissipates it as heat. Ferrite beads, unlike inductors, are intentionally resistive at high frequencies. This is why the above plot has the red dotted line labelled "R" – from about 100 MHz to 1 GHz, the bead exhibits significant resistive impedance, not reactive impedance. Actually, some ferrite beads and ferrite-core inductors are almost identical in construction, except that the ferrite bead uses a more "lossy" ferrite material because the manufacturer wants the bead to dissipate rather than store high-frequency energy.

The key to maximizing the noise-suppression benefits of a ferrite bead is to ensure that the targeted noise frequencies fall within the bead's resistive band – i.e., that portion of the frequency response where the resistive impedance dominates the reactive impedance. This is a fundamental aspect of maximizing the ferrite bead's ability to suppress noise, but there are other specifications that needed to keep in mind – DC resistance and rated current.

Unlike bypass capacitors, ferrite beads are used in series with the power line, which means that any DC current flowing through the bead creates a voltage drop proportional to the DC resistance.

A ferrite bead's DC resistance – much less than an ohm for typical surface mount parts – is rarely an issue in the range of low-power ICs. However, in the case, e.g., an unusually high-power device could draw enough current to cause a problem.

Rated current is not as straightforward as it seems. Indeed, if the steady state current through the bead is higher than the rated current, damage may occur. But there are two nuances that need to be aware of. First, rated current is not constant over temperature. Second, DC currents well below the rated maximum can degrade the bead's performance because the ferrite material becomes saturated. Ferrite saturation reduces the bead's peak impedance and shifts the impedance curve toward higher frequencies. To reduce the effects of core saturation, the bead's rated current should be at least 50% higher than expected maximum current. The SMD ferrite bead (MH2029-300Y) and its schematic design are shown in Figure 5.18.

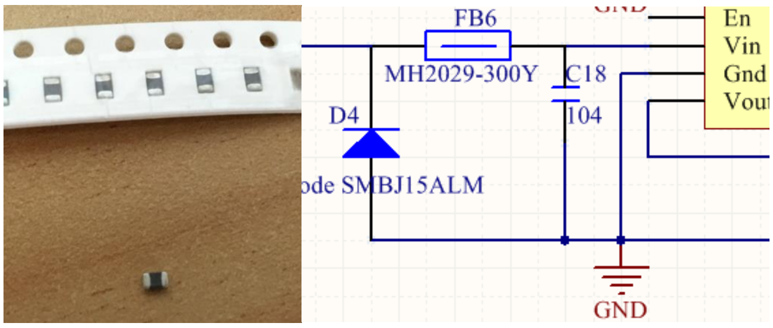


Figure 5.18 Ferrite Bead and its schematic design

3) *Motor Electrical Noise*

One major drawback to working with motors is the large amounts of electrical noise they produce. This noise can interfere with sensors and can even impair the microcontroller by causing voltage dips on regulated power line. Large enough voltage dips can corrupt the data in microcontroller registers or cause the microcontroller to reset.

The main source of motor noise for brushed DC motor is the commutator brushes, which can bounce as the motor shaft rotates. This bouncing, when coupled with the inductance of the motor coils and motor leads, can lead to a lot of noise on the power line and can even induce noise in nearby lines.

For stepper motor, the noise is mainly from the current chopping. Current chopping is the preferred method of driving steppers nowadays, because this method occupies less space, costs less and generates less heat comparing to using huge resistor. However, the current chopping means utilizing PWM signal to turn the H-bridge on and off continuously. During this process, the noise occurs.

These motor noise (electrical noise) can be classified into two types.

Table 5.1 motor electrical noise

Type	**Unit**	**Definition**	**Typical Frequency**
Line Noise (conductive noise)	dBuV	Noise that travels through power cables and connection cables	0.15 – 30MHz
Radiation Noise (radio wave)	dBuV/m	Noise that is radiated from the source to the air and causes interference in TV and a radio.	30 – 1000MHz

There are several approaches to reduce the effects of motor noise on the robot control system PCB:

- Using the bypass capacitors across power and ground of motor (figure 5.19): a $220\mu F$ capacitor and a $0.1\mu F$ capacitor. The larger capacitor smooths out low-frequency variations in the supply voltage, while the smaller capacitor more effectively filters out high-frequency noise on the power line.
- Route motor and power wires away from signal lines.
- For stepper motor, increase switching frequency and decrease stepper current.

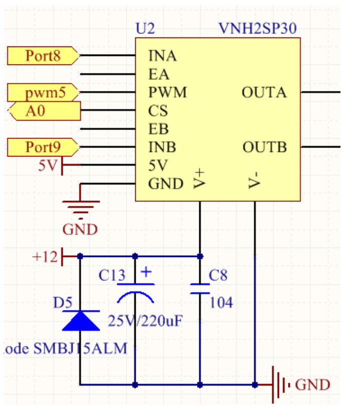


Figure 5.19 Bypass Capacitors for Reducing the Motor Noise

5.2.2 Heat Dissipation

Another issue is thermal dissipation. Since the robot operates 3 DC motors, 3 Servomotors and 1 stepper, the current that runs through the PCB can be up to around 14A in the worst situation. Such a high current renders the PCB temperature rising sharply and even lead to the electronic components malfunctioned.

There are several ways to cope with this problem. One is in the PCB track designing which means the power track should be widened. For example, the 12V track is 75 mils. Another way is to use more copper during the PCB manufacturing. There is a relationship among PCB copper thickness, current magnitude and track width which is displayed in table 5.2.

Table 5.2 Relationship among Copper Thickness, Track Width and Current Magnitude for PCB design

Copper Thickness/35 um		**Copper Thickness/50 um**		**Copper Thickness/75 um**	
Current (A)	Track Thickness (mm)	Current (A)	Track Thickness (mm)	Current (A)	Track Thickness (mm)
4.5	2.5	5.1	2.5	6	2.5
4	2	4.3	2.5	5.1	2
3.2	1.5	3.5	1.5	4.2	1.5
2.7	1.2	3	1.2	3.6	1.2
3.2	1	2.6	1	2.3	1
2	0.8	2.4	0.8	2.8	0.8
1.6	0.6	1.9	0.6	2.3	0.6
1.35	0.5	1.7	0.5	2	0.5
1.1	0.4	1.35	0.4	1.7	0.4
0.8	0.3	1.1	0.3	1.3	0.3
0.55	0.2	0.7	0.2	0.9	0.2
0.2	0.15	0.5	0.15	0.7	0.15

5.2.3 Transient Voltage and Current

Another issue needed to be solved is the transient voltage and current when designing the PCB. When the robot system power turns on, inevitably, there are some voltage and current pulses for the components and ICs on the circuit board. Without enough protection, these sensitive components or ICs will be damaged.

One solution is to use the transient voltage suppressor diode (TVS Diode). The TVS diode is an electronic component used to protect sensitive electronics from voltage spikes induced on connected wires. It operates by shunting excess current when the induced voltage exceeds the avalanche breakdown potential. It is a clamping device, suppressing all over-voltages above its breakdown voltage. It automatically resets when the overvoltage goes away.

When using TVS, the most important parameters are the rated working peak voltage or rated standoff voltage (V_R), the peak pulse power dissipation (P_{pp}), peak impulse current (I_{pp}), and clamping voltage (V_c). The first step of selecting a TVS is to determine what the highest continuous peak operating voltage will be at the point of intended protection in the circuit. This highest operating voltage determines the rated standoff voltage of the TVS component. The clamping voltage is typically 60% higher than V_R. The second thing is to select the peak pulse power dissipation. For board level designs, 400W to 600W at 10/1000 µs or 300W to 500W at 8/20 µs are most often used [43]. The TVS diode (SMBJ15ALM) used in this robot is shown in Figure 5.20.

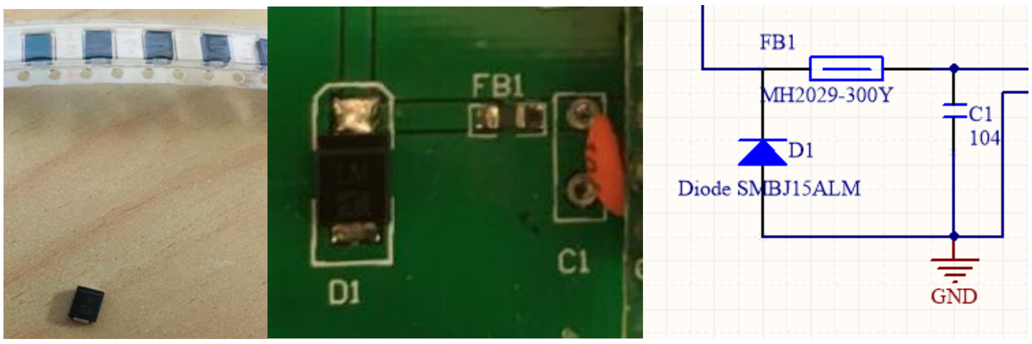


Figure 5.20 Transient Voltage Suppressor Diode

5.3 Circuit Design Tips to Reduce EMI/EMC Problems

As discussed above, there are several areas where good circuit design practices are critical to the reduction or elimination of EMI/EMC problems. The PCB layout is important – not only in the design but also the choice of components – directly affects the degree of EMI/EMC interference. Another area of concern is the circuit design of power supply.

5.3.1 PCB Layout Design Tips

- Avoid slit apertures in PCB layout, particularly in ground planes or near current paths.
- Areas of high impedance give rise to high EMI, so use wide tracks for power lines on the trace sides.
- Keep HF and RF tracks as short as possible and lay out the HF tracks first.
- Avoid track stubs, as they cause reflections and harmonics.
- When having separate power planes, keep them over a common ground to reduce system noise and power coupling.
- If possible, make tracks run orthogonally between adjacent layers.
- Do not loop tracks, even between layers, as this can form a receiving or radiating antenna.
- Do not leave floating conductor areas, as they act as EMI radiators; if possible, connect them to ground plane [44].

5.3.2 Power Supply Considerations

- Eliminate loops in the supply lines.
- Decouple supply lines at local boundaries.
- Place high-speed circuits close to power supply unit and slowest sections furthest away to reduce power plane transients.
- Isolate individual systems where possible (especially analog and digital systems) on both power supply and signal lines.

5.3.3 Component Considerations

- Locate biasing and pull down/up components close to driver/bias points.
- Minimize output drive from clock circuits.
- Decouple close to chip supply lines, to reduce component noise and power line transients.
- Use low impedance capacitors for decoupling and bypassing (ceramic multilayer capacitors).

The whole tree pruning robot PCB and schematic is displayed in Appendix D.

Chapter 6 Tree Pruning Robot Tilting Measurement

The major part of a tree pruning robot is the climbing mechanism which determines whether the robot can climb up the trunk safely and stably. In reality, during the robot climbing process, each of the three DC motors may suffer slippage due to the complicated trunk surface. Therefore, even if the three wheels keep the same speed during the climbing procedure, the robot platform could dynamically tilt at a random direction. In extreme situation, if the wheel suffers constant slippage, the robot may fall to ground or get jammed on the trunk. Hence, the tree pruning robot tilting control strategy must be developed.

Before the study and development of tilting control algorithm, the precision tilting angle of robot platform in real time should be obtained. There are some tilting sensor products on the market. They can offer filtered and optimized tilting angles directly, but the price of the sensor is very expensive (usually cost several thousands of US dollars). From economic considerations, in this robot design, the MEMS IMU sensor MPU-0150 is selected as the tilting sensor. In order to obtain the accurate tilting angles of the robot platform in real-time, two IMU data fusion algorithms are studied and compared. This chapter mainly focuses on the tilting measurement of robot platform while the tilting control strategy will be discussed in the next chapter.

6.1 Introduction of MEMS IMU

The IMU, or inertial measurement unit, is an electronic device that measures accelerations, rotation rates and possibly earth magnetic field with the use of tri-axis accelerometer, tri-axis gyroscope, and sometimes tri-axis magnetometer to determine an object's attitude or orientation. Due to its unique characteristics, IMUs have long been the subject of extensive research in aerospace [45] and navigation [46] fields although its size was bulky initially. In recent years, with the advent of MEMS (Micro-Electro-Mechanical-system) based IMU, the size of IMU is dramatically reduced to chip size along with the reduction in cost and power consumption [47]. Such a significant improvement in IMU made the research and applications of IMU quickly extended to many new areas, e.g., robotics and human motion analysis [48].

One of the applications of MEMS IMU is tilt measurement (or orientation sensing), which has a significant role in the fields of consumer electronics, robotics and navigation. A typical example is smart phone and handheld electronics. The control of menu options, image rotation or function selection all link to the tilt measurements [49]. In the robotics area, in terms of navigation robot, climbing robot, all-terrain robots etc., tilt sensing plays a key role in system balancing control. Michelle et al. in 2005 used a low-g MEMS accelerometer and trigonometric function relationship to measure the tilt of an object in a static environment. The result was simple and straightforward but noisy and nonlinear [49]. In 2013, Mark Pedley documented the mathematics of orientation using a MEMS three-axis accelerometer and also made an analysis on the regions of instability (gimbal lock) [50]. As the data from the accelerometer is in general very noisy and susceptible to external acceleration interference, when it is used to measure the gravitational acceleration, they still could not obtain accurate result in the vibrating environment (e.g. car or plane). Hence tilt measurement only using accelerometer is not effective enough though the accelerometer data is stable without drift in long term. A gyroscope offers angular velocities around the three axes and the signals are not susceptible to external forces comparing to accelerometers. Therefore, a tri-axis gyroscope and a tri-axis accelerometer (mutually orthogonal) based MEMS IMU seems to be a complete solution for tilt sensing. However, the gyroscope also has its own disadvantages that the data measured tends to drift because of the integration over time. In short, we can only trust the data from the gyroscope on a short term. Taking the advantages and disadvantages of gyroscopes and accelerometers into consideration, the IMU data fusion is essential for tilt sensing.

In terms of IMU data fusion algorithm, the complementary filter [51]- [57] and Kalman filter [58]- [63] are the most widely used algorithms. They have their unique advantages and disadvantages [64]. Kalman filter is an iterative filter, which is efficient but high computational complexity and burden. The complementary filter is relatively easy and computational light to implement, which makes it preferred for embedded systems. This chapter presents a research of IMU data fusion for tilt sensing based on a 6 DOF IMU. The results and the comparison of the two IMU data fusion algorithms applied for different scenarios are achieved through the applications of complementary filter and Kalman filter.

6.2 Complementary Filter

A complementary filter was proposed by Shane Colton [65]. For tilt sensing, the filter performs low-pass filtering on a low-frequency tilt estimation and the data is from the accelerometer while the high-pass filtering on a biased high-frequency tilt estimation is handled by directly integrating with the gyroscope output. The fusion of the two estimations gives an all-pass estimation of the orientation [51]. Obviously, the complementary filter takes the advantage of both accelerometer and gyroscope. On the short term, it uses the data from the gyroscope and the data is precise and not susceptible to external forces; on the long term, it relies on the data from the accelerometer to prevent data drift. The principle of the complementary filter is illustrated in figure 6.1.

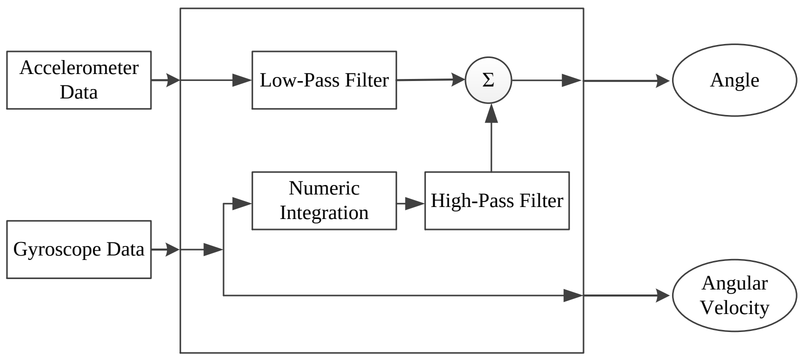


Figure 6.1 Block diagram of complementary filter algorithm for IMU data fusion

The goal of the low-pass filter is to only let through long-term changes, filtering out short-term fluctuations. One way to do this is to force the changes to build up little by little in subsequent times through the program loop. For example, if the angle starts at zero and the accelerometer reading suddenly jumps to 10 degrees and stays at that level, the angle estimate will smoothly rise to 10 degrees without spikes. The time it takes to reach the full value depends on both the filter parameters and the sample rate of the code loop (dt). The high-pass filter almost does the opposite: it allows short-duration signals to pass through while filtering out signals that are steady over time. This character is essential to cancel out the gyroscope drift in order to get an accurate estimate angle.

The mathematical model of the complementary filter can be represented as follows.

$$Angle = \alpha * (Angle + GyroData * dt) + (1 - \alpha) * AccData \quad (6.1)$$

"α" is the filter coefficient, "Angle" means the tilt angle (pitch and roll), "GyroData" and "AccData" represent the output of the gyroscope and accelerometer from the IMU respectively. Before applying the formula to calculate the angle, the data from the gyro and accelerometer must be zeroed and scaled.

$$\alpha = \frac{\tau}{\tau + dt} \qquad (6.2)$$

"τ" means the time constant of a filter. For a low-pass filter, the signals that are much longer than the time constant pass through the filter unaltered while the signals shorter than the time constant are filtered out. The opposite is true for a high-pass filter. "dt" is the sample period. For every time step, the gyroscope data is first integrated with the current angle and then combined with the low-pass data from the accelerometer. The filter coefficients (α and 1- α) have to add up to one, so that the output is an accurate, linear estimate in units that make sense [65].

The data processing procedure of the complementary filter in the IMU data fusion for tilt sensing is shown in Figure 6.2.

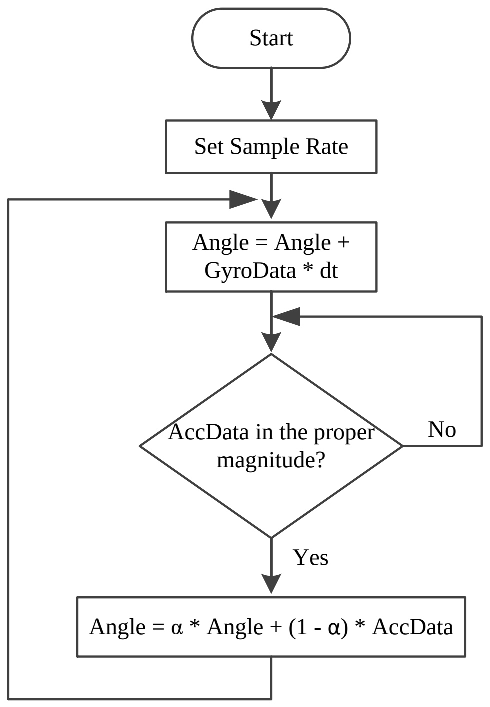

Start
Set Sample Rate
Angle = Angle + GyroData * dt
AccData in the proper magnitude?
No
Yes
Angle = α * Angle + (1 - α) * AccData

Figure 6.2 Complementary filter for tilt sensing flowchart

The complementary filter applied in the process of IMU data fusion has to be contained in an infinite loop. The pitch and roll angles are updated in every iteration with the new gyroscope values by means of integration over time. The filter then checks whether the magnitude of the force measured by the accelerometer has a reasonable value that could be the real g-force vector. If the value is too small or too big, it is not taken into account as a disturbance. Afterwards, it updates the pitch and roll angles with the gyroscope data by taking α of the current value and adding 1- α of the angle calculated by the accelerometer. This ensures the measurement is smooth, accurate and without drift.

The complementary filter is effective and has low computation burden in terms of IMU data fusion, but it is not easy to tune the filter coefficient which heavily relies on personal experience.

6.3 Kalman Filter

Since Kalman filter was first published by R.E. Kalman in 1960, due to its advances in digital computing, the Kalman filter has been the subject of extensive research and applications, particularly in the area of autonomous vehicles, navigation, robotic systems, and human motion control [66].

The Kalman filter, also known as linear quadratic estimation, is an iterative algorithm which relies on a series of measurements observed over time. Noises in measurement data contribute to the error. In the IMU data fusion process, the Kalman filter takes the noises into account via covariance matrices and updates the matrices at each time interval. It estimates the state of system based on the current and previous states, which tends to be more precise than the measurement alone. The key of Kalman filter based 6-DOF IMU data fusion algorithm is to find the weighted average (Kalman gain K), with more weight being given to the estimation with higher certainty. Such a process based on the accelerometer model and gyroscope model.

6.3.1 Accelerometer Model

An accelerometer measures all forces that are working on the object, which includes instantaneous linear acceleration as well as the gravitational acceleration plus some added bias and noise [47]. The accelerometer model can be represented as,

$$\alpha_{Acc} = \alpha_{extra} - g + b_a + n_a \qquad (6.3)$$

Where, α_{extra} , g, b_a, and $n_a$means external acceleration, gravitational acceleration, accelerometer bias and noise respectively.

6.3.2 Gyroscope Model

A gyroscope is used to measure the angular velocity around three mutually perpendicular axes. It is not easy to be affected by external interference, but it suffers from drift in long term.

$$\omega_{gyro} = \omega + b_g + n_g \qquad (6.4)$$

Here, b_g and n_g are the gyroscope bias and noise.

The Kalman filter operates by producing a statistically optimal estimation of the system state based upon the measurements. It needs to know the noise of the input to the filter called the measurement noise and the noise of the system itself called the process noise.

To make things easier, assume the noises are Gaussian distributed and have a mathematical expectation of zero.

6.3.3 Kalman Filter for IMU Data Fusion

Equation (6.5) and (6.6) are the standard Kalman filter formulas.

$$x_k = Fx_{k-1} + Bu_k + w_k \qquad (6.5)$$

$$z_k = Hx_k + v_k \qquad (6.6)$$

x_k is the system state matrix at time k, which is given by:

$$x_k = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_k \qquad (6.7)$$

The outputs of the filter are the angle θ and the bias $\dot{\theta}_b$ based upon the measurements from the accelerometer and gyroscope. The bias is the amount that the gyroscope has drifted. F is the state transition matrix which is applied to the previous state x_{k-1}. In this case, F is defined as:

$$F = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \qquad (6.8)$$

u_k is the control input. In this case it is the gyroscope measurement in degrees per second at time k, which is also called the angular rate $\dot{\theta}$. Then the state equation could be rewritten as:

$$x_k = Fx_{k-1} + B\dot{\theta}_k + w_k \qquad (6.9)$$

B is called the control matrix, which is defined as:

$$B = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \qquad (6.10)$$

This obtains the angle θ when multiplying the rate $\dot{\theta}$ by the time Δt. Since the bias cannot be calculated directly based on the angular velocity the bottom of the matrix is set to zero.

In Eqn. (6.5) and (6.6), the variables w_k and $v_k$represent the process and measurement noises respectively. They are assumed to be independent of each other and with normal probability distributions (Gaussian white noise) [66].

$$w_k = N(0, Q_k) \qquad (6.11)$$

$$v_k = N(0, R) \qquad (6.12)$$

Q_k is the process noise covariance matrix which represents the state estimation of the accelerometer and bias. If consider the estimate of the bias and the accelerometer to be independent, then Q_k is equal to the variance of the estimation of the accelerometer and bias.

$$Q_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \qquad (6.13)$$

The Q_k covariance matrix depends on the current time k, so the accelerometer variance Q_θ and the variance of the bias $Q_{\dot{\theta}_b}$ are multiplied by timeΔt. As the time goes, the process noise becomes larger since the last update of the state. The larger value set, the more noise in the estimation of the state. So, if the estimated angle starts to drift, the value of $Q_{\dot{\theta}_b}$ should be increased. If the estimated angle tends to lag, the value of Q_θ should be decreased to make it more responsive.

In equation (6.6), z_k is the measured output. H is the measurement matrix and is used to map the true state space into the observed space. The true state cannot be observed since the measurement is just from the accelerometer. H is given by:

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad (6.14)$$

The measurement noise covariance R is not a matrix. It is equal to the variance of the measurement noise since the covariance of the same variable is equal to its variance.

$$R = E\left[v_k \quad v_k^T\right] = \operatorname{var}(v_k) \qquad (6.15)$$

Assume that the measurement noise is the same and does not depend on the time k:

$$\operatorname{var}(v_k) = \operatorname{var}(v) \qquad (6.16)$$

If the measurement noise variance $var(v)$ is set too high, the filter responds slowly as it trusts new measurements less. On the contrast, if the value is set too small the filter overshoots and is noisy since it trusts the accelerometer measurement too much.

Therefore, the Kalman filter can be rewritten as:

$$\begin{aligned} \begin{pmatrix} \theta \\ \dot{\theta}_b \end{pmatrix}_k &= \begin{pmatrix} 1 & -\Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \theta \\ \dot{\theta}_b \end{pmatrix}_{k-1} + \begin{pmatrix} \Delta t \\ 0 \end{pmatrix} \dot{\theta}_k + w_k \\ z_k &= \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} \theta \\ \dot{\theta}_b \end{pmatrix}_k + v_k \end{aligned} \qquad (6.17)$$

6.3.4 Kalman Filter Implementation on IMU Data Fusion

The implementation of Kalman filter includes two steps: predict process and update process.

1) *Predict Process*

In predict process, the filter first estimates the current state and the error covariance matrix at time k. Equation (6.18) is for the estimation of the current state based on the previous state and the gyroscope measurement.

$$\hat{x}_{k|k-1} = F\,\hat{x}_{k-1|k-1} + B\,\dot{\theta}_k \qquad (6.18)$$

Here, $\hat{x}_{k-1|k-1}$ is the previous estimated state based on the previous state and the estimation of the states before. $\hat{x}_{k|k-1}$ is priori state which is the estimation of the state matrix at the current time k based on the previous state of the system. $\hat{x}_{k|k}$ is posteriori state which represents the estimation of the state at time k given observations up to and including at time k. The next step for the filter is to estimate the priori error covariance matrix $P_{k|k-1}$ based on the previous error covariance matrix $P_{k-1|k-1}$, which is defined as:

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k \qquad (6.19)$$

The matrix $P_{k|k-1}$is used to estimate how much trust can be put on the current values of the estimated state. The smaller the values are, the more trust on the current estimated state. Therefore, for the 6-DOF IMU case discussed, the error covariance matrix P is a 2 x 2 matrix:

$$P = \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix} \qquad (6.20)$$

2) Update Process

In the update process, the filter first computes the difference between the measurement z_k and the priori state $\hat{x}_{k|k-1}$.

$$y_k = z_k - H\hat{x}_{k|k-1} \qquad (6.21)$$

Then the filter calculates the innovation covariance:

$$S_k = HP_{k|k-1}H^T + R \qquad (6.22)$$

Eqn. (6.22) predicts how much we trust the measurement based on the priori error covariance matrix $P_{k|k-1}$ and the measurement covariance matrix R. The next step is to calculate the Kalman gain. The Kalman gain indicates how much we trust the innovation and is defined as:

$$K_k = P_{k|k-1}H^T S_k^{-1} \qquad (6.23)$$

For the 6-DOF IMU case, the Kalman gain is a 2 x 1 matrix:

$$K = \begin{pmatrix} K_0 \\ K_1 \end{pmatrix} \qquad (6.24)$$

Updating the posteriori estimate of the current state gives:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k \tag{6.25}$$

Finally update the posteriori error covariance matrix:

$$P_{k|k} = \left(I - K_k H\right) P_{k|k-1} \tag{6.26}$$

The Kalman filter algorithm in terms of IMU data fusion could be summarized as 7 steps: Eqn. (6.18), (6.19), (6.21), (6.22), (6.23), (6.25) and (6.26).

- *Step 1 - Predict*

$$\begin{aligned}
&\hat{x}_{k|k-1} = F\,\hat{x}_{k-1|k-1} + B\dot{\theta}_k \\
&\begin{pmatrix} \theta \\ \dot{\theta}_b \end{pmatrix}_{k|k-1} = \begin{pmatrix} 1 & -\Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \theta \\ \dot{\theta}_b \end{pmatrix}_{k-1|k-1} + \begin{pmatrix} \Delta t \\ 0 \end{pmatrix} \dot{\theta}_k \\
&= \begin{pmatrix} \theta - \dot{\theta}_b \Delta t \\ \dot{\theta}_b \end{pmatrix}_{k-1|k-1} + \begin{pmatrix} \Delta t \\ 0 \end{pmatrix} \dot{\theta}_k \qquad (6.27) \\
&= \begin{pmatrix} \theta + \Delta t(\dot{\theta} - \dot{\theta}_b) \\ \dot{\theta}_b \end{pmatrix}
\end{aligned}$$

- *Step 2 - Predict*

$$\begin{aligned}
&P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k \\
&\begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_{k|k-1} = \begin{pmatrix} 1 & -\Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_{k-1|k-1} \begin{pmatrix} 1 & 0 \\ -\Delta t & 1 \end{pmatrix} \\
&+ \begin{pmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{pmatrix} \Delta t \\
&= \begin{pmatrix} P_{00} - \Delta t P_{10} - \Delta t\left(P_{01} - \Delta t P_{11}\right) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} \end{pmatrix}_{k-1|k-1} \qquad (6.28) \\
&+ \begin{pmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{pmatrix} \Delta t \\
&= \begin{pmatrix} P_{00} + \Delta t\left(\Delta t P_{11} - P_{01} - P_{10} + Q_\theta\right) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{pmatrix}
\end{aligned}$$

- *Step 3 - update*

$$\begin{aligned} y_k &= z_k - H\hat{x}_{k|k-1} \\ &= z_k - \begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix}\theta \\ \dot{\theta}_b\end{pmatrix}_{k|k-1} \\ &= z_k - \theta_{k|k-1} \end{aligned} \quad (6.29)$$

- *Step 4 - update*

$$\begin{aligned} S_k &= HP_{k|k-1}H^T + R \\ &= \begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix}P_{00} & P_{01} \\ P_{10} & P_{11}\end{pmatrix}_{k|k-1}\begin{pmatrix}1 \\ 0\end{pmatrix} + R \\ &= P_{00k|k-1} + \operatorname{var}(v) \end{aligned} \quad (6.30)$$

- *Step 5 - update*

$$\begin{aligned} K_k &= P_{k|k-1}H^T S_k^{-1} \\ \begin{pmatrix}K_0 \\ K_1\end{pmatrix}_k &= \begin{pmatrix}P_{00} & P_{01} \\ P_{10} & P_{11}\end{pmatrix}_{k|k-1}\begin{pmatrix}1 \\ 0\end{pmatrix} S_k^{-1} \\ &= \frac{\begin{pmatrix}P_{00} \\ P_{10}\end{pmatrix}_{k|k-1}}{S_k} \end{aligned} \quad (6.31)$$

- *Step 6 - update*

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k y_k \\ \begin{pmatrix}\theta \\ \dot{\theta}_b\end{pmatrix}_{k|k} &= \begin{pmatrix}\theta \\ \dot{\theta}_b\end{pmatrix}_{k|k-1} + \begin{pmatrix}K_0 \\ K_1\end{pmatrix}_k y_k \\ &= \begin{pmatrix}\theta \\ \dot{\theta}_b\end{pmatrix}_{k|k-1} + \begin{pmatrix}K_0 y \\ K_1 y\end{pmatrix}_k \end{aligned} \quad (6.32)$$

- *Step 7 - update*

$$\begin{aligned} P_{k|k} &= \left(I - K_k H\right)P_{k|k-1} \\ \begin{pmatrix}P_{00} & P_{01} \\ P_{10} & P_{11}\end{pmatrix}_{k|k} &= \left\{\begin{pmatrix}1 & 0 \\ 0 & 1\end{pmatrix} - \begin{pmatrix}K_0 \\ K_1\end{pmatrix}\begin{pmatrix}1 & 0\end{pmatrix}\right\}\begin{pmatrix}P_{00} & P_{01} \\ P_{10} & P_{11}\end{pmatrix}_{k|k-1} \\ &= \begin{pmatrix}P_{00} & P_{01} \\ P_{10} & P_{11}\end{pmatrix}_{k|k-1} - \begin{pmatrix}K_0P_{00} & K_0P_{01} \\ K_1P_{00} & K_1P_{01}\end{pmatrix} \end{aligned} \quad (6.33)$$

The program flowchart of the Kalman filter in the IMU data fusion in tilt sensing is shown in figure 6.3. After several debugging, the following variances work perfectly for our MEMS IMU board (MPU-9150).

- $Q_\theta(Q\_angle)$ = 0.001
- $Q_{\dot{\theta}_b}(Q\_gyrobias)$ = 0.003
- R = 0.03

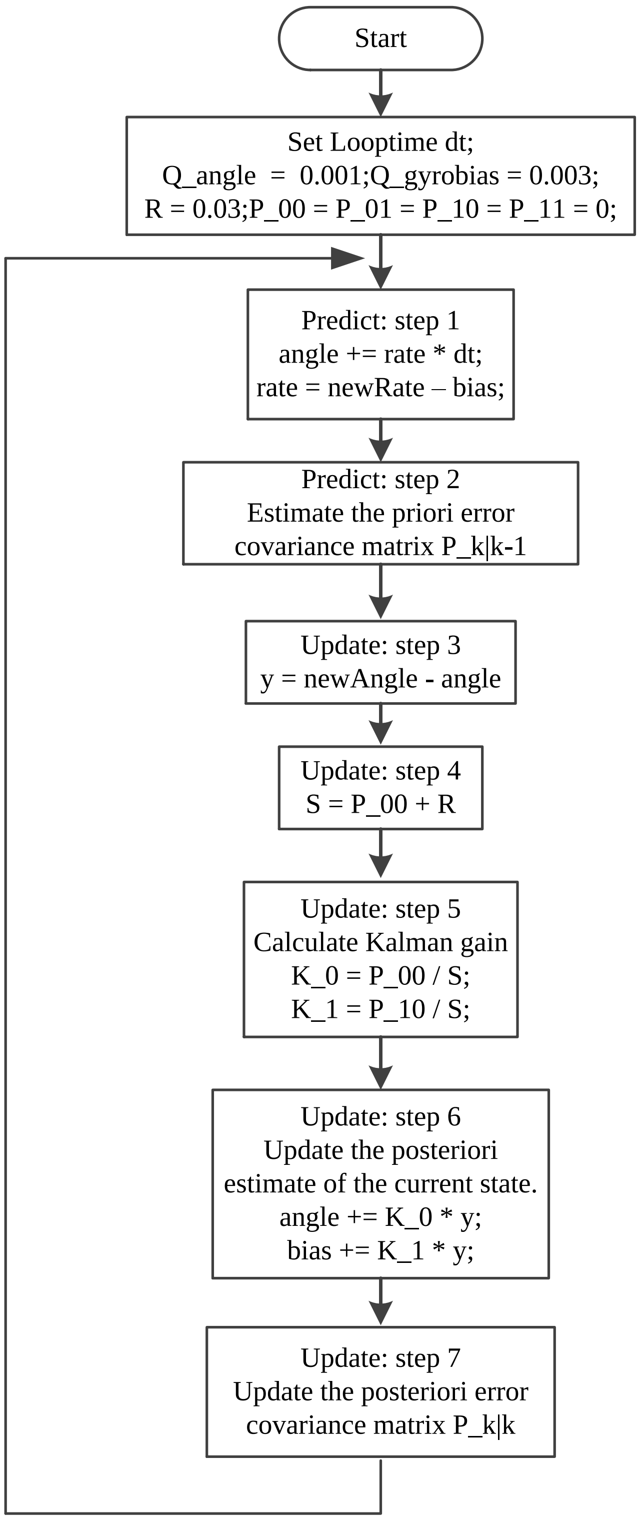


Figure 6.3 Kalman filter implementation flowchart for IMU data fusion

6.4 IMU Data Fusion Experiment Result

To investigate the behavior of Complementary filter and Kalman filter in IMU data fusion, MATLAB and Microcontroller based hardware were employed. The experiment is to evaluate the two algorithms and to find out the best suit coefficients for the two filters.

Arduino Uno (ATMEGA 328p)[18] is used as the MCU to acquire the 6-DOF IMU data and run the algorithms. The IMU used in the experiment is the InvenSense's MPU-9150, which offers 9-DOF (3-axis MEMS gyroscope, 3-axis MEMS accelerometer and 3-axis MEMS magnetometer). Only 6-DOF (gyroscope and accelerometer) is used in the experiment since 6-DOF data is sufficient for tilting sensing. After the raw data is processed by Arduino board, MATLAB drew the filtering data and compared with the unfiltered data obtained from the accelerometer. The angles obtained from the accelerometer for pitch and roll are:

$$\begin{aligned} AccPitch &= a\tan 2\left(AccData_z,\quad AccData_x\right) \\ AccRoll &= a\tan 2\left(AccData_z,\quad AccData_y\right) \end{aligned} \qquad (6.34)$$

Here, $AccPitch$and $AccRoll$ represent pitch and roll angle which are calculated from accelerometer data using anti-trigonometric function respectively. $AccData_x$, $AccData_y$and $AccData_z$ are gravitational acceleration components projected on three mutually perpendicular axes.

6.4.1 Complementary Filter Experiment Result

In the complementary filter experiment, the time constant "τ" and program loop-time is set at 0.75 and 0.03 (s) respectively. Then the complementary filter coefficient is calculated and α = 0.96. In the experiment, the output angles of the complementary filter are compared with angle $\theta_{AccPitch}$ and $\theta_{AccRoll}$. The experiment consisted of static and dynamic test. The dynamic test can be divided into three cases: single rotation around x-axis (roll), single rotation around y-axis (pitch) and rotation around x-axis and y-axis simultaneously (pitch and roll).

Figure 6.4 shows the static experiment result. It is obvious that the angle $\theta_{AccPitch}$ and $\theta_{AccRoll}$ ($AccPitch$ and $AccRoll$) derived from accelerometer contain a lot of noises and disturbances. It is hard to use such angle signal in tilting orientation system as it is

[18] ATmega328/P datasheet, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf.

vibratory and noisy. However, the filtered angles ($CFPitch$and $CFRoll$) through the complementary filter are much more stable and with less sparks and drift.

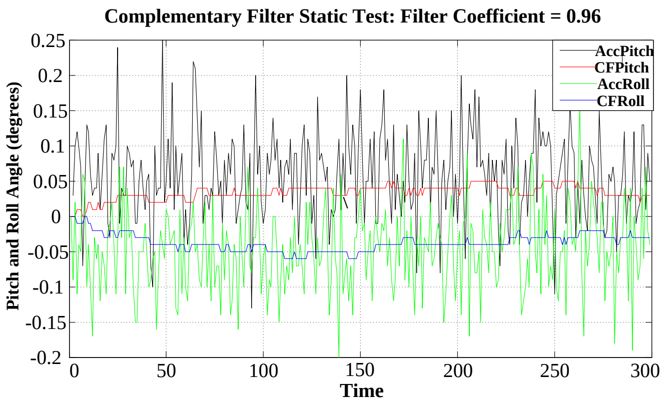


Figure 6.4 Complementary Filter Static Test

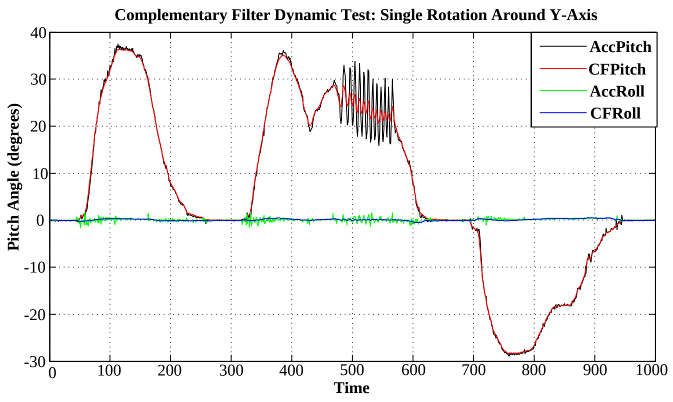


Figure 6.5 Complementary Filter Dynamic Test: Pitch

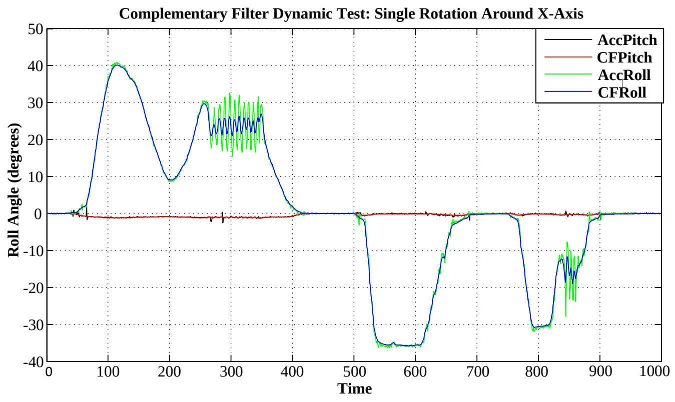


Figure 6.6 Complementary Filter Dynamic Test: Roll

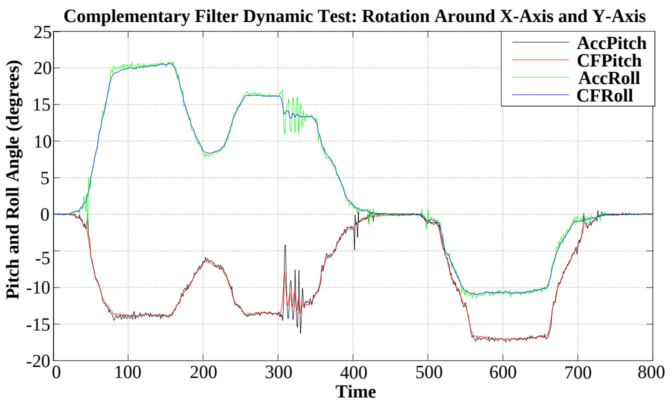


Figure 6.7 Complementary Filter Dynamic Test: Pitch and Roll

Figure 6.5 to Figure 6.7 show the outputs of the dynamic test of the complementary filter on IMU data fusion. Figure 6.5 and Figure 6.6 are the outcomes of the IMU rotates only around y-axis or x-axis. Figure 6.7 is the result that IMU rotates around x and y-axis simultaneously. Obviously, the filtered signal can nicely track the change trend of the signal derived from the accelerometer. Moreover, the signal filtered by complementary filter is smooth with less vibration, even some artificial shaking purposely added on the IMU board. When the IMU rotates around one single axis, there is almost no vibration on the other axis. This means there is no signal coupling happening on both axes (pitch and roll). Therefore, the complementary filter is efficient and reliable in IMU data fusion once the filter coefficient is fine tuned.

6.4.2 Kalman Filter Experiment Result

In the Kalman Filter IMU data fusion experiment, the filter parameters Q_θ (Q_angle), $Q_{\dot{\theta}_b}$ (Q_gyrobias) and R are set at 0.001, 0.003 and 0.03 respectively. And the program loop-time is set to 0.033s. These parameters are chosen after many trials. The same as in the complementary filter experiment, the $\theta_{AccPitch}$ and $\theta_{AccRoll}$ ($AccPitch$ and $AccRoll$) are used as the references to determine the effectiveness of the Kalman algorithm. The experiment also included static and dynamic tests. Figure 6.8 is the output of the static test while Figures 6.9 to Figure 6.11 present the dynamic results.

Figure 6.8 shows that, in the static test, the signal filtered by Kalman filter is more stable than that derived from the accelerometer. However, the filtered roll signal is smoother than the filtered pitch signal.

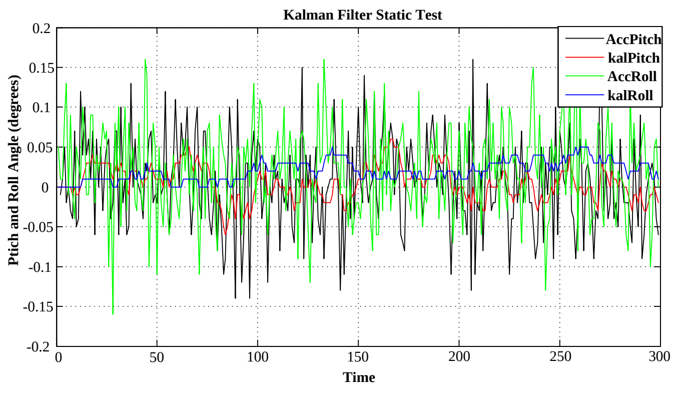


Figure 6.8 Kalman Filter Static Test

Figure 6.8 shows that, in the static test, the signal filtered by Kalman filter is more stable than that derived from the accelerometer. However, the filtered roll signal is smoother than the filtered pitch signal.

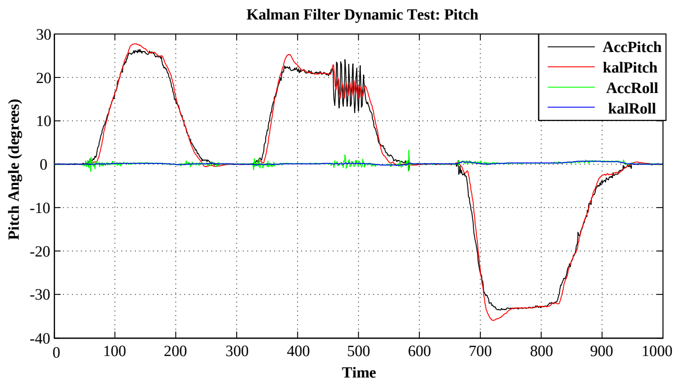


Figure 6.9 Kalman Filter Dynamic Test: Pitch

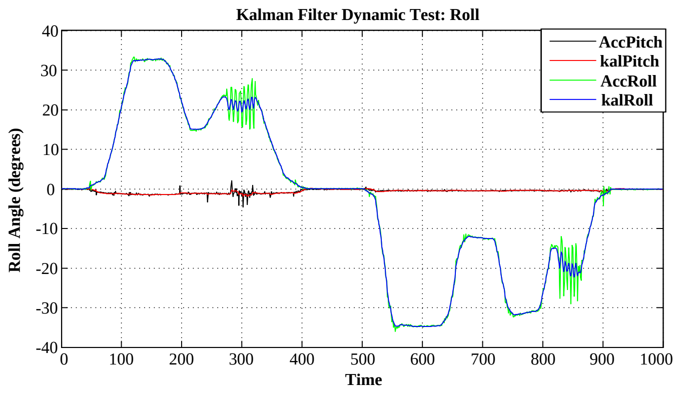


Figure 6.10 Kalman Filter Dynamic Test: Roll

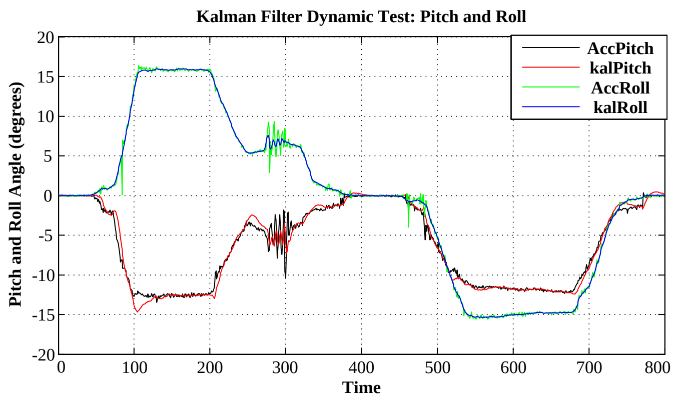


Figure 6.11 Kalman Filter Dynamic Test: Pitch and Roll

Figure 6.9, 6.10 and 6.11 are the outcomes of the Kalman filter dynamic test. Figure 6.9 and 6.10 show the IMU board rotates only around one axis while Figure 6.11 displays the IMU board rotates simultaneously around two axes in different directions. In Figure 6.9 and 6.10, the filtered signals were also able to nicely and smoothly follow the AccPitch and AccRoll signal even adding some artificial vibrations. And there was almost no signal coupling phenomenon. However, the result of "$kalRoll$" was better than the "$kalPitch$". There were small overshoot and lagging in the filtered pitch signal. This may be caused by the IMU PCB board manufacturing design fault. When rotating around two axes simultaneously, the Kalman filter also gives smooth and fairly accurate tracking signal.

Theoretically, the Kalman filter should have been more accurate than the complementary filter as it is a more complex and considerate algorithm that operates by producing a statistically optimal estimation of the system state based upon the measurements. It is an iterative process which always tries to find the statistically optimal value. However, the Kalman filter discussed in this case has three parameters that need to be tuned. This heavily increases the difficulty of the Kalman filter to achieve a more accurate result. In addition, the effectiveness of a Kalman filter demands computational complexity which requires a more powerful microcontroller. In the experiments presented, the program loop-time is set to 0.033s, which is the highest sample rate the Arduino ATMEGA 328p can handle. If the sample rate could be improved, the Kalman filter should get a better result.

6.5 IMU Date Fusion Filter Selection

Two IMU data fusion algorithms for tilt sensing are presented and analyzed. It is difficult to get an accurate tilt orientation only using accelerometer data because the signal derived from the inverse trigonometric function is sensitive to variation. Therefore, it contains vibration and cannot be used for precise tilt measurement. Gyroscope data is used to offset the disadvantage of data from accelerometer. However, gyroscope data suffers from drift on a long term. Hence filter algorithms are needed to solve such problems.

The Kalman Filter is one of the most widely used methods for tracking and estimation due to its optimality, tractability and robustness. It is an iterative algorithm which operates by producing a statistically optimal estimation of the system state based upon the measurements. However, under certain circumstances, it is difficult to apply due to its associated computation burden and relatively complicated theory. Moreover, for multiple state-variable situations, there are more filter parameters to be tuned. For such situations, complementary filter is ideal as it requires less computation and only one filter coefficient to be tuned, which greatly reduces the workload.

Therefore, the complementary filter is finally chosen as the IMU data fusion algorithm to acquire the smooth and accurate tilting results either in the static or dynamic situation. It showed less sensitive to variations and no signal coupling phenomenon. Given a fine-tuned filter coefficient, the result of complementary filter can be more stable and accurate than that of Kalman filter. After the selection of IMU data fusion filter, the issue of robot tilting measurement in real-time is solved. The tree pruning robot tilting control strategy will be discussed in the next chapter.

Chapter 7 Tree Pruning Robot Tilting Control Using Fuzzy Logic

This chapter presents an intelligent control strategy for this wheel-driven tree pruning robot tilting control during climbing procedure. There are two types of tilting scenarios needed to be considered, initial setting tilt and dynamic tilt. To compensate the initial installation tilt, a tiling fuzzy controller is specifically designed to adjust the robot platform to the horizontal position before climbing up. For robot dynamic tilt, which is mainly caused by wheel slippage during the climbing, a slippage fuzzy control system is developed through the control of stepper and DC motor via the IMU. These fuzzy controllers were designed in MATLAB Simulink and simulated in MATLAB SimMechanics. The simulations verify the feasibility of the robot tilting control algorithm and the effectiveness of the fuzzy controllers.

7.1 Definition of Two Types of Tilt for Tree Pruning Robot

The major part of a tree pruning robot is the climbing mechanism which determines whether the robot can climb up the trunk safely and stably. During the entire climbing process, no matter in the vertical or spiral pattern, an important issue that must be taken into consideration is to prevent the tilting of the robot platform. This is to guarantee the pruning tools installed on the platform can be kept in the desired orientation. In extreme situation, large tilting angle during climbing could cause the robot jammed on the trunk. Therefore, the tree pruning robot tilting control strategy must be developed.

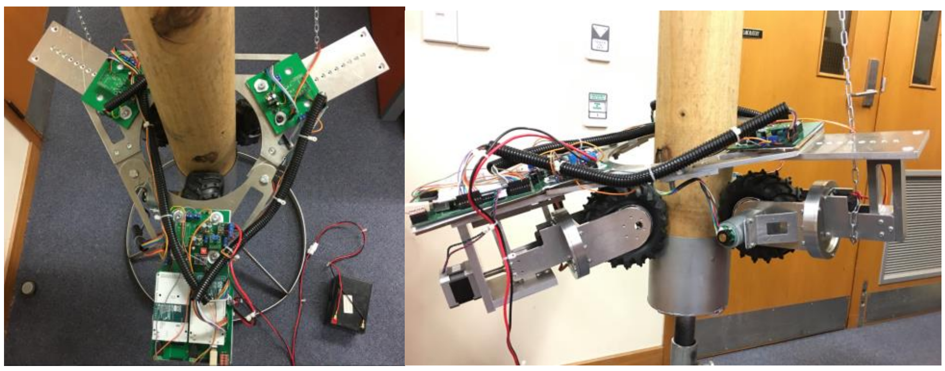


Figure 7.1 Top and lateral views of tree pruning robot in static situation using passive anti-falling mechanism

From the study and analysis of experimental results in robot climbing, there are two types of tilt required to balance: initial setting tilt and dynamic tilt. The initial setting tilt is caused by the passive anti-falling mechanism [67] because of the center of the gravity offsetting. The robot remains stationary at a certain tilting angle on the trunk utilizing its own weight without energy consumption (Fig. 7.2). This preset tilting angle is named as the initial setting tilt and illustrated in figure 2. During the climbing process, each of the three DC motors may suffer slippage due to the complicated trunk surface. Therefore, even if the three wheels keep the same speed during climbing procedure, the robot platform could dynamically tilt at a random direction. In an extreme situation, if the wheel suffers constant slippage, the robot may fall to ground or get jammed on the trunk. This is defined as the dynamic tilting.

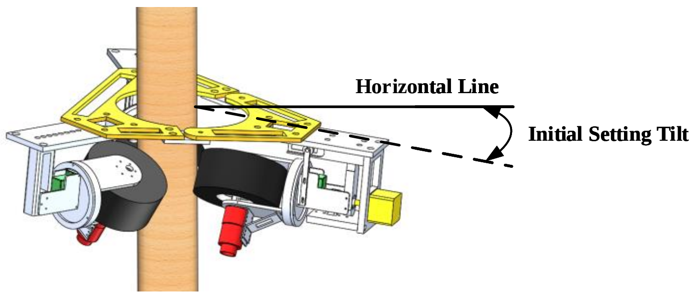


Figure 7.2 Tree pruning robot initial setting tilt

No matter which tilting case, to keep the robot platform horizontal, the control of the tilting angle of the platform is needed in real time. A MEMS based IMU module (MPU-9150) is installed on the platform which can offer pitch, roll and yaw angle through the onboard accelerometer and gyroscope. In order to obtain the reliable and accurate tilting angle, an IMU data fusion algorithm based on complementary filter [68] is adopted and illustrated in detail in the last chapter.

The pruning robot tilting control strategy can be divided into two parts. When the robot starts to climb up from the stationary status, the initial setting tilt control is employed to first make the robot platform horizontal. Once the platform tilting angle is in the pre-defined horizontal range, the dynamic tilt control is engaged to handle the fine tilting tuning while climbing.

The study and analysis of robot climbing tests reveal that the dynamic tilt is caused mainly by robot wheel slippage. Therefore, dynamic tilt could be fulfilled by wheel slippage control. In terms of wheel slippage control, many approaches have been proposed and studied in both robotics and automotive fields. These methods are based on different techniques and technologies, e.g., torque control, dynamic-model-based or vision-based control, and fuzzy logic [69] - [72]. Of course, each method is best suitable for its own specific system due to the characteristic of the system and required sensors [69].

Fuzzy logic theory is a powerful soft computing technique to control complex and non-linear systems based on human expert knowledge [73]. Compared to conventional control method, the biggest advantage of fuzzy control is that it can solve a complicated control problem without requiring the system's mathematic model. For this climbing robot, fuzzy logic control is adopted for robot platform tilting control. The tilting fuzzy controller is designed to handle the initial setting tilt while the slippage fuzzy controller is for the dynamic tilt.

7.2 Tilting Fuzzy Controller

The tilting fuzzy controller is designed to eliminate the initial setting tilt. According to passive anti-falling mechanism, robot mechanical structure and the IMU sensor mounting position, in this design the pitch angle of IMU sensor is aligned with the axis of lowest robot leg which is illustrated in Fig. 7.3. Under this circumstance, to balance the robot from stationary tilting condition, the robot only needs to control the robot platform's pitch angle. Therefore, the inputs of the tilting fuzzy controller are pitch error and change of pitch error of robot platform from horizon position while the output is the lowest DC motor speed.

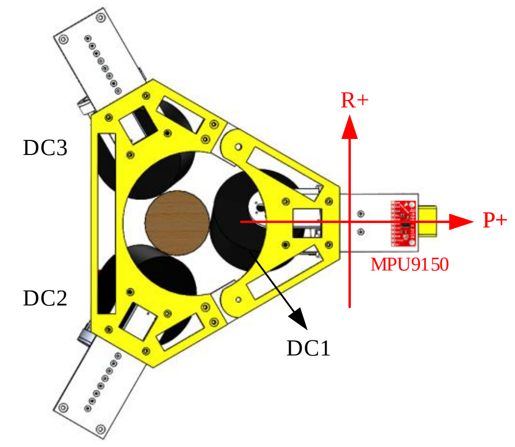


Figure 7.3 Mounting position of IMU sensor on tree pruning robot platform

A typical fuzzy controller consists of four parts: fuzzification interface, rule-base, inference mechanism and de-fuzzification interface [74]. In this tilting controller design, a Mamdani implication method is used [75] and the block diagram of this tilting fuzzy control system is shown in Fig. 7.4. The essential equation of Mamdani approach using fuzzy logic control is shown in Eqn. (7.1).

$$\mu_y(k) = \max\left[min\left\{\mu_u(k), \mu_r\left(e_p(k), e_r(k)\right)\right\}\right] \qquad (7.1)$$

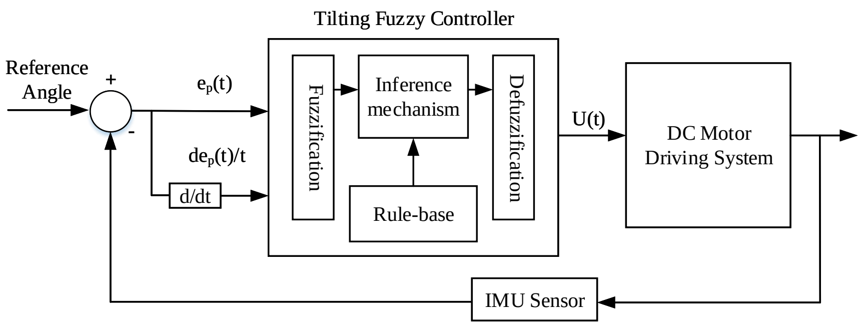


Figure 7.4 Block diagram of tilting fuzzy control system

To minimize the MCU computation cost, three fuzzy subsets are assigned for each input $e_{p(t)}$ $(pitch\ error)$ and $de_{p(t)}/t$ $(change\ of\ pitch\ error)$: N - negative; Z - zero; P - positive. This produced 9 rules for adjusting the lowest DC motor speed to keep the robot platform horizontal from initial tilting condition. The fuzzy controller output is classified as five membership functions: VS – very slow; S – slow; $Normal$ – 12 rpm; F – fast; VF – very fast. In this design, membership functions of inputs are selected as triangular and trapezoid functions while the output MFs are defined as triangular functions which are displayed in Figure. 7.5.

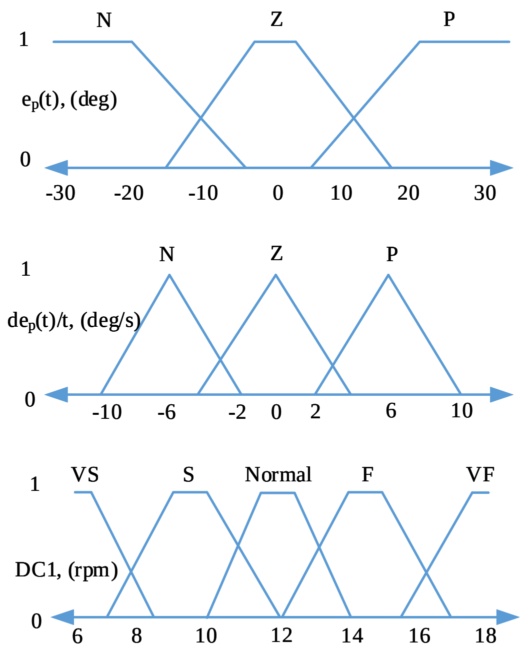


Figure 7.5 Membership functions of tilting fuzzy controller

The two inputs' universe of discourses are [-30, 30] and [-10, 10] while the universe of discourse of output is [6, 18]. The control rule base contains the following rules R_{ij}:

$$\text{IF } (e_p \text{ is } A_i)\ AND\ (de_p(t)/t\ is\ B_j)\ THEN\ (U\ is\ C_k) \qquad (7.2)$$

For i, j, and k = 1…5. A_i, B_j and C_k are fuzzy subsets defined by their corresponding membership functions. Table 7.1 represents the fuzzy rules of this tilting fuzzy controller.

Table 7.1. Rule table of tilting fuzzy controller

U(t)		$e_p(t)$		
		N	Z	P
$de_p(t)/t$	N	PB	PS	NS
	Z	PB	Z	NB
	P	PS	NS	NB

Centroid de-fuzzification is used to solve the simulation cases. However, for real implementation the weight average (WA) method is adopted to reduce the computational complexity. The equation of de-fuzzification used WA is shown in Eqn. (7.3), where b_i denotes the center of the membership function of the consequent of rule (i), u^{crisp} represents the conclusions of the fuzzy controller that are represented with the implied fuzzy sets and $\mu_{premise_{(i)}}$ means the certainty of premise rule (i) [74].

$$u^{crisp} = \frac{\sum_i b_i \mu_{premise_{(i)}}}{\sum_i \mu_{premise_{(i)}}} \qquad (7.3)$$

7.3 Slippage Fuzzy Control System

A typical problem of a wheel type tree climbing robot is the wheel slippage. During robot climbing, even if the three DC motors' rotation speeds are the same, once one of the wheels suffers slippage, the robot platform begins to tilt.

The slippage not only causes the whole robot climbing speed error and distance error, but also increases the overall energy consumption and decreases robot locomotion performance (dynamic tilt) [69]. In this tree pruning robot design, wheel slippage is the primary reason for robot dynamic tilt. To tackle this issue without extra sensors or high MCU computation load, the slippage fuzzy control system composed of one stepper unit and one IMU sensor has been developed based on experimental data collected from three DC motor encoders.

The stepper and screw-nut unit play an essential role in wheel slippage control. When the nut of the screw-nut unit moves towards the trunk, the three wheels' normal forces increase. If the friction coefficient is constant, the total friction forces between the robot wheels and the tree trunk increase correspondingly. Therefore, the wheel slippage control can be fulfilled by adjusting the step motor to control the nut position. The structure of the stepper and screw-nut unit is displayed in Fig. 7.6.

As illustrated in chapter 4, to guarantee the lead screw efficiency and control accuracy of the stepper unit, a bearing support mechanism is adopted. It aims to overcome and eliminate the robot wheel module gravity force and other disturbing forces applied on the robot wheel that affect the servomotor module and stepper's nut and screw unit.

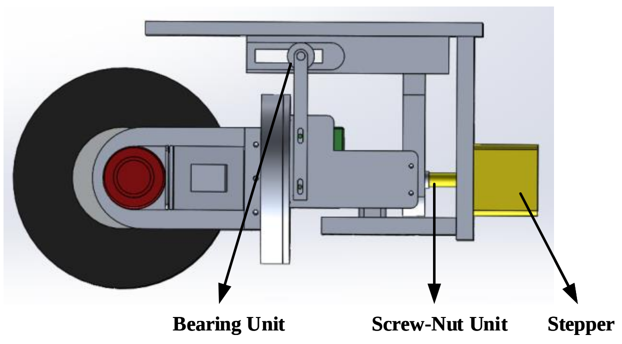


Figure 7.6 Tree pruning robot stepper and screw-nut unit

According to the Robot climbing experimental data, a slippage fuzzy control system is developed. The inputs are the two errors of platform pitch and roll angles referencing to the horizontal position while outputs are the steps of the sept motor and the rotational speeds of the three DC motors. The block diagram of slippage fuzzy control system is shown in Figure. 7.7.

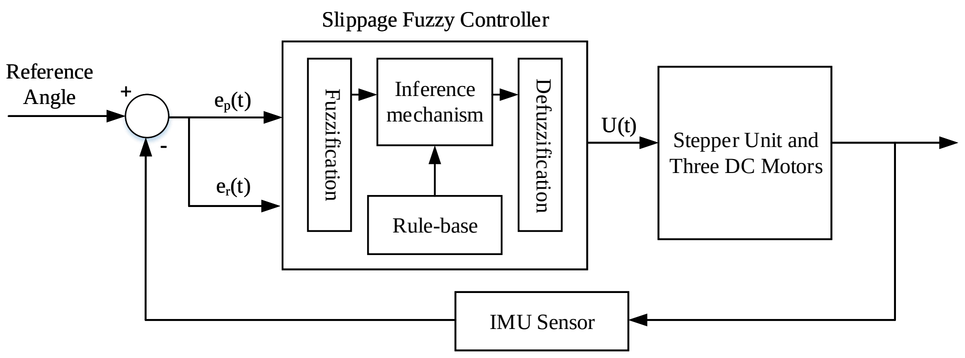


Figure 7.7 Block diagram of slippage fuzzy control system

The membership function of this slippage fuzzy control system is illustrated in Figure 7.8. Each input and output is classified as five MFs and their membership function types are considered as triangular ones. The universe of discourse of input is [-5, 5], for stepper outputs, they are [-100, 100] and [4, 20] for three DC motors.

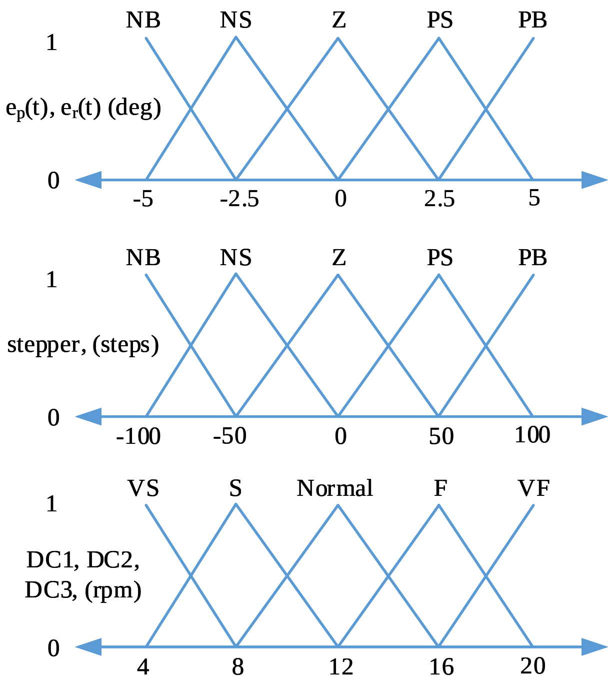


Figure 7.8 Membership functions of slippage fuzzy controller

The surface view of the slippage fuzzy controller in Simulink is shown in Figure 7.9. It illustrates the relationship between the controller's two inputs and four outputs.

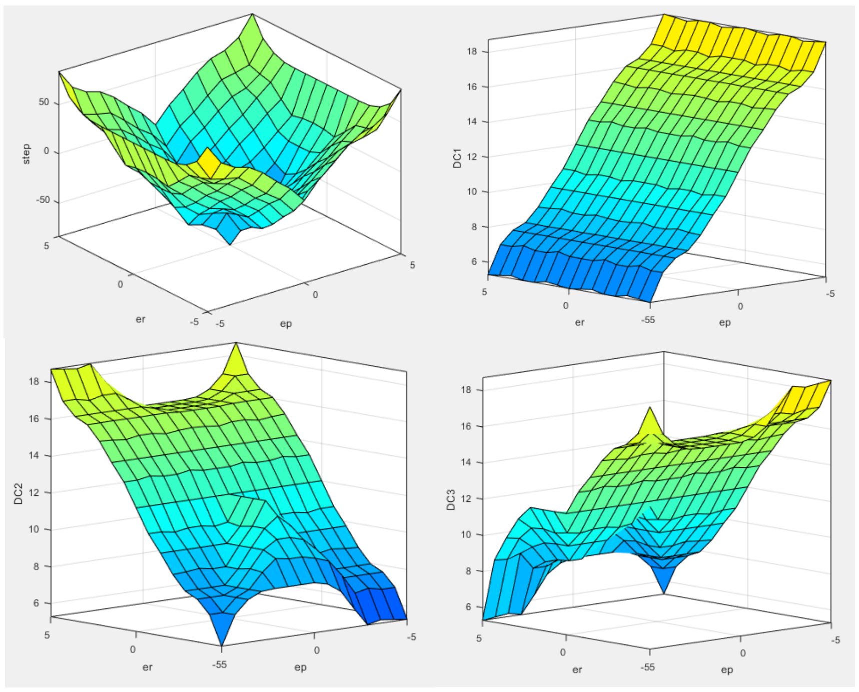


Figure 7.9 Slippage fuzzy controller surface view in Simulink

7.4 Tilting Control System Block Diagram

The overall tree pruning robot tilting control system block diagram is illustrated in Figure. 7.10. In this control flow chart, microcontroller Arduino Due plays a host computer role. It employs a complementary filter to obtain the stable and accurate tilting data by distilling and fusing the raw pitch and roll data from IMU. The tilting fuzzy control and slippage fuzzy control are also operated on this MCU. Except Arduino Due, one Arduino Mega is employed as the slave computer to actuate the robot driving systems and return feedbacks according to the commands from Arduino Due. The communication between Arduino Due and Mega relies on the serial communication (UART).

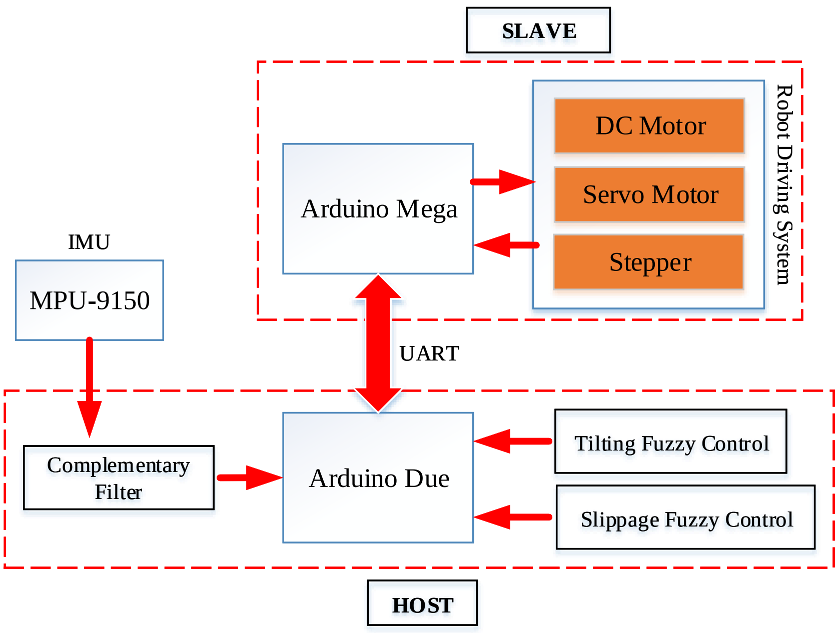
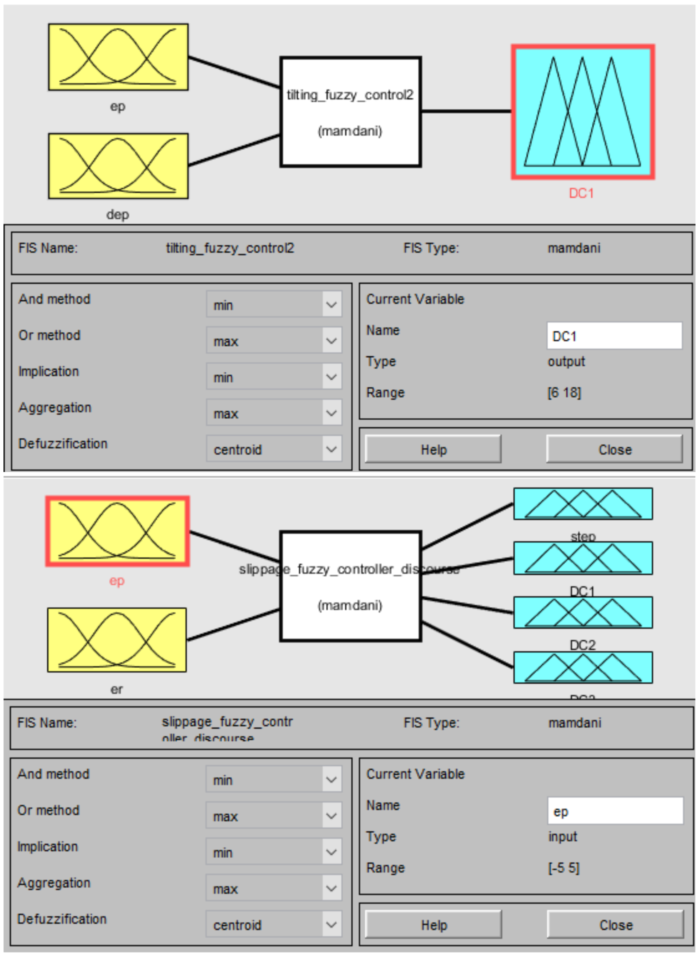


Figure 7.10 Block diagram of robot tilting control system

7.5 Simulation Results of Robot Tilting Control

The two fuzzy controllers are designed in MATLAB Simulink, which is shown in Figure. 7.11. To verify the effectiveness of the tilting fuzzy control system, it is imported into the MATLAB SimMechanics and simulated with the robot model using robot physical climbing data. The block diagram of slippage fuzzy control system with climbing robot in MATLAB SimMechanics is displayed in Figure. 7.12.



FIS Name: tilting_fuzzy_control2 FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid

Current Variable
Name: DC1
Type: output
Range: [6 18]

Help Close

FIS Name: slippage_fuzzy_controller_discourse FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid

Current Variable
Name: ep
Type: input
Range: [-5 5]

Help Close

Figure 7.11 Tilting fuzzy controller and slippage fuzzy controller in MATLAB Simulink

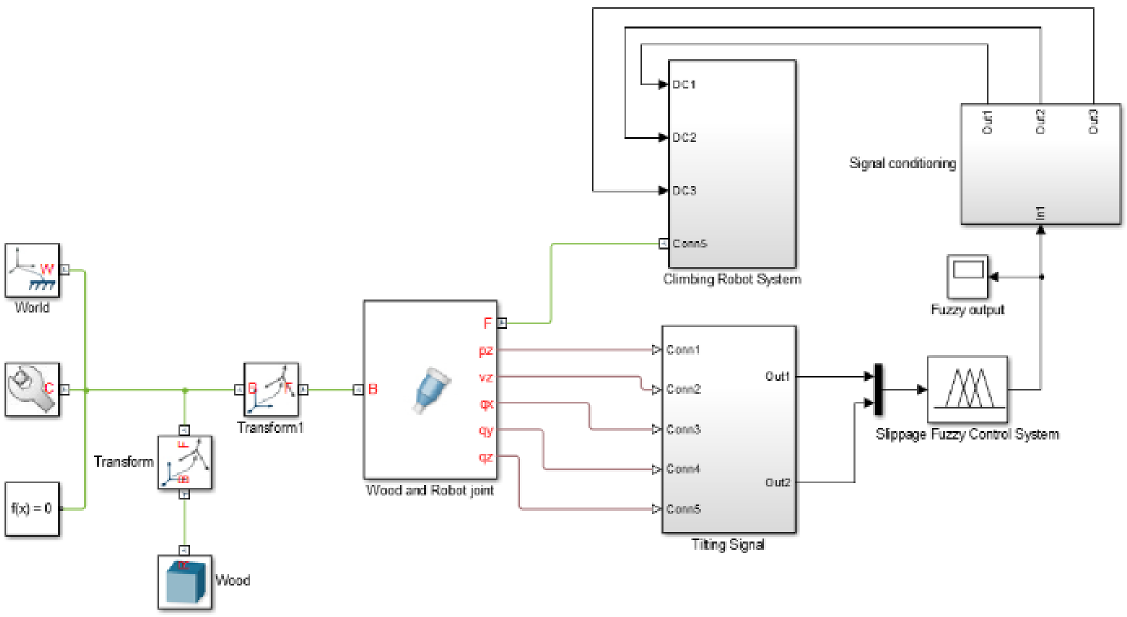


Figure 7.12 Slippage fuzzy control system and climbing robot in MATLAB SimMechanics

The simulation results of slippage fuzzy control system are displayed in Figure. 7.13 and 7.14, which illustrate the tilting differences of robot in pitch and roll angles during robot climbing procedure. The orange-color line represents the robot system using slippage fuzzy controller while the blue line without fuzzy system.

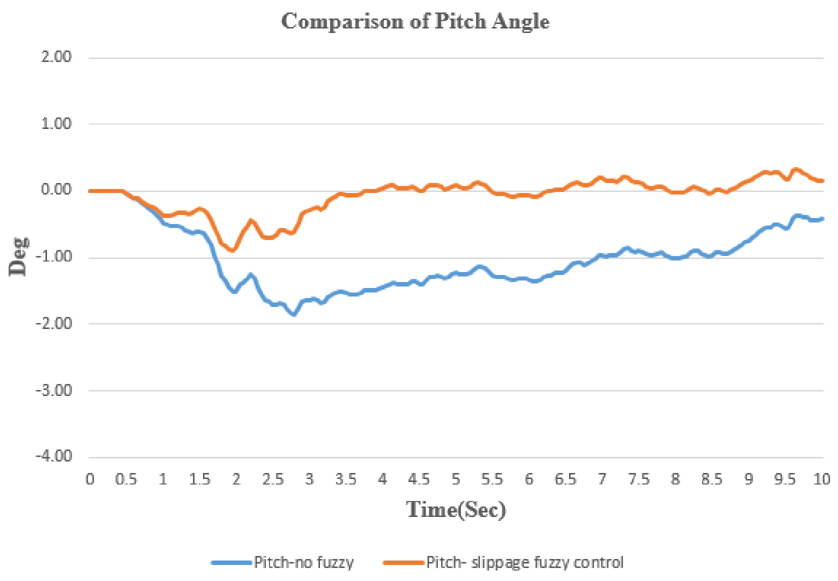


Figure 7.13 Tree pruning robot pitch angle comparison in MATLAB SimMechanics simulation

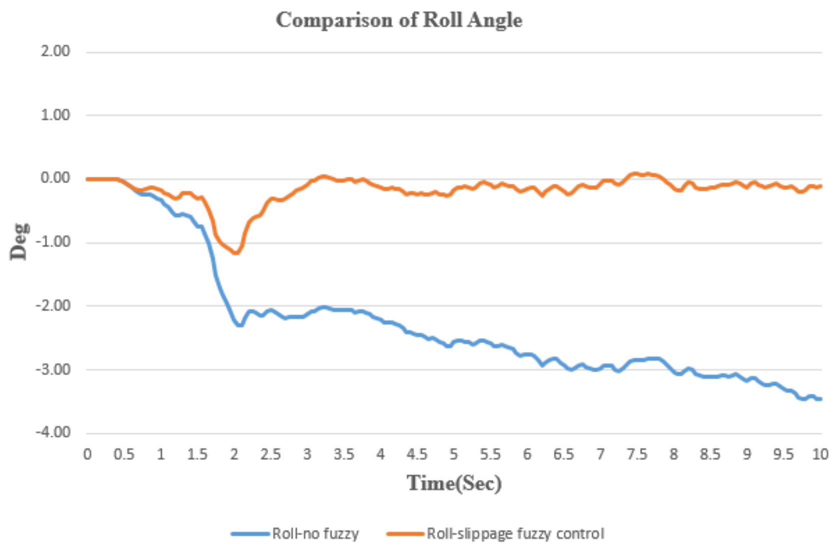


Figure 7.14 Tree pruning robot roll angle comparison in MATLAB SimMechanics simulation

As shown in Figure. 7.13 and 7.14, during robot climbing process, without slippage fuzzy control, the robot suffers tilting issue (blue line). There is a drift from the horizontal position. However, when the robot employs the slippage fuzzy control, the situation is significantly improved (orange line).

The robot utilizes the slippage fuzzy control method to adjust the three DC motors and the stepper according to the tilting angle obtained from the physical climbing data. After 2 seconds, the pitch and roll angle of climbing robot is fluctuating at a small amplitude around zero degree which means the robot climbs up horizontally. RMSE is used as the performance index. With a RMSE value of 1.046 for pitch angle and 2.317 for roll angle.

7.5 Robot Tilting Control Conclusion

Two fuzzy controllers are designed to solve the tilting problem of wheel-driven tree climbing robot. The tilting fuzzy controller is used to tackle the initial presetting tilt while the slippage fuzzy control system is developed to control and eliminate the dynamic tilt during robot climbing process. Both fuzzy controllers are developed in MATLAB Simulink and verified with the robot model in MATLAB SimMechanics. The simulation results confirm that the slippage fuzzy control system is feasible and effective to solve the robot climbing tilting problem.

However, this research still remains work for further study. For example, the specific implementation of these two tilting fuzzy controllers in tree climbing robot control system which requires the fine tuning of fuzzy controller coefficients according to different climbing environment and climbing morphologies.

Chapter 8 Tree Pruning Robot Longitudinal Wheel Slip Control Using Dynamic Neural Networks

The control of wheel slip is a challenging problem due to the complex and nonlinear dynamics of tire-surface interaction. When the wheel slip is not limited in its optimally stable zone, for wheel-driven tree climbing robots, the desired tire-trunk longitudinal force cannot be achieved, which not only causes distance error, but also increases overall energy consumption and decreases the climbing performance. In this research, a new approach based on dynamic neural networks (DNN) and direct inverse control (DIC) were employed for improving the tree climbing robot longitudinal wheel slip control. This control strategy is fulfilled based on the dynamic adjustment of the stepper's lead screw thrust (or steps) according to the wheel slip stable zone constraints. Robot wheel slip is obtained using online wheel slip estimation method which is derived from wheel speed errors and verified through offline wheel slip estimation. The dynamic neural network has been used for modelling of a nonlinear relationship between the tree climbing robot' stepper lead screw thrust and the longitudinal wheel slip during the vertical climbing procedure. It provided preconditions for control of the stepper lead screw thrust based on the wheel slip change.

8.1 Introduction

The physically manufactured tree pruning robot testing prototype is previously illustrated in chapter 4. Its mechanical construct is depicted in Figure. 8.1. To obtain high climbing speeds, wheel mechanism is adopted. The robot driving system has three legs evenly distributed under the platform. These legs consist of the driving mechanism which supports the entire pruning robot. Each leg is composed of four major parts: servo motor, DC motor, linkage, and wheel unit. In addition to these parts, one leg is equipped with a stepper and a screw-nut unit as shown in Figure. 8.2.

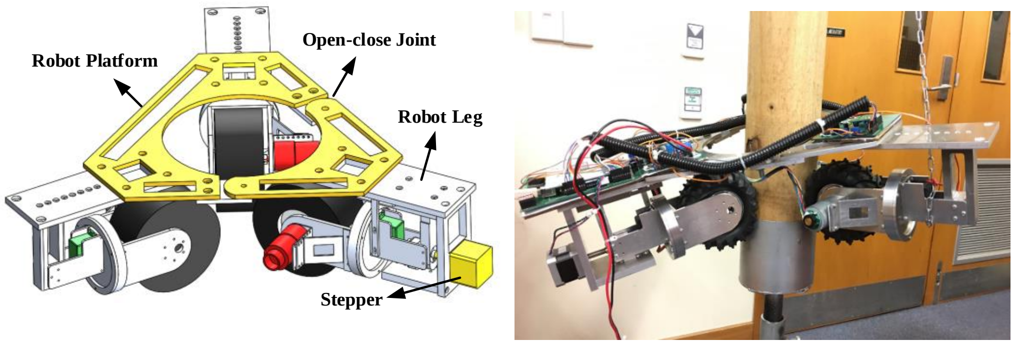


Figure. 8.1 Mechanical construction of the tree pruning robot

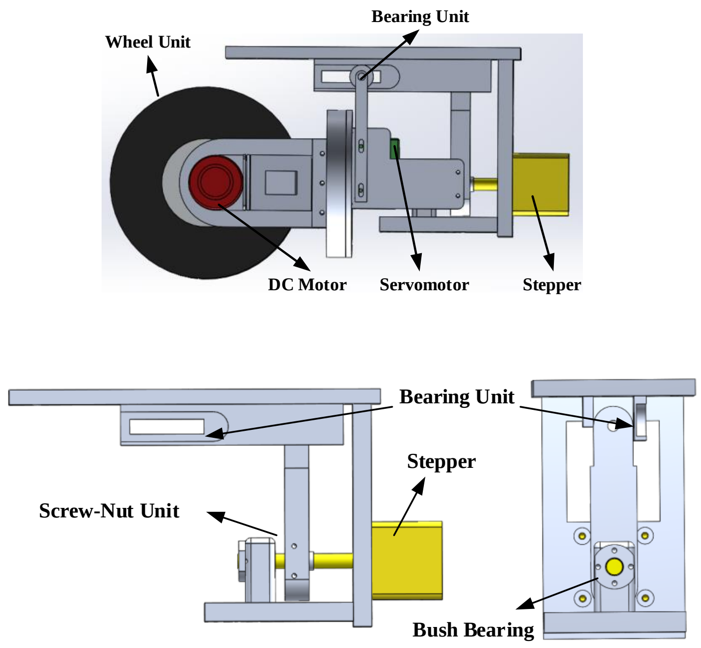


Figure. 8.2 Climbing robot leg module and stepper screw-nut unit

This step motor together with the screw-nut unit is the primary mechanism that adjusts the diameter of the circle constructed by the three robot wheels to suit different trunk sizes. In this design, according to the dimension of the screw-nut unit and robot tire sidewall (120×60mm), the trunk diameter that the robot can climb up ranges from 120 to 160 mm [76]. Furthermore, during the climbing process, the stepper can adjust the normal forces between the robot wheels and the tree trunk through the screw-nut mechanism. This holding force adjustment plays a key role in the robot wheel slippage control. The other two legs are fixed legs with a wheel against the tree trunk without installing step motors.

From the robot physical climbing tests on a rod, it is found that for 10 seconds, the actual climbing distance is about 42 cm which is smaller than the target 75.36 cm (12 rpm) mainly due to robot wheel slippage [76]. Wheel slip is a typical problem for wheeled tree or pole climbing robots in the estimation of climbing distance. During the robot climbing process, when the applied DC motor tractive force exceeds the level of maximum longitudinal tire adhesion available at the tire-trunk surface, the wheel slip occurs. The slippage not only enlarges the distance error, but also increases the overall energy consumption and decreases the climbing performance of the robot. Owing to the complicated trunk surface condition, three robot wheels have different friction coefficients. Therefore, during the climbing process, if the robot holding force is constant, the three longitudinal wheel forces are dynamic changing. In the extreme situation, if the robot wheel suffered serious slippage, it may have the chance to fall to the ground.

Therefore, to ease this situation and guarantee the robot climbing up the trunk quickly and stably, the wheel longitudinal slip should be regulated at its optimum value in order to generate the maximum longitudinal adhesion. In this chapter, an approach based on dynamic neural networks has been employed for improving the tree climbing robot longitudinal wheel slip control. This control strategy is fulfilled based on the dynamic adjustment of the stepper's lead screw thrust and three DC motors' velocity errors according to the wheel slip stable zone constraints and robot platform tilting angles. Robot wheel slip is estimated from the experimental test data and difference of three wheels' angular velocities during climbing. A nonlinear relationship between the tree climbing robot's stepper lead screw thrust and the longitudinal wheel slip during the vertical climbing procedure was modelled which provides preconditions for control of the stepper steps based on the wheel slip change. The proposed controller can also dynamically adjust the tractions of three wheels according to the wheel slip and finally improve robot climbing performance.

8.2 Research Status of Wheel Slip Control

In the last ten years, wheel slip control has been studied extensively. Numerous approaches have been proposed, not only within the robotics community, but also more in the automotive industry. For instance, the active safety systems like ABS, ESP in passenger cars have been broadly studied and applied. Many advanced wheel slip control technologies are also proposed and developed.

A. Harifi et al [77] present a sliding mode controller for wheel slip control based on a two-axle vehicle model. Important considered parameters for vehicle dynamic include two separated brake torques for front and rear wheels as well as longitudinal weight transfer caused by the acceleration and deceleration. Integral switching surface was utilized to reduce the chattering effects. Simulation results show the success of integral switching surface in elimination of chattering side effects and by high performance of the controller.

Velimir Ćirović and co-workers [78-79] proposed an approach based on dynamic neural networks to improve the vehicle longitudinal wheel slip control. This approach is based on dynamic adaptation of the brake actuation pressure, during a braking cycle, according to the identified maximum adhesion coefficient between the wheel and road. The brake actuated pressure was adjusted on the level which provides the optimal longitudinal wheel slip versus the brake actuated pressure selected by a driver, the current vehicle speed, load conditions, the brake interface temperature and the current value of the wheel slip. A nonlinear functional relationship between the brake pressure and wheel slip was modelled to offer preconditions for the wheel slip control.

Lei Yuan et al [80] describes a new slip control system for electric vehicles equipped with four in-wheel motors, based on nonlinear model predictive control (nonlinear MPC) scheme. To ensure vehicle safety, wheel slip stable zone is considered as time-domain constraints of the nonlinear MPC. Besides, the motor output torque is limited by the motor maximum torque which varies with motor angular velocity and battery voltage, so the motor maximum output torque limitation is considered as system time-varying constraints. The effectiveness of the proposed controller is verified in the off-line con-simulation environment of AMESim and Simulink, and a rapid control prototyping platform based on FPGA and dSPACE is completed to evaluate the real-time functionality and computational performance of the nonlinear MPC controller.

P. K. Pranav et al [81] designed a microcontroller based automatic wheel slip control system for a 2WD tractor to replace the inefficient draft control system. The new system measures wheel slip under field conditions and generates commands for depth adjustment if the wheel slip falls outside the desired range. Wheel slip was calculated using the actual and theoretical speeds of tractor obtained by measuring the rotational speed of front and rear wheels, respectively. The performance data indicated a significant reduction in fuel consumption per hectare and gain in tractive efficiency with slip control system versus the existing draft control system.

In robotic areas, especially for the wheel mobile robots, wheel slip control is also studied extensively. Daisuke Chugo et al [82] developed a holonomic mobile system which is capable of running over the step. In order to realize the high mobile performance during step climbing, a new plural wheel control method based on wheel slip estimation is proposed to reduce wheel slippage for maximizing wheel traction. The key idea is the estimation of wheel slippage comparing with the loads and rotation velocities of all actuated wheels and using this result for wheel control of the vehicle for reducing wheel slippage. The controller can adjust control tractions of plural wheels when the wheel begins to slip and improve the mobile performance of the vehicle.

W. Chonnaparamutt and H. Kawasaki [30] developed a slippage control system based on experimental data collected from encoders and a motion capture system to cope with wheel slip without spending extra sensors or high processing load. The slippage control system composed of two fuzzy modules, namely the trajectory estimator and velocity controller. The control system applied the cross-coupling control technique by employing the estimated velocity from the estimator as a part of an input for the velocity controller of four wheels. In this way, a simple yet effective slippage control system was designed.

P. Lamon et al [83] presented a quasi-static modelling of a six-wheeled robot with a passive suspension mechanism to select the optimal torques considering the system constraints: maximal and minimal torques, positive normal forces. The aim of this research is to limit wheel slip and to improve climbing capabilities. The proposed modelling and optimization were applied on a Shrimp rover.

Besides, there are many others' wheel slip control techniques and approaches available in different application areas [84-89]. These proposed wheel slip methods are based on several techniques, e.g. torque control, vision-based or dynamic-model-based control, sliding mode control, fuzzy logic and neural networks. These approaches give impressive results for regulating or moderating the wheel slippage. However, most methods are suitable for each specific system owing to the characteristic of the system and required sensors. For this three-wheeled tree climbing robot wheel slip control, taking the available MCU computation resources and affordable sensors into consideration, the neural network is adopted as wheel slip control method. Moreover, the control of wheel slip during climbing process is a challenging task due to nonlinear dynamics between the robot longitudinal climbing force and the steps of stepper, followed by uncertainty and dynamic varying of the trunk parameters such as the friction coefficient and trunk diameter. In this occasion, neural network is more suitable and competent for such a complicated task.

8.3 System Modelling and Problem Statement

In order to design a slip control law for wheeled tree climbing robot, a nonlinear slip control system model based on robot climbing dynamics is developed in this section, which mainly consists of robot climbing dynamics, tyres, wheel motors and stepper. Basically, a comprehensive tree climbing robot model that includes all relevant characteristics of the robot (both longitudinal and lateral dynamics) should be considered together. However, this paper only focuses on the influence of slip control system on the robot's longitudinal (or vertical) dynamic performance, and in order to deal with the slip control issue easily, the assumptions and neglections are made to study the robot dynamics in longitudinal direction alone, including: weight transfer and lateral wheel slip are not considered. The symbols in modelling procedure and the related physical meanings are listed in Table 8.1.

Table 8.1 Tree Pruning Robot Nomenclature

Symbol	Description
M	Robot mass
$v, \dot{v}, \omega, \dot{\omega}$	Robot velocity, robot acceleration, wheel angular velocity, wheel angular acceleration
$F_x, F_{x1}, F_{x2}, F_{x3}$	Robot wheel longitudinal force
G	Robot gravity force
F_{roll}	Wheel rolling resistance
J_i	Moment of inertia of robot wheel
T_{mi}, T_M	Tractive torque of robot wheel motor
R	Wheel rolling radius
T_{ri}, T_r	Wheel rolling resistance torque
T_G	Robot gravity torque
T_F	Robot wheel longitudinal torque
F_s	Thrust from stepper lead screw
μ	Trunk friction coefficient
N	Wheel resultant normal force
F_{lead}	Lead screw actuate force
f_s	Stepper thread friction force coefficient
N_s	Stepper thread normal force
d	Stepper thread diameter
l	Stepper thread lead
T_s	Stepper actuate torque
v_s	Stepper nut linear velocity
ω_s	Stepper lead screw angular velocity
η	Stepper efficiency
E_r	Stepper rotational kinetic energy
E_t	Stepper translational kinetic energy
D	Stepper nut translation distance
θ	Stepper lead screw rotation angle
δ	Stepper step angle
n	Stepper steps

8.3.1 Slip control system model

1. Robot dynamic model

Since this research mainly focuses on the influence of slip control on the robot's longitudinal dynamic performance, the robot dynamics model adopted here mainly contains the robot longitudinal motion and the rotational movement of the three wheels. According to Newton's law, the motion equations of the robot are shown as follows.

a) ROBOT LONGITUDINAL MOVEMENT

$$M\dot{v} = F_{x1} + F_{x2} + F_{x3} - G - F_{roll} \tag{8.1}$$

As shown in Fig. 8.3, F_{x1}, F_{x2}, F_{x3} are the tyre longitudinal forces of the three wheels.

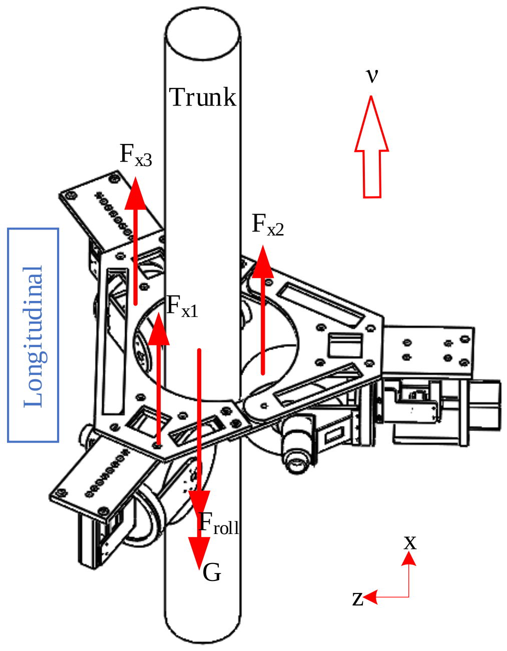


Figure. 8.3 Robot dynamics during vertical climbing

According to the mechanical construction of tree climbing robot in Fig. 8.1, to illustrate the dynamic model more easily and clearly, the wheel connected with a stepper unit is selected as the tire dynamic model which is displayed in Fig. 8.4.

b) THREE WHEEL'S ROTATION MOVEMENT

$$J_i\dot{\omega}_i = T_{mi} - F_{xi}R - T_{ri}\,; \quad i = 1, 2, 3 \tag{8.2}$$

Where R is rolling radius of the wheel, T_{mi} is wheel i's motor output torque, T_{ri} is wheel i's rolling resistance torque.

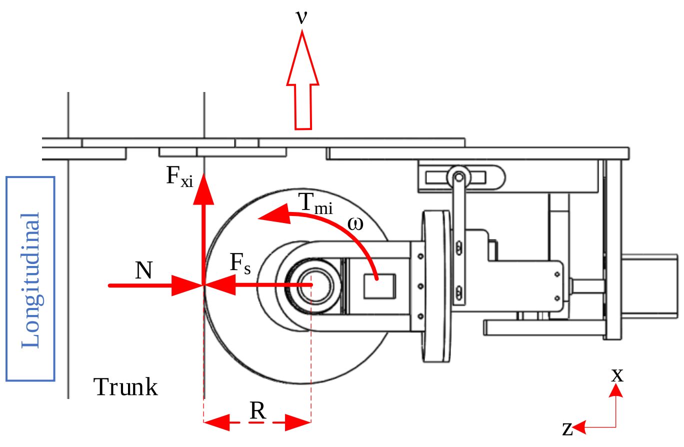


Figure. 8.4 Robot wheel dynamics during vertical climbing

2. Wheel motor modelling

$$T_M = T_F + T_G + T_r \qquad (8.3)$$

T_M is the whole robot wheel torque required for three DC motors during climbing. To enable the robot climbing the trunk, the three-wheel tractive torques need to overcome the robot gravity, tire longitudinal force and wheel rolling resistance. During the climbing process, assuming the wheel rolling resistance and gravity force are constant and other disturbances are neglected. When the wheel motor tractive torque exceeds the wheel longitudinal force, wheel slip occurs. In fact, the robot wheel slip is inevitable since the three wheels' longitudinal forces are dynamically fluctuating in the climbing procedure. The reason for this phenomenon is the complex trunk surface condition. In this robot climbing case, one effective way to deal with this problem is utilizing the dynamic adjustment wheel normal force to compensate the longitudinal force.

3. Tyre force modelling

The longitudinal tyre force F_x is often modelled by the relation:

$$F_x = f(\lambda, F_s) = \mu F_s \qquad (8.4)$$

That is, by a function that depends linearly on the thrust force F_s from the stepper lead screw and nonlinearly on the wheel slip:

$$\lambda = \frac{R\omega - v}{R\omega} \qquad (8.5)$$

Differentiating Eqn. (8.5) with respect to time gives the derivative form:

$$\dot{\lambda} = \frac{R\dot{\omega}(1-\lambda) - \dot{v}}{R\omega} \qquad (8.6)$$

From the study of vehicle tyre dynamic characteristics in the automobile industry, the typical relationships between longitudinal slip λ and the friction coefficient are illustrated in Fig. 8.5 [90]. It demonstrated that if the longitudinal slip is small, the relationship between the longitudinal force and slip is approximately linear. But with a further increase of the slip, the longitudinal force reaches a maximum at the certain value of the slip specified by tire-road adhesion and is saturated beyond that point [85]. That means as the increase of the slip, the longitudinal force decreases. In this tree climbing case, the dynamic behavior of the robot is nonlinear due to wheel slip. The robot wheel normal force (thrust force from stepper) as well as the trunk coefficient of friction strongly affect the tire longitudinal force's behavior.

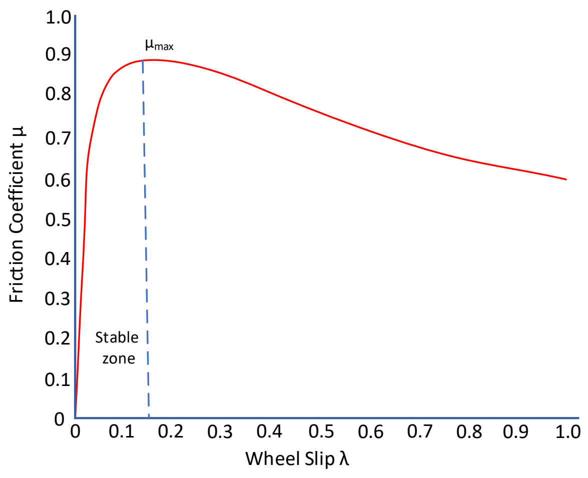


Figure. 8.5 Typical relation between friction coefficient and longitudinal wheel slip [109]

To climb up the trunk fast and stably, the robot three-wheel longitudinal slip should be controlled at the optimal level within the stable zone [84]. Meanwhile, the robot wheel can obtain the maximum longitudinal tractive force F_x with this optimal wheel slip value according to Figure. 8.5.

However, taking the investment of sensors and measurement accuracy into consideration, for many wheeled mobile or climbing robots, it is difficult to acquire the wheel slip directly in real time. To tackle this issue, two types of wheel slip estimation methods are adopted in this paper: off-line and on-line wheel slip estimation. This off-line approach utilized the experimental data from the physical robot climbing test to estimate the robot wheel slip. This method can be adopted as a reference for the online wheel slip estimation.

In general, when the robot climbs up the trunk, all three wheels don't begin to slip at the same time. If one wheel starts to slip, its rotation velocity is different from the setting value (12 RPM). The rotation velocities of each wheel are derived from the encoder of each DC motor. Therefore, comparing with velocity error e_i of each wheel, the system can estimate wheel slippage [82]. In this paper, this type of wheel slip estimation method is defined as on-line wheel slip estimation since the robot can realize the wheel slip in real time during the climbing process.

$$e_i = \left|\omega_{ref} - \omega_i\right|,\ i = 1, 2, 3 \qquad (8.7)$$

Where ω_{ref} is robot wheel setting angular velocity (12 RPM) and ω_i is actual rotation velocity.

Besides, for this tree climbing robot, the tilting angle of the robot platform (from IMU sensor) can be used as an alternate reference for this online wheel slip estimation method.

4. Stepper modelling

As previously discussed, from the robot vertical climbing tests [76], it can be found that owing to the diverse and complicated tree surface, the robot suffers wheel slip. To solve this issue and limit the slip in its optimally stable zone without extra sensors, one stepper and lead screw-nut unit are implemented. Another function of this stepper and screw-nut unit is to prevent the robot from falling to the ground which is defined as active anti-falling mechanism in [76]. As shown in Fig. 8.1, when the stepper motor moves towards the trunk, the corresponding robot wheel normal force increases. Consequently, this influences the robot wheel longitudinal force and wheel slip. To study the nonlinear relationship between stepper steps and wheel longitudinal force, the stepper model is given as follows.

a) STEPPER LEAD SCREW FORCE ANALYSIS

When the stepper moves to the trunk, to illustrate the stepper model clearly and easily, the forces act on the lead screw can be calculated by "unwrapping" one revolution of a thread which is shown in Fig. 8.6. One edge of the thread forms the hypotenuse of a right triangle while its base is the circumference of the thread diameter circle and the height is the lead. α is the lead angle of the thread. F_{lead} is the force to actuate the lead screw while f_s is the thread friction force coefficient.

$$\sum F_x = F_{lead} - N_s sin\alpha - f_s N_s cos\alpha = 0 \qquad (8.8)$$

$$\sum F_y = -F_s - f_s N_s sin\alpha + N_s cos\alpha = 0 \qquad (8.9)$$

According to Eqn. (8.9) and (8.10),

$$F_{lead} = \frac{F_s(sin\alpha + f_s cos\alpha)}{cos\alpha - f_s sin\alpha} \qquad (8.10)$$

Dividing the numerator and denominator of Eqn. (8.10) by $cos\alpha$ and according to equation $tan\alpha = l/\pi d$, it can get:

$$F_{lead} = \frac{F_s(l/\pi d + f_s)}{1 - (f_s l/\pi d)} \qquad (8.11)$$

Hence, to overcome the axially dynamical thrust F_s , the correspondingly required actuate stepper torque is:

$$T_s = \frac{F_s d}{2}\left(\frac{l+\pi f_s d}{\pi d - f_s l}\right) \qquad (8.12)$$

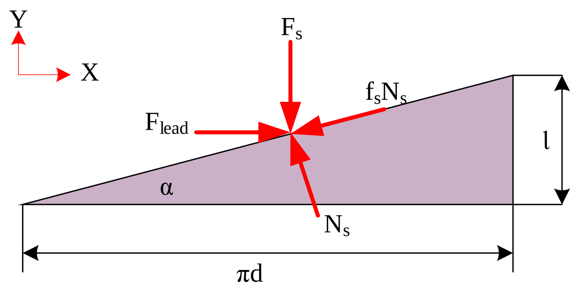


Figure. 8.6 Force analysis of stepper

b) STEPPER LEAD SCREW DYNAMIC ANALYSIS

When the stepper lead screw rotates, the nut connected with robot wheel moves. Through the screw-nut unit, the rotational motion is converted into the translational motion which is illustrated in Fig. 8.7. During this motion conversion process, according to conservation of energy, the rotational energy E_r is converted to translational energy E_t with efficient η :

$$E_r\eta = E_t \qquad (8.13)$$

$$F_s v_s = T_s\omega_s\eta \qquad (8.14)$$

If this process happens in time $\varDelta t$, it can get:

$$F_s v_s \varDelta t = T_s\omega_s\eta\varDelta t \qquad (8.15)$$

$$F_s D = T_s\theta\eta \qquad (8.16)$$

Since the stepper screw lead is l (mm/rev), then $L = {}^{l}/_{2\pi}$ (mm/rad). Hence, the transform from rotation motion to linear motion, it can get:

$$D = L\theta;\ \theta = n\delta \qquad (8.17)$$

Where D is stepper nut translational distance, θ is lead screw rotation angle, n is stepper steps and δ is stepper step angles.

$$F_s D = T_s n\delta\eta \qquad (8.18)$$

$$F_s \leq \frac{T_{smax} n\delta\eta}{D} \qquad (8.19)$$

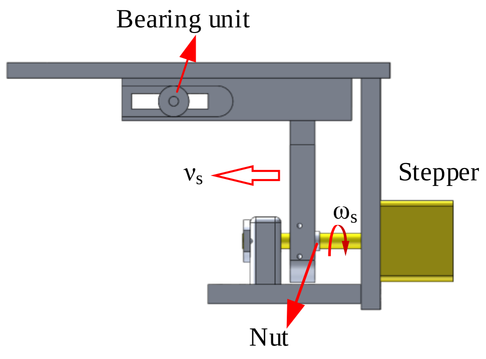


Figure. 8.7 Stepper dynamic model

During the lead screw rotating process, the stepper thrust F_s gradually increase since the distance between the wheel and trunk is reducing and the corresponding pressure is increasing. According to Eqn. (8.19), the maximal thrust F_{smax}that the stepper can offer is the value when the stepper activate torque reaches its maximum T_{smax} with step n_{max}.

Therefore, the wheel longitudinal force F_x can be rewritten as:

$$F_x = f(\lambda, F_s) = f(\lambda, g(n)) \qquad (8.20)$$

That means the wheel longitudinal force F_x can be expressed by stepper step n instead of stepper thrust F_s.

8.3.2 Control problem statement

Based on this tree climbing robot dynamic model, the following control objectives should be considered in the slip controller design.

1. To ensure the robot climb up the trunk fast and stably, according to the analysis of wheel longitudinal force F_x, the wheel longitudinal slip should be controlled at the optimal level within the stable zone based on the stepper steps.
2. Robot three wheels' speed errors between the actual speed and setting value should be in the desired range.
3. Under the premise of safety, using higher climbing speed with low power consumption.

In order to fulfil these control objectives, firstly, the wheel slip is calculated using off-line method according to the robot physical climbing data [76]. The three wheels' slip are shown in Fig. 8.8, Fig. 8.9 and Fig. 8.10, respectively. From time 3, the robot's climbing speed reaches its setting value (12 rpm). It can be seen from these three figures the robot wheels suffer slippage (λ>0.2) during the climbing process when the stepper step is set as n. The highest slip is around 52% which occurred on wheel 3.

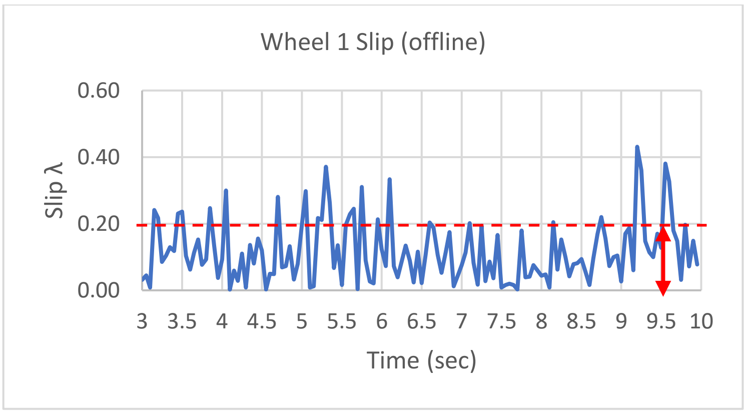


Figure. 8.8 Robot wheel 1 slip

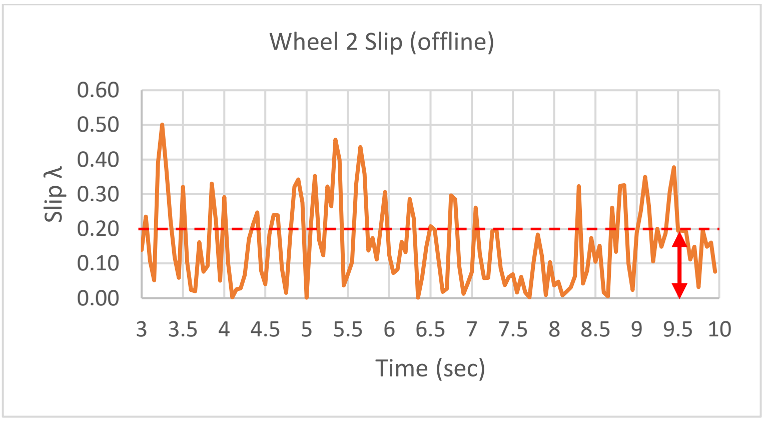


Figure. 8.9 Robot wheel 2 slip

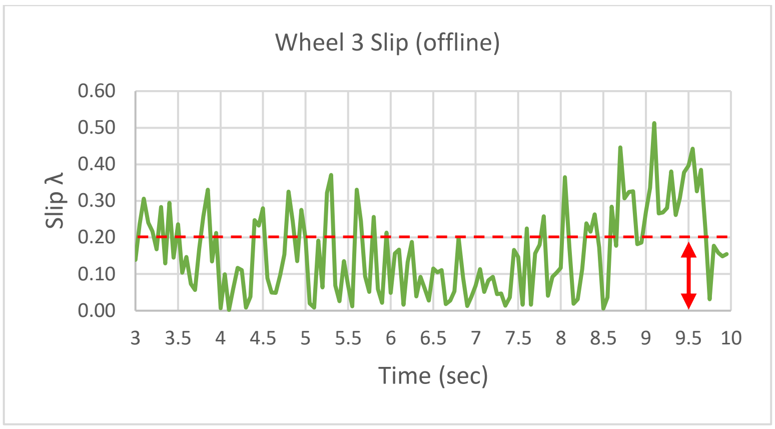


Figure. 8.10 Robot wheel 3 slip

Fig. 8.11 displays the three robot wheel speed errors between the actual wheel climbing speed and setting value (12 RPM) during the climbing process with a constant stepper step $\boldsymbol{n}$. It illustrates the wheel velocity changing level owing to the wheel slip when robot reaches its setting speed from time 3. As previously discussed, this wheel speed error can be used as on-line wheel slip estimation. Comparing Fig.8.11 with Fig. 8.8, Fig. 8.9 and Fig. 8.10, it can be found that the online wheel slip estimation almost matches the offline wheel slip estimation which is derived from the robot wheel velocity errors. The correlation between the offline and online wheel slip estimation is displayed in Table 8.2 which demonstrates that the online wheel slip estimation method (wheel speed errors) can reflect the real wheel slip in real time using the wheel speed errors during the climbing process.

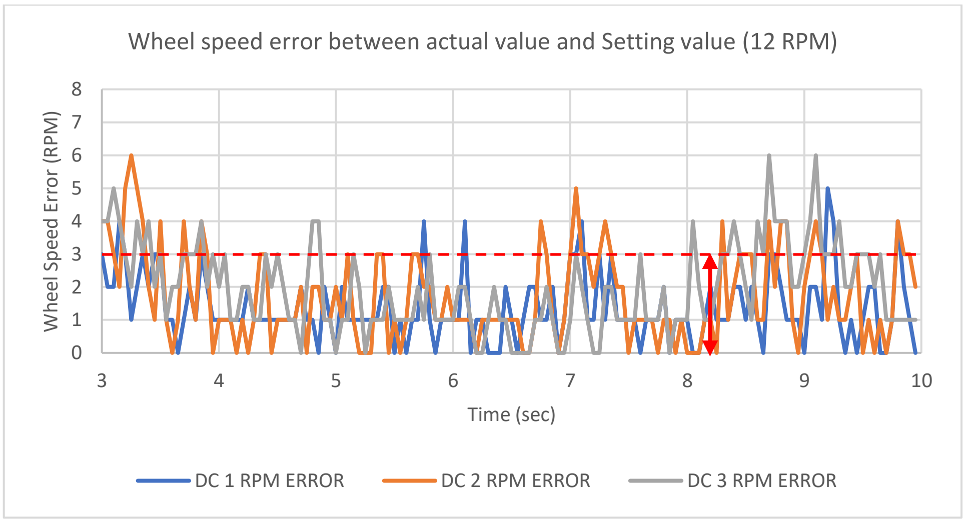


Figure. 8.11 Robot wheel speed error between the actual value and setting value (12 RPM)

Table 8.2 Correlation between offline and online wheel slip

	Wheel slip λ_1VS. Wheel speed e_1	Wheel slip λ_2VS. Wheel speed e_2	Wheel slip λ_3VS. Wheel speed e_3
R^2	0.73	0.659	0.896
RMSE	1.02	1.17	1.05

8.4 Dynamic neural network controller design

Due to the high nonlinearity and complexity between the robot wheel slip and wheel longitudinal force (or stepper steps) during the climbing process, wheel slip control creates a need for advanced control methods for better controlling. Hence, an intelligent dynamic adjusting of the stepper steps VS. robot wheel slip change by dynamic neural networks is proposed. This provided an intelligent control of stepper's steps in terms of providing the maximum climbing effectiveness and maintaining the robot wheel slip within the optimal level. Dynamic neural networks have demonstrated to be an effective method for solving complex nonlinear functions and time series issues[19] [91].

The control scheme of the dynamic neural networks in the robot wheel slip control is shown in Figure. 8.12. Considering the computational capacity of the robot MCU, the simple supervised control and direct inverse control are implied in this wheel slip control and combined in a feedback loop. The main components of the intelligent control configuration are the dynamic inverse neural network model and the dynamic neural network controller, shown in Figure. 8.12. For this research project, the dynamic artificial neural network (ANN) controller was designed for intelligent adjusting the stepper steps to achieve desired level of robot wheel slip. Its neural network has the same structure as the inverse dynamic neural model which enables a transfer of the weight matrix from the inverse dynamic ANN model to the ANN controller at every time step. This enables the ANN controller to deal with a dynamic change of desired wheel slip and to perform good and stable predictions. The dynamic ANN model was designed to approximate the robot wheel slip in real time through the wheel speed errors during robot climbing. It runs faster than the dynamic ANN controller that ensures the

[19] MATLAB Version R2015b, Help: "Neural Network Time Series Prediction and Modelling".

achievement of well-adjusted weight coefficients. Besides, the ANN model can also update its own weight coefficients online using the error signal. That means if a large disturbance or uncertainty occurs in the climbing process, the large error signal is fed back to the ANN model to adjust its weight coefficients in order to remain the system stable [97].

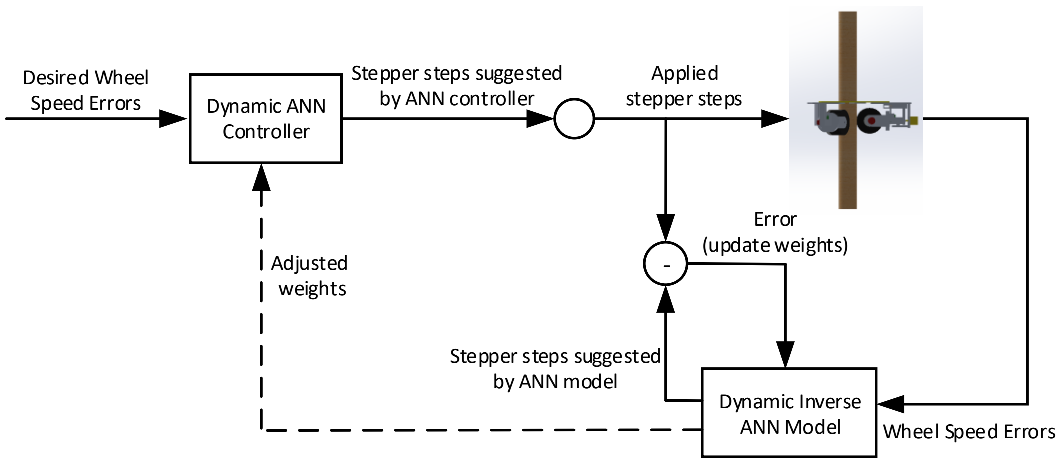

Desired Wheel Speed Errors
Dynamic ANN Controller
Stepper steps suggested by ANN controller
Applied stepper steps
Error (update weights)
-
Adjusted weights
Stepper steps suggested by ANN model
Dynamic Inverse ANN Model
Wheel Speed Errors

Figure. 8.12 Dynamic neural network control scheme of climbing robot longitudinal wheel slip

8.5 Tree pruning robot wheel slip control experiment and result

In this paper, the time series NARX (nonlinear autoregressive network with exogenous inputs) feedback neural network [91] is chosen as the ANN controller. The NARX is a recurrent dynamic network with feedback connections enclosing several layers of the network. Its model is based on the linear ARX model which is commonly used in time series modelling. The defining equation for the NARX model is:

$$\boldsymbol{y}(\boldsymbol{t}) = \boldsymbol{f}(\boldsymbol{y}(\boldsymbol{t}-\mathbf{1}), \boldsymbol{y}(\boldsymbol{t}-\mathbf{2}), \ldots, \boldsymbol{y}(\boldsymbol{t}-\boldsymbol{n_y}), \boldsymbol{u}(\boldsymbol{t}-\mathbf{1}), \boldsymbol{u}(\boldsymbol{t}-\mathbf{2}), \ldots, \boldsymbol{u}(\boldsymbol{t}-\boldsymbol{n_u})) \quad (8.21)$$

Where the next value of the dependent output signal $y(t)$ is regressed on previous values of the output signal and previous values of an independent (exogenous) input signal[20].

In this research, the main objective was to keep the robot wheel slip at the optimal level to obtain the maximum longitudinal wheel force according to the wheel speed error

[20] MATLAB Version R2015b, Help: “Design Time Series NARX Feedback Neural Networks”.

during climbing. Firstly, the NARX neural network of the controller is trained using the physical robot climbing data through MATLAB ANN tool box. The networks structure is displayed in Fig. 8.13 which includes one input layer, one hidden layer and one output layer. The hidden layer has ten neurons and the transfer functions are the “tansig” and “purelin” between network layers. The Levenberg-Marquardt learning algorithm was used in the neural networks training process.

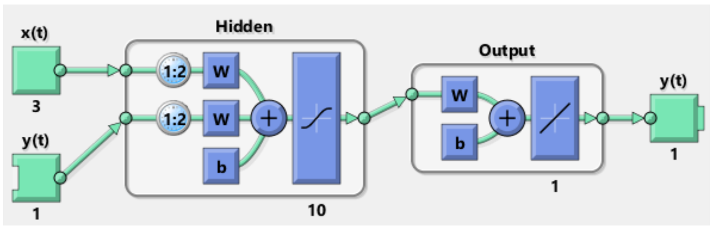


Figure. 8.13 NARX neural network in MATLAB

The training results are shown from Figure.8.14 to 8.17, which represent the neural network training performance, training regression, training time series response, and training input-error cross correlation.

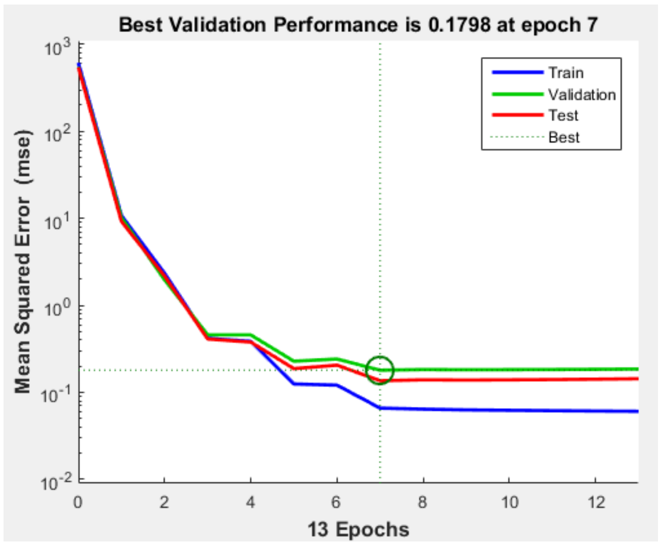


Figure. 8.14 Neural network training performance

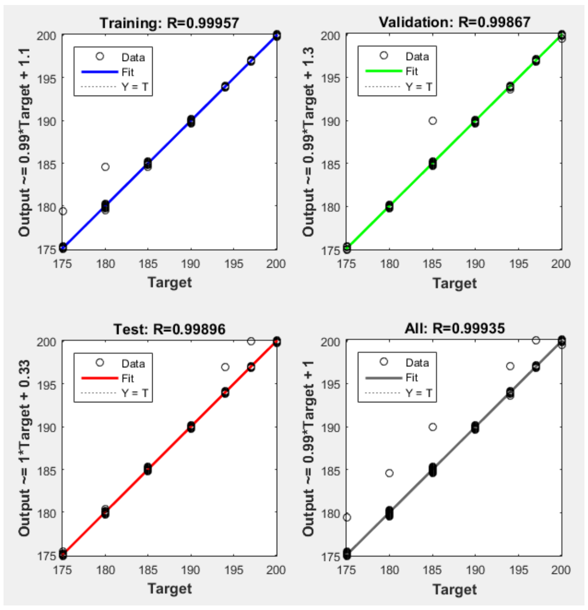


Figure. 8.15 Neural network training regression

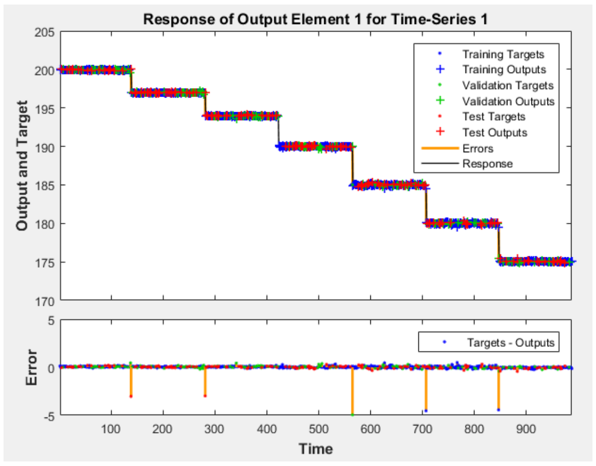


Figure. 8.16 Neural network training time series response

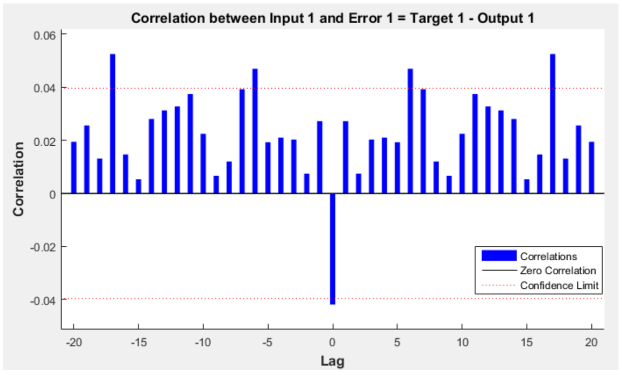


Figure. 8.17 Neural network training input-error cross correlation

For the design of inverse dynamic ANN model, it is based on dynamic feedforward time delay neural network with one hidden layer containing ten neurons.

After the training, the wheel slip dynamic neural network control scheme is applied on the physical robot climbing test on a rod which is shown in Figure. 8.18. The length of the test rod is 1.6 m and the diameter is 122.5 mm. The total mass of the tree pruning robot is around 6.8 Kg [95].

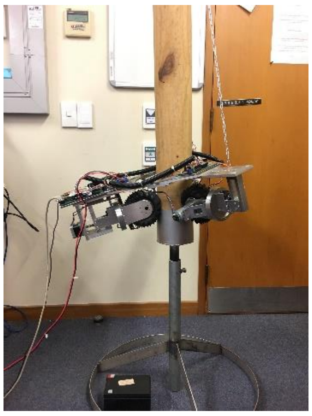


Figure. 8.18 Robot wheel slip control test on a rod

After the implementation of the dynamic neural networks control scheme in the robot physical climbing test, the robot three wheels' slip is displayed in Figure. 8.19, 8.20 and 8.21, which is based on the experimental climbing data. Figure. 8.22 illustrates the robot three wheels' speed errors during the test.

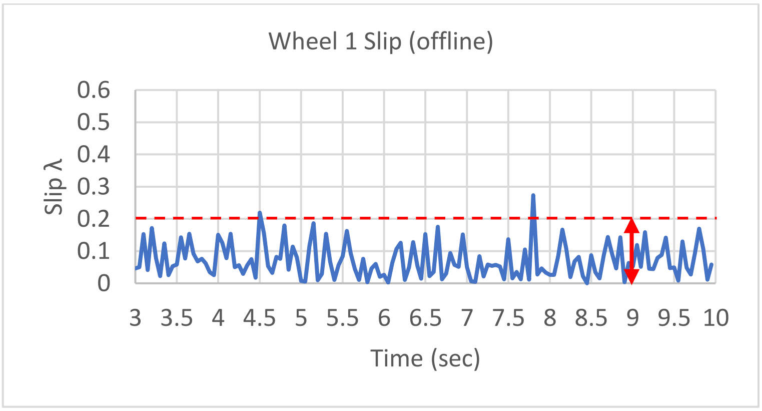


Figure. 8.19 Robot wheel 1 slip after the implementation of dynamic neural network control

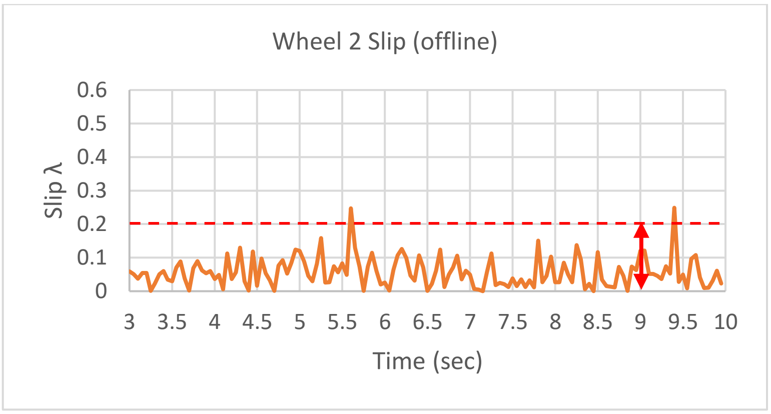


Figure. 8.20 Robot wheel 2 slip after the implementation of dynamic neural network control

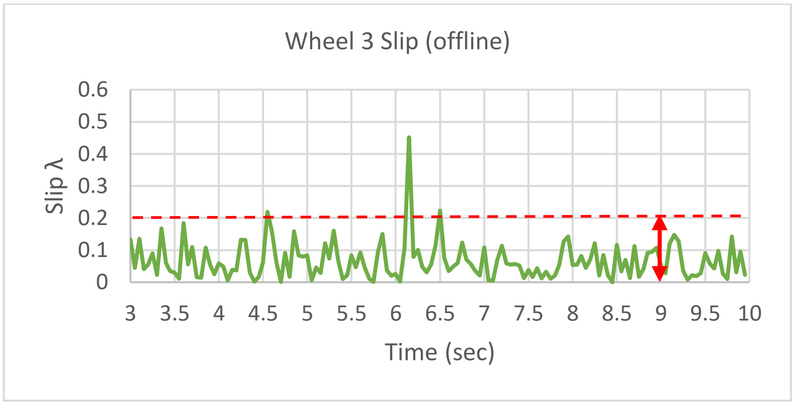


Figure. 8.21 Robot wheel 3 slip after the implementation of dynamic neural network control

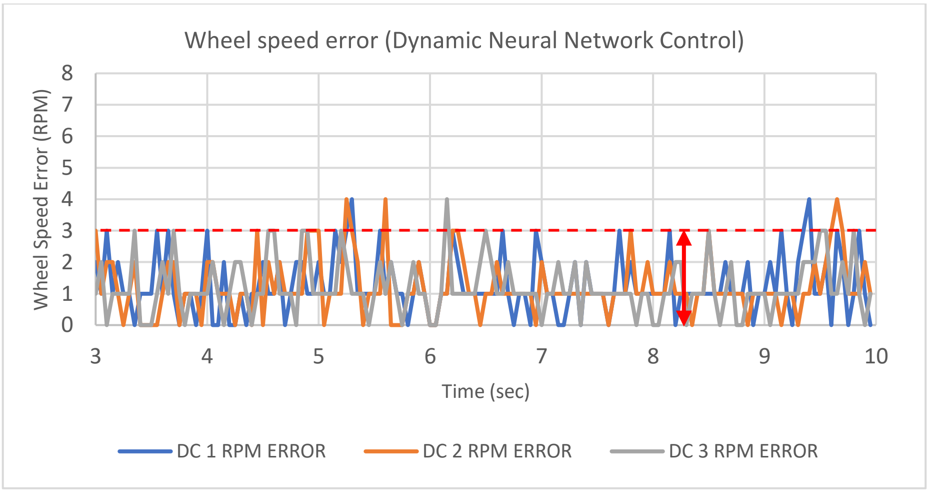


Figure. 8.22 Robot wheel speed error during the test (dynamic neural network control)

Comparing Fig. 8.8-8.10 with Fig. 8.19-8.21, it can be seen that the robot wheel slip has improved significantly after the implementation of the dynamic neural network control system. The wheel slip is controlled within the stable zone (λ <0.2) during the climbing process. This is also can be reflected in Figure. 8.22 that illustrated the robot wheel speed errors in the climbing test. From time 3, the robot reaches the setting speed (12 RPM) and most of the wheel errors fluctuate under the error 3. Table 8.3 shows the percentage of robot wheel slip outside the stable zone between robot climbing using dynamic neural network control (DNNC) and without it.

Table 8.3 Wheel slip outside the stable zone between robot climbing using DNNC and without DNNC

Percentage (%)	Wheel slip λ_1	Wheel slip λ_2	Wheel slip λ_3
Robot climbing without DNNC	17.86	30.71	35
Robot climbing with DNNC	1.43	1.43	2.14

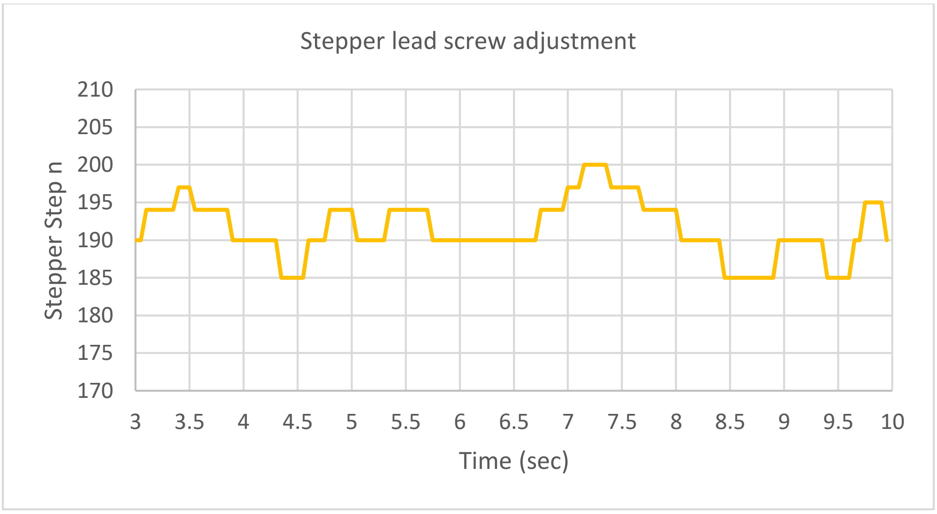


Figure. 8.23 Stepper lead screw adjustment based on dynamic neural network control

Figure. 8.23 shows the robot stepper steps adjustment according to the dynamic neural network control system during the climbing process. In this test, the stepper motor has a 1.8° step angle (200 steps/revolution) and the nut moves 8 mm per full revolution, which allows for a linear resolution of 0.04 mm per step. To relieve the MCU computation burden and reduce the stepper noise interference, the stepper steps in this research are set as discrete which are 175, 180, 185, 190, 194, 197, 200, respectively.

8.6 Tree pruning robot wheel slip control conclusion

For wheeled tree or pole climbing robot, wheel slip is an inevitable issue during the climbing process due to the unsmooth and irregular tree or pole surface. To solve this complex and nonlinear problem, a wheel slip control system in longitudinal direction based on dynamic neural network is proposed in this chapter. The control strategy is fulfilled through the dynamic adjustment of the stepper's steps based on the longitudinal wheel slip optimal level ($\lambda \leq 0.2$). The wheel speed error is used as the estimation of wheel slip during the climbing and the input of the dynamic neural network controller. The dynamic neural network controller is trained based on the previous experimental climbing data and then used to predict the longitudinal wheel slip according to the wheel speed errors during the physical climbing test. The test result shows that the wheel slip is almost all within the stable zone which verified the effectiveness of this dynamic neural network longitudinal wheel slip control method.

Chapter 9 Conclusions and Future Works

In this research, a novel three-wheeled climbing robotic system is developed, and a testing physical model is built, which is capable of autonomously climbing tree or pole to perform specific tasks like tree pruning. The mechanical structure of this robot is designed in SolidWorks and simulated in MATLAB SimMechanics. The initial first climbing robot prototype weights around 8.76 Kg which equipped with two steppers. However, for the first model, some drawbacks needed to overcome such as heavyweight, robustness and flexibility for various trunk diameter. To solve these issues, an optimized climbing robot system that only relies on one stepper to achieve the anti-falling and anti-jamming functionality was designed and built. The optimized climbing robot weights 6.8 Kg and possesses three other mechanisms: screw-nut unit, servomotor module and bearing support mechanism. To perform the tree pruning work, each robot wheel is installed a servomotor module. The robot uses these modules to change the climbing morphology. Except vertical climbing, the robot can also climb the trunk in a spiral pattern by turning the wheels at given angles. The bearing support mechanism consists of one ball bearing, bearing shaft, bearing linkage and guild rail. Its function is to overcome the robot wheel module gravity force and other disturbing forces that affect the servomotor module and stepper's screw-nut unit. This guarantees the lead screw efficiency and control accuracy of the step motor.

The main feature of this tree climbing robot is that it implements two sets of anti-falling strategies: passive and active anti-falling mechanisms. The passive anti-falling mechanism uses only friction forces and robot gravity force to maintain a hold on the tree trunk. The primary point of achieving this feature is to let the center of the mass of the robot offset from the center of the tree. In terms of active anti-falling mechanism, it is fulfilled through the stepper's screw-nut unit during climbing process. When the stepper moves towards the trunk, the normal forces on the wheel-trunk contact areas increases. So, do the longitudinal wheel tractive forces. With such an active anti-falling mechanism, it guarantees that the robot can climb up the trunk safely and steadily without slip or falling. This function is elaborated in detail in chapter 8 in the form of wheel slip control using dynamic neural networks. The types of driving mechanism of this robot are determined by the outcome of the robot mechanical analysis and the SimMechanics simulation. During the physical climbing test, the average vertical climbing speed is around 48.02 mm/s when the DC motor rotation speed is set at 12 RPM. The trunk diameter that the robot can climb up ranges from 120 to 160 mm. the maximum load that the robot can take is approximately 2.1 kg at the setting speed of 12 RPM for the DC motors.

During the physical climbing test, the robot suffers tilting issue. To solve this problem, first of all, the accurate tilting angle of the robot platform in real time during the climbing process should be obtained. In this robot design, a MEMS based IMU sensor (MPU9150) was installed to acquire the accurate tilting information. Two IMU data fusion methods (complementary filter and Kalman filter) were studied and compared using the robot tilting angles. Finally, the complementary filter was chosen as the IMU data fusion method owing to its less computation complexity and only one filter coefficient to be tuned. In this research, there are two types of tilt required to balance: initial setting tilt and dynamic tilt. The initial setting tilt is caused by the passive anti-falling mechanism because of the center of the gravity offsetting. The robot remains stationary at a certain tilting angle on the trunk utilizing its own weight without energy consumption. This pre-set tilting angle is named as the initial setting tilt. During the climbing process, each of the three DC motors can suffer slippage due to the complicated trunk surface. Therefore, even if the three wheels keep the same speed during climbing procedure, the robot platform could dynamically tilt at a random direction. To solve these two types of tilt, a tilting fuzzy controller and one slippage fuzzy control system were designed and implemented. When the robot employs this fuzzy control system, the pitch and roll angles of the robot platform are limited in the desired range.

In fact, the robot wheel slip issue not only incurs robot platform tilting but also causes some other problems like distance error, more overall energy consumption and worse robot climbing performance. To avoid this, the wheel slip must be controlled in its optimally stable zone during the robot climbing. However, the control of wheel slip is a challenging problem due to the complex and nonlinear dynamics of tire-surface interaction. In this research, a new approach based on dynamic neural networks (DNN) and direct inverse control (DIC) were employed for improving the tree climbing robot longitudinal wheel slip control. This control strategy is fulfilled based on the dynamic adjustment of the stepper's lead screw thrust (or steps) according to the wheel slip stable zone constraints. Robot wheel slip is obtained using online wheel slip estimation method which is derived from wheel speed errors and verified through offline wheel slip estimation. The dynamic neural network has been used for modelling of a nonlinear relationship between the tree climbing robot' stepper lead screw thrust and the longitudinal wheel slip during the vertical climbing procedure. It provided preconditions for control of the stepper lead screw thrust based on the wheel slip change.

9.1 Works for future research

This tree pruning robot research still remains work for further research. These works mainly focus on the following aspects. The first one is the robot hardware design and configuration. For example, during the passive anti-falling test, when the robot remains stationary on the testing rod without power consumption, sometimes the robot wheel rolls back if the servomotor turns the wheel not at a proper angle (DC motor back drive). To eliminate this problem, a worm-gear DC motor could be considered. Furthermore, there are many ways to improve the robot climbing performance such as a further reduction in the weight of the robot by using varied materials or reconstructing the robot platform frame and adding intelligent sensing, adding tree pruning or inspection tools on the robot platform and tested on real trees.

Another work that needs to be done in the future is the wheel slip control. For instance, in this research, the longitudinal wheel slip controller is trained only using the specific rod climbing data, which means it may not work well at other surface conditions like trees. Furthermore, due to the computation capacity of the MCU, the wheel slip is estimated using wheel speed error during the robot climbing. For more accurate wheel slip measurement, in the future the more professional robot speed measurement equipment is needed instead of the speed sensor. And the lateral wheel slip control of the robot wheel during climbing will be investigated and analyzed in the future.

9.2 Main contributions and publications

- In this research, a novel tree climbing robotic system is designed and the motion simulation has also conducted. Its kinematic characteristics have been analyzed. The research outcomes are published in the following journals and conferences.

1. **Pengfei Gui**, Liqiong Tang and Subhas Mukhopadhyay, 2017, "A Novel Robotic Tree Climbing Mechanism with Anti-Falling Functionality for Tree Pruning," Journal of Mechanisms and Robotics, 2018, 10(1), doi:10.1115/1.4038219, **Impact Factor: 2.371**, pp: 014502-1 – 014502-8.
2. **Pengfei Gui**, Liqiong Tang and Subhas Mukhopadhyay, 2015, "A Novel Design of Anti-Falling Mechanism for Tree Pruning Robot," IEEE 10th Conference on Industrial Electronics and Applications, 15-17 June, Auckland, New Zealand, doi: 10.1109/ICIEA.2015.7334222, pp: 812-816.
3. **Pengfei Gui**, Liqiong Tang and Subhas Mukhopadhyay, 2017, "Anti-falling tree climbing mechanism optimization," 2017 2nd Asia-Pacific Conference on Intelligent Robot System, 16-18 June, Wuhan, China, doi: 10.1109/ACIRS.2017.7986109, pp: 284-288.

- During the robot climbing process, it suffers tilting. To solve this problem, firstly the complementary filter and Kalman filter based IMU data fusion methods for tilting measurement are studied and compared in order to obtain the accurate robot tilting angle in real time. Then two fuzzy controllers are designed and applied to solve the initial tilt which is caused by the passive anti-falling mechanism and dynamic tilt.

1. **Pengfei Gui**, Liqiong Tang and Subhas Mukhopadhyay, 2015, "MEMS Based IMU for Tilting Measurement: Comparison of Complementary and Kalman Filter Based Data Fusion," IEEE 10th Conference on Industrial Electronics and Applications, 15-17 June, Auckland, New Zealand, doi: 10.1109/ICIEA.2015.7334442, pp: 2004-2009.
2. **Pengfei Gui**, Liqiong Tang and Subhas Mukhopadhyay, 2017, "Tree Pruning Robot Tilting Control Using Fuzzy Logic," 2017 11th International Conference on

Sensing Technology, 4-6 December, Sydney, Australia, doi: 10.1109/ICSensT.2017.8304509, pp: 1-5.

- To climb the trunk safely and stably, the robot longitudinal wheel slip is studied, and a wheel slip controller is designed based on dynamic neural networks. This control strategy is fulfilled based on the dynamic adjustment of the stepper's lead screw thrust (or steps) according to the wheel slip stable zone constraints.

1. **Pengfei Gui**, Liqiong Tang and Subhas Mukhopadhyay, 2018, "Wheel-Driven Climbing Robot Longitudinal Wheel Slip Control Using Dynamic Neural Networks," submitted and under review, Journal of Dynamic Systems, Measurement and Control, ASME, **Impact Factor: 1.388**.

Appendix A – Program Codes of Tree Pruning Robot

Tree Pruning Robot Tilting Fuzzy Control

Arduino Mega:

```
#include <Servo.h>
#include<AccelStepper.h>


/*****************************Serial Communication*****************/
const char startOfNumberDelimiter = '<';
const char endOfNumberDelimiter   = '>';
volatile int t = 0;

/**************************DC Motor variables definition*********/

#define U2A1          8                    // U2INA motor pin
#define U2B1          9                    // U2INB motor pin
#define U3A1          10                   // U3INA motor pin
#define U3B1          11                   // U3INB motor pin
#define U4A1          12                   // U4INA motor pin
#define U4B1          13                   // U4INB motor pin
#define U2PWM          5                   // U2 PWM motor pin
#define U3PWM          6                   // U3 PWM motor pin
#define U4PWM          7                   // U4 PWM motor pin
#define U2encodA     2                     //U2 encoder A pin
#define U2encodB     14                    //U2 encoder B pin
#define U3encodA     3                     //U3 encoder A pin
#define U3encodB     15                    //U3 encoder B pin
#define U4encodA     21                    //U4 encoder A pin
#define U4encodB     16                    //U4 encoder B pin

/************************Servo Motor Definition********************/
Servo servo1;
Servo servo2;
Servo servo3;
int pos1 = 90;
int pos2 = 95;
int pos3 = 105;

/************************System Running Time Control********************/
int sys_run_timer = 0;

/****************************** Stepper Setup ***********************************/

AccelStepper stepper1(1,24,25);
int stepperPin = 24;
int dirPin = 25;
boolean step_dir;
int step_count = 0;
int step_mid = 0;

/************************power switch******************************/

//#define Vpin          0                    // battery monitoring analog pin
#define U2Apin          0                  // U2 motor current monitoring analog pin
#define U3Apin          1                  // U3 motor current monitoring analog pin
#define U4Apin          2                  // U4 motor current monitoring analog pin
#define Apin            3                  // System current
#define PowerOn          22                  // Power Switch On Pin, High switch on; Low switch off
#define CURRENT_LIMIT   20000                  // high current warning
//#define LOW_BAT         10000                  // low bat warning


/************************PID loop Time********************/
#define LOOPTIME       60                   // PID loop time

/************************Current Filter********************/
#define NUMREADINGS     10                   // samples for Amp average
long readings[NUMREADINGS];
long readings_2[NUMREADINGS];
long readings_3[NUMREADINGS];
long readings_4[NUMREADINGS];


/************************System Loop Time********************/
unsigned long lastMilli = 0;                  // loop timing
unsigned long lastMilliPrint = 0;              // loop timing

/************************3 DC Motor Definition********************/
//int speed_req = 12;                          // speed (Set Point)
int DC1,DC2,DC3;                               // speed (from fuzzy control)
int speed_act2 = 0;                            // speed (actual value)
int speed_act3 = 0;                            // speed (actual value)
int speed_act4 = 0;                            // speed (actual value)
int PWM_val2 = 0;                              // (25% = 64; 50% = 127; 75% = 191; 100% = 255)   Arduino Due: (25% = 1024; 50% = 2048; 75% = 3072; 100% = 4096)
int PWM_val3 = 0;                              // (25% = 64; 50% = 127; 75% = 191; 100% = 255)   Arduino Due: (25% = 1024; 50% = 2048; 75% = 3072; 100% = 4096)
int PWM_val4 = 0;                              // (25% = 64; 50% = 127; 75% = 191; 100% = 255)   Arduino Due: (25% = 1024; 50% = 2048; 75% = 3072; 100% = 4096)
//int voltage = 0;                             // in mV
long current2 = 0;                             // in mA
volatile long count2 = 0;                      // rev counter
```

```
long current3 = 0;                              // in mA
volatile long count3 = 0;                         // rev counter
long current4 = 0;                              // in mA
volatile long count4 = 0;                         // rev counter


/************************Whole System Current********************/
long current = 0;


/************************PID Control papameter********************/
float Kp =   .4;                              // PID proportional control Gain
float Kd =  1.0;                              // PID Derivitave control gain

/****************************** system parameter setup *****************************************/

void setup() {
  //analogReference(EXTERNAL);                                                                    // Current external ref is 3.3V
  Serial.begin(115200);
  Serial1.begin(115200);
  while(!Serial){};
  while(!Serial1){};


/************************initialize readings to 0********************/
  for(int i=0; i<NUMREADINGS; i++)   readings[i] = 0;
  for(int i=0; i<NUMREADINGS; i++)   readings_2[i] = 0;
  for(int i=0; i<NUMREADINGS; i++)   readings_3[i] = 0;
  for(int i=0; i<NUMREADINGS; i++)   readings_4[i] = 0;


/************************DC motor********************/
  pinMode(U2A1, OUTPUT);
  pinMode(U2B1, OUTPUT);
  pinMode(U3A1, OUTPUT);
  pinMode(U3B1, OUTPUT);
  pinMode(U4A1, OUTPUT);
  pinMode(U4B1, OUTPUT);
  pinMode(U2PWM, OUTPUT);
  pinMode(U3PWM, OUTPUT);
  pinMode(U4PWM, OUTPUT);
  pinMode(PowerOn, OUTPUT);
  pinMode(U2encodA, INPUT_PULLUP);
  pinMode(U2encodB, INPUT_PULLUP);
  pinMode(U3encodA, INPUT_PULLUP);
  pinMode(U3encodB, INPUT_PULLUP);
  pinMode(U4encodA, INPUT_PULLUP);
  pinMode(U4encodB, INPUT_PULLUP);
//  digitalWrite(U2encodA, HIGH);                                                                 // turn on pullup resistor
//  digitalWrite(U2encodB, HIGH);
//  digitalWrite(U3encodA, HIGH);                                                                 // turn on pullup resistor
//  digitalWrite(U3encodB, HIGH);
//  digitalWrite(U4encodA, HIGH);                                                                 // turn on pullup resistor
//  digitalWrite(U4encodB, HIGH);
  attachInterrupt(0, rencoder2, FALLING);
  attachInterrupt(1, rencoder3, FALLING);
  attachInterrupt(2, rencoder4, FALLING);
  delay(2);

  analogWrite(U2PWM, PWM_val2);
  digitalWrite(U2A1, HIGH);
  digitalWrite(U2B1, LOW);
  analogWrite(U3PWM, PWM_val3);
  digitalWrite(U3A1, HIGH);
  digitalWrite(U3B1, LOW);
  analogWrite(U4PWM, PWM_val4);
  digitalWrite(U4A1, HIGH);
  digitalWrite(U4B1, LOW);


/************************Servo motor********************/
  servo1.attach(44);
  delay(1);
  servo2.attach(45);
  delay(1);
  servo3.attach(46);
  delay(10);

  servo_dir_90();
  //servo1.write(pos1);
  //delay(2);
  //servo2.write(pos2);
  //delay(2);
  //servo3.write(pos3);
  //delay(2);


/************************Serial Display********************/
    Serial.print("DC1: ");
    Serial.print("  RPM2: ");
    Serial.print("  U2PWM: ");
    Serial.print("  DC2: ");
    Serial.print("  RPM3: ");
    Serial.print("  U3PWM: ");
    Serial.print("  DC3: ");
    Serial.print("  RPM4: ");
    Serial.print("  U4PWM: ");
    Serial.print("  2CS: ");
    Serial.print("  3CS: ");
    Serial.print("  4CS: ");
    Serial.print("  3 DC current:  ");
    Serial.println("     total current:   ");


/************************Step motor********************/
  stepper1.setMaxSpeed(2000);
  stepper1.setAcceleration(800);
  pinMode(dirPin,OUTPUT);
  pinMode(stepperPin, OUTPUT);
  stepper1.moveTo(-400);

  delay(2000);

  lastMilli = millis();
}



/*********************************** main loop ***********************************/

void loop() {

  //stepper_climb_init();

    getParam();                                              //check keyboard

    getFuzzy();                                              //DC1, DC2, DC3

    DC_motor_pid();                                            //DC motor pid control

    printMotorInfo();                                          //display data



   if(sys_run_timer != 250)
    servo_dir_90();



    if(sys_run_timer == 250)
    {

      anti_falling_DC_servo_stepper();
      digitalWrite(PowerOn, LOW);
      sys_run_timer = 0;
      Serial.println("*************************** System Time Run Out *************************** ");
      //Serial.println();
     }

}


void stepper_climb_init()   {

   while(abs(stepper1.distanceToGo()) > 0)
   {
     stepper1.run();
    }

}


/************************ get DC1 DC2 DC3 from Arduino Due *********************************************/
void getFuzzy() {


  if(Serial1.available())
  processInput();

 }


void processInput ()
 {
 static long receivedNumber = 0;
 static boolean negative = false;

 byte c = Serial1.read ();

 switch (c)
   {

   case endOfNumberDelimiter:
     if (negative)
       processNumber (- receivedNumber);
     else
       processNumber (receivedNumber);

   // fall through to start a new number
   case startOfNumberDelimiter:
     receivedNumber = 0;
     negative = false;
     break;

   case '0' ... '9':
     receivedNumber *= 10;
     receivedNumber += c - '0';
     break;

   case '-':
     negative = true;
     break;
   } // end of switch
 }  // end of processInput


void processNumber (const long n)
 {

  switch(t){

case 0:
  DC1 = n;
  //Serial.print(n);
  //Serial.print('\t');
  t++;
  break;

case 1:
  DC2 = n;
  //Serial.print(n);
  //Serial.print('\t');
  t++;
  break;

case 2:
  DC3 = n;
  //Serial.println(n);
  //Serial.print('\t');
  //Serial.print('\t');
  t = 0;
  break;
/*
case 3:
  Serial.println(n);
  i = 0;
  break;
*/
}

 }  // end of processNumber


/************************************ DC motor pid control ****************************************************/
void DC_motor_pid(){

    if((millis()-lastMilli) >= LOOPTIME)
  {                                                                                         // enter tmed loop
    getMotorData();                                                                             // calculate speed, volts and Amps
    PWM_val2 = updatePid2(PWM_val2, DC1, speed_act2);                                               // compute PWM value
    PWM_val3 = updatePid3(PWM_val3, DC2, speed_act3);                                               // compute PWM value
    PWM_val4 = updatePid4(PWM_val4, DC3, speed_act4);                                               // compute PWM value
    analogWrite(U2PWM, PWM_val2);                                                              // send PWM to motor
    analogWrite(U3PWM, PWM_val3);                                                              // send PWM to motor
    analogWrite(U4PWM, PWM_val4);                                                              // send PWM to motor
    sys_run_timer = sys_run_timer + 1;
    lastMilli = millis();
  }

   }


/************************************** servo motor control ********************************************/
void servo_dir_90()   {

    servo1.write(pos1);
    servo2.write(pos2);
    servo3.write(pos3);
}


/************************** stop DC motor and rotate servo motor ****************************************/
void anti_falling_DC_servo_stepper()  {

    digitalWrite(U2A1,LOW);
    digitalWrite(U2B1,LOW);
    digitalWrite(U3A1,LOW);
    digitalWrite(U3B1,LOW);
    digitalWrite(U4A1,LOW);
    digitalWrite(U4B1,LOW);
    for(int i = 0; i < 50; i++)
    {
    servo1.write(30);
    servo2.write(170);
    servo3.write(170);
    delay(1);
    //stepper1.moveTo(-200);
    //while(stepper1.distanceToGo() != 0)
    // stepper1.run();
    }

}


/********************************** get DC motor data ************************************************/
void getMotorData()  {                                                                          // calculate speed, volts and Amps
static long countAnt2 = 0;                                                                       // U2 last count
static long countAnt3 = 0;                                                                       // U3 last count
static long countAnt4 = 0;                                                                       // U4 last count

  speed_act2 = int(((count2 - countAnt2)*(60.0*(1000.0/LOOPTIME)))/(16*102.083));                                  // 16 pulses X 102 gear ratio = 1632 counts per output shaft rev
mei fei zhong zhuan de quan shu
```

```
  countAnt2 = count2;
  current2 = int(analogRead(U2Apin) * 4.882 / 130.00 * 1000.00);                              // motor current - output: 130mV per Amp
  current2 = digital_smooth_2(current2, readings_2);                                          // remove signal noise

  speed_act3 = int(((count3 - countAnt3)*(60.0*(1000.0/LOOPTIME)))/(16*102.083));                          // 16 pulses X 102 gear ratio = 1632 counts per output shaft rev
mei fei zhong zhuan de quan shu
  countAnt3 = count3;
  current3 = int(analogRead(U3Apin) * 4.882 / 130.00 * 1000.00);                              // motor current - output: 130mV per Amp
  current3 = digital_smooth_3(current3, readings_3);                                          // remove signal noise

  speed_act4 = int(((count4 - countAnt4)*(60.0*(1000.0/LOOPTIME)))/(16*102.083));                          // 16 pulses X 102 gear ratio = 1632 counts per output shaft rev
mei fei zhong zhuan de quan shu
  countAnt4 = count4;
  current4 = int(analogRead(U4Apin) * 4.882 / 130.00 * 1000.00);                              // motor current - output: 130mV per Amp
  current4 = digital_smooth_4(current4, readings_4);                                          // remove signal noise

  current = int((analogRead(Apin) * 4.882 - 2500.0) / 136.0 * 1000.0);
  current = digital_smooth(current, readings);

}


/************************************ DC motor1 pid *******************************************/
int updatePid2(int command2, int targetValue2, int currentValue2)   {                         // compute PWM value
float pidTerm2 = 0.0;                                                                          // PID correction
int error2=0;
static int last_error2=0;
  error2 = abs(targetValue2) - abs(currentValue2);
  pidTerm2 = (Kp * error2) + (Kd * (error2 - last_error2));
  last_error2 = error2;
  return constrain(command2 + int(pidTerm2), 0, 255);
}


/********************************************* DC motor2 pid ************************************/
int updatePid3(int command3, int targetValue3, int currentValue3)   {                         // compute PWM value
float pidTerm3 = 0.0;                                                                          // PID correction
int error3=0;
static int last_error3=0;
  error3 = abs(targetValue3) - abs(currentValue3);
  pidTerm3 = (Kp * error3) + (Kd * (error3 - last_error3));
  last_error3 = error3;
  return constrain(command3 + int(pidTerm3), 0, 255);
}


/********************************* DC motor3 pid **********************************************/
int updatePid4(int command4, int targetValue4, int currentValue4)   {                         // compute PWM value
float pidTerm4 = 0.0;                                                                          // PID correction
int error4=0;
static int last_error4=0;
  error4 = abs(targetValue4) - abs(currentValue4);
  pidTerm4 = (Kp * error4) + (Kd * (error4 - last_error4));
  last_error4 = error4;
  return constrain(command4 + int(pidTerm4), 0, 255);
}

/********************************* Serial print Motor info ****************************************/
void printMotorInfo()  {                                                                       // display data
  if((millis()-lastMilliPrint) >= 300)
  {
        lastMilliPrint = millis();
        Serial.print(DC1);              Serial.print('\t');  // Serial.print('\t');
        Serial.print(speed_act2);          Serial.print('\t');   //Serial.print('\t');
        Serial.print(PWM_val2);             Serial.print('\t');   //Serial.print('\t');
        Serial.print(DC2);              Serial.print('\t');  // Serial.print('\t');
        Serial.print(speed_act3);          Serial.print('\t');   //Serial.print('\t');
        Serial.print(PWM_val3);             Serial.print('\t');   //Serial.print('\t');
        Serial.print(DC3);              Serial.print('\t');  // Serial.print('\t');
        Serial.print(speed_act4);          Serial.print('\t');  // Serial.print('\t');
        Serial.print(PWM_val4);             Serial.print('\t');   //Serial.print('\t');
        Serial.print(current2);          Serial.print('\t');  // Serial.print('\t');
        Serial.print(current3);          Serial.print('\t');  // Serial.print('\t');
        Serial.print(current4);          Serial.print('\t');
        Serial.print(current2 + current3 + current4);   Serial.print('\t');     Serial.print('\t');   //Serial.print('\t');
        Serial.println(current);
  //  Serial.print("   Stepper_Pos:  ");       Serial.println(step_count*0.04);
    if ((current2 + current3 + current4) > CURRENT_LIMIT)
    {
      Serial.println("********** 3 DC Motors Exceed CURRENT_LIMIT ***********");
      Serial.println();
      digitalWrite(PowerOn, LOW);
    }
    if ( current > 14000)
    {
      Serial.println("********** System Current Limit *********");
      Serial.println();
      digitalWrite(PowerOn, LOW);
    }
    //if (voltage > 1000 && voltage < LOW_BAT)   Serial.println("*** LOW_BAT ***");

  }

}


/************************************ interrupt 1, 2 3 **************************************************/
void rencoder2()  {                                                                            // pulse and direction, direct port reading to save cycles
  //if(PINJ & 0b00000010)        count2 ++;
  if (digitalRead(U2encodB) == HIGH)    count2++;                                              // if(digitalRead(encodPinB1)==HIGH)   count ++;
  else                  count2--;                                                              // if (digitalRead(encodPinB1)==LOW)   count --;
```

```
}


void rencoder3()  {                                                                      // pulse and direction, direct port reading to save cycles
  //if(PINJ & 0b00000001)        count3 ++;
  if (digitalRead(U3encodB) == HIGH)    count3++;                                                       // if(digitalRead(encodPinB1)==HIGH)   count ++;
  else                    count3--;                                                    // if (digitalRead(encodPinB1)==LOW)   count --;
}


void rencoder4()  {                                                                      // pulse and direction, direct port reading to save cycles
  //if(PINH & 0b00000010)        count4 ++;
  if (digitalRead(U4encodB) == HIGH)    count4++;                                                       // if(digitalRead(encodPinB1)==HIGH)   count ++;
  else                    count4--;                                                    // if (digitalRead(encodPinB1)==LOW)   count --;
}


/************************** get command from serial: p+ start the motor ***********************************/
int getParam()  {
char param, cmd;
  if(!Serial.available())    return 0;
  delay(2);
  param = Serial.read();                                                                 // get parameter byte
  if(!Serial.available())    return 0;
  cmd = Serial.read();                                                                   // get command byte
  Serial.flush();
  switch (param) {

    case 'v':                                                                            // adjust speed
      if(cmd=='+')  {
        DC1 += 5;
        DC2 += 5;
        DC3 += 5;
        if(DC1>100)   DC1=100;
        else if(DC2>100) DC2=100;
        else if(DC3>100) DC3=100;
      }
      if(cmd=='-')    {
        DC1 -= 5;
        DC2 -= 5;
        DC3 -= 5;
        if(DC1<0)   DC1=0;
        else if (DC2<0) DC2=0;
        else if (DC3<0)  DC3=0;
      }
      break;

    case 'd':                                                                            // adjust direction
      if(cmd=='+'){
        digitalWrite(U2A1, HIGH);
        digitalWrite(U2B1, LOW);
        digitalWrite(U3A1, HIGH);
        digitalWrite(U3B1, LOW);
        digitalWrite(U4A1, HIGH);
        digitalWrite(U4B1, LOW);
      }
      if(cmd=='-')   {
        digitalWrite(U2A1, LOW);
        digitalWrite(U2B1, HIGH);
        digitalWrite(U3A1, LOW);
        digitalWrite(U3B1, HIGH);
        digitalWrite(U4A1, LOW);
        digitalWrite(U4B1, HIGH);
      }
      break;

    case 'o':                                                                            // user should type "oo"
      if(cmd == 'o')
     {
      digitalWrite(U2A1, LOW);
      digitalWrite(U2B1, LOW);
      digitalWrite(U3A1, LOW);
      digitalWrite(U3B1, LOW);
      digitalWrite(U4A1, LOW);
      digitalWrite(U4B1, LOW);
      //speed_req = 0;

     }
      break;

    case 's':
      if(cmd=='s')
      {
        digitalWrite(U2A1,HIGH);
        digitalWrite(U2B1,HIGH);
        digitalWrite(U3A1,HIGH);
        digitalWrite(U3B1,HIGH);
        digitalWrite(U4A1,HIGH);
        digitalWrite(U4B1,HIGH);
        //speed_req = 0;
      }
      break;

    case 'a':
      if(cmd == '+')
      {
        pos1 += 20;
        pos2 += 20;
        pos3 += 20;
      }
      if(cmd == '-')
      {
        pos1 -= 20;
        pos2 -= 20;
        pos3 -= 20;
      }
      break;

    case 'b':
        if(cmd == '+')
        {
          step_dir = true;
          step_count += 25;
          stepper1(step_dir,step_count);
          }
         if(cmd == '-')
         {
          step_dir = false;
          step_count -= 25;
          stepper1(step_dir,step_count);
          }
          break;

    case 'p':
      if(cmd == '+')
      {
       digitalWrite(PowerOn, HIGH);
      }
      if(cmd == '-')
      {
       digitalWrite(PowerOn, LOW);
      }
      break;

    default:
      Serial.println("???");
    }
}


/**************************************** current signal smooth *********************************************/
long digital_smooth(long value, long *data_array)  {                                                          // remove signal noise
static int ndx=0;
static int rcount=0;
static long total=0;
  total -= data_array[ndx];
  data_array[ndx] = value;
  total += data_array[ndx];
  ndx = (ndx+1) % NUMREADINGS;
  if(rcount < NUMREADINGS)      rcount++;
  return total/rcount;
}


long digital_smooth_2(long value, long *data_array)  {                                                    // remove signal noise
static int ndx_2=0;
static int rcount_2=0;
static long total_2=0;
  total_2 -= data_array[ndx_2];
  data_array[ndx_2] = value;
  total_2 += data_array[ndx_2];
  ndx_2 = (ndx_2 + 1) % NUMREADINGS;
  if(rcount_2 < NUMREADINGS)      rcount_2 ++;
  return total_2 / rcount_2;
}


long digital_smooth_3(long value, long *data_array)  {                                                    // remove signal noise
static int ndx_3=0;
static int rcount_3=0;
static long total_3=0;
  total_3 -= data_array[ndx_3];
  data_array[ndx_3] = value;
  total_3 += data_array[ndx_3];
  ndx_3 = (ndx_3 + 1) % NUMREADINGS;
  if(rcount_3 < NUMREADINGS)      rcount_3 ++;
  return total_3 / rcount_3;
}


long digital_smooth_4(long value, long *data_array)  {                                                    // remove signal noise
static int ndx_4 = 0;
static int rcount_4 = 0;
static long total_4 = 0;
  total_4 -= data_array[ndx_4];
  data_array[ndx_4] = value;
  total_4 += data_array[ndx_4];
  ndx_4 = (ndx_4 + 1) % NUMREADINGS;
  if(rcount_4 < NUMREADINGS)      rcount_4 ++;
  return total_4 / rcount_4;
}


void stepper(boolean dir, int steps){
static int last_steps = 0;
digitalWrite(dirPin,dir);
delay(20);

step_mid = abs(steps - last_steps);
last_steps = steps;

for(int i = 0; i< step_mid; i++)
```

```
 {
  digitalWrite(stepperPin, HIGH);
  delayMicroseconds(80);
  digitalWrite(stepperPin,LOW);
  delay(2);
 }
}
```

Arduino Due:

```
#include <MultiStepper.h>
#include "fis_header.h"
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050_9Axis_MotionApps41.h"
#include "math.h"

//*****************************************Arduino Due and Mega Serial Communication****************************
const char startOfNumberDelimiter = '<';
const char endOfNumberDelimiter   = '>';

//**************************************Fuzzy Logic(slippage control discourse -5~5; 4~20)*****************************
// Number of inputs to the fuzzy inference system
const int fis_gcI = 2;
// Number of outputs to the fuzzy inference system
const int fis_gcO = 3;
// Number of rules to the fuzzy inference system
const int fis_gcR = 25;
FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];

//***************************************Pitch and Roll Complementary Filter*****************************
//int steps  = 0;
int DC1    = 12;
int DC2    = 12;
int DC3    = 12;

//*****************************************Pitch and Roll Complementary Filter*****************************
int   STD_LOOP_TIME          =          150;
int   lastLoopTime           =          STD_LOOP_TIME;
int   lastLoopUsefulTime     =          STD_LOOP_TIME;
unsigned long loopStartTime    =          0;
int     forceMagnitudeApprox   =          0;
float   sensorValue[6]        = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
float   sensorZero[6]         = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
float   act_Y_Angle  = 0.0, act_X_Angle  = 0.0;
float   ACC_Y_angle  = 0.0, ACC_X_angle  = 0.0;
float   GYRO_Y_rate  = 0.0, GYRO_X_rate  = 0.0;
int16_t a1, a2, a3, g1, g2, g3;
float   ax, ay, az, gx, gy, gz;
float   dt = 0.0;

MPU6050 mpu;

//*********************************************setup mpu9150 and serial*****************************
void setup()
{
  Wire.begin();                                   //I2C protocol
 // delay(1);
  Serial1.begin(115200);
  while (!Serial1) {};
  //while(!Wire.available()) {};

  //initialize MPU9150 device
  mpu.initialize();
  //verify connection
  //Serial.println(F("Testing device connections..."));
  //Serial.println(mpu.testConnection() ? F("MPU9150 connection successful!") : F("MPU9150 connection failed!"));
  //set sample rate to 8 kHz/(1+rate) shows 1 kHz, accelerometer ODR is fixed at 1 kHz
  mpu.setRate(7);
  //set bandwidth of both gyro and accelerometer to ~20Hz
  mpu.setDLPFMode(4);
  //set gyro range to 250 degrees/sec : 131 LSB/(deg/s); 0 = +/- 250 degrees/sec:131; 1 = +/- 500 degrees/sec:65.5; 2 = +/- 1000 degrees/sec:32.8;3 = +/- 2000
degrees/sec:16.4
  mpu.setFullScaleGyroRange(0);
  //set acceleromter to 2 g range : 16384 LSB/g; 0 = +/- 2g:16384 LSB/g; 1 = +/- 4g:8192 LSB/g; 2 = +/- 8g:4096 LSB/g; 3 = +/- 16g:2048 LSB/g
  mpu.setFullScaleAccelRange(0);
  //enable data ready interrupt
  mpu.setIntDataReadyEnabled(true);
  delay(1);

  //*********************************************calibrate sensor*****************************
  calibrateSensors();
  delay(2);

  loopStartTime = millis();

}

//*********************************************main loop*********************************************
void loop()
{
  Pitch_Roll_Complementary();                              //get accurate Pitch and Roll angle from Complementary Filter
  Pitch_Roll_Fuzzy();                                      //according to pitch roll angle then give three DC motors speed

  serialOut_Fuzzy();                                       //serial output DC1, DC2, DC3

  //*********************************************loop timing control*************************************
  lastLoopUsefulTime = millis() - loopStartTime;
  if (lastLoopUsefulTime < STD_LOOP_TIME)
    delay(STD_LOOP_TIME - lastLoopUsefulTime);
  lastLoopTime = millis() - loopStartTime;
  loopStartTime = millis();

}

//**********************************************calibrate sensors****************************************
void calibrateSensors()
{
  float v;
  for (int n = 0; n < 6; n++)
  {
    v = 0.0;
    for (int i = 0; i < 50; i++)
      v = v + readSensors(n);
    sensorZero[n] = v / 50.0;
  }
  sensorZero[2] -= 16384.0;
}

//*****************************************Get Accurate Pitch and Roll Angle*********************************
void Pitch_Roll_Complementary() {

  if (mpu.getIntDataReadyStatus() == 1)
    updateSensors();                                       //update the ax,ay,az,gx,gy,gz
  ACC_Y_angle = getAcc_Y_Angle();                    // rotate y axis
  GYRO_Y_rate = getGyro_Y_Rate();

  ACC_X_angle = getAcc_X_Angle();                    // rotate x axis
  GYRO_X_rate = getGyro_X_Rate();
  dt = lastLoopTime / 1000.0;
  act_Y_Angle -= GYRO_Y_rate * dt;
  act_X_Angle += GYRO_X_rate * dt;
  // compensate for drift with accelerometer data if without extra force. sensitivity = -2g to 2g at 16Bit -> 2g = 32768, 1g =16384, 0.5g =8192.
  forceMagnitudeApprox = abs(a1) + abs(a2) + abs(a3);
  if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)
  {
    //if t = 0.75, then a = 0.75/(0.75+0.03) = 0.96; a is the complementary coefficient; sample rate is 33.33Hz
    act_Y_Angle = act_Y_Angle * 0.96 + ACC_Y_angle * 0.04;
    delay(1);
    act_X_Angle = act_X_Angle * 0.96 + ACC_X_angle * 0.04;
  }

}

//***************************************Pitch And Roll Fuzzy Logic Control*************************************
void Pitch_Roll_Fuzzy() {

  // Read Input: ep
  g_fisInput[0] = act_Y_Angle;
  // Read Input: er
  g_fisInput[1] = act_X_Angle;

  g_fisOutput[0] = 0;
  g_fisOutput[1] = 0;
  g_fisOutput[2] = 0;

  fis_evaluate();

  // Set output vlaue: step
  //steps = int(g_fisOutput[0]);
  // Set output vlaue: DC1
  DC1 = int(g_fisOutput[0]);
  // Set output vlaue: DC2
  DC2 = int(g_fisOutput[1]);
  // Set output vlaue: DC3
  DC3 = int(g_fisOutput[2]);

}

//**********************************************serial output data***************************************
void serialOut_Fuzzy()
{
  //Serial.print(int(ACC_Y_angle));
  //Serial.print("\t");
  //delay(1);
  Serial1.print(startOfNumberDelimiter);
  Serial1.print(DC1); //DC motor 1 speed
  Serial1.print(endOfNumberDelimiter);
  //Serial.print("\t");
  //delay(1);
  //Serial.print(int(ACC_X_angle));
  //Serial.print("\t");
  //delay(1);
  Serial1.print(startOfNumberDelimiter);
  Serial1.print(DC2);//DC motor 2 speed
  Serial1.print(endOfNumberDelimiter);
  //delay(1);
  //Serial.print("\t");
  //Serial.println(lastLoopTime);
  Serial1.print(startOfNumberDelimiter);
  Serial1.print(DC3);//DC motor 3 speed
  Serial1.print(endOfNumberDelimiter);
/*  delay(2);
  Serial1.print(startOfNumberDelimiter);
  Serial1.print(lastLoopTime);
  Serial1.print(endOfNumberDelimiter);
  delay(5);
*/

}

//***********************************************************************
// Support functions for Fuzzy Inference System
//***********************************************************************
// Triangular Member Function
FIS_TYPE fis_trimf(FIS_TYPE x, FIS_TYPE* p)
{
    FIS_TYPE a = p[0], b = p[1], c = p[2];
    FIS_TYPE t1 = (x - a) / (b - a);
    FIS_TYPE t2 = (c - x) / (c - b);
    if ((a == b) && (b == c)) return (FIS_TYPE) (x == a);
    if (a == b) return (FIS_TYPE) (t2*(b <= x)*(x <= c));
    if (b == c) return (FIS_TYPE) (t1*(a <= x)*(x <= b));
    t1 = min(t1, t2);
    return (FIS_TYPE) max(t1, 0);
}

FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
{
    return min(a, b);
}

FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
{
    return max(a, b);
}

FIS_TYPE fis_array_operation(FIS_TYPE *array, int size, _FIS_ARR_OP pfnOp)
{
    int i;
    FIS_TYPE ret = 0;

    if (size == 0) return ret;
    if (size == 1) return array[0];

    ret = array[0];
    for (i = 1; i < size; i++)
    {
        ret = (*pfnOp)(ret, array[i]);
    }

    return ret;
}

//***********************************************************************
// Data for Fuzzy Inference System
//***********************************************************************
// Pointers to the implementations of member functions
_FIS_MF fis_gMF[] =
{
    fis_trimf
};

// Count of member function for each Input
int fis_gIMFCount[] = { 5, 5 };

// Count of member function for each Output
int fis_gOMFCount[] = { 5, 5, 5 };

// Coefficients for the Input Member Functions
FIS_TYPE fis_gMFI0Coeff1[] = { -50, -15, -3.75 };
FIS_TYPE fis_gMFI0Coeff2[] = { -7.5, -3.75, 0 };
FIS_TYPE fis_gMFI0Coeff3[] = { -3.75, 0, 3.75 };
FIS_TYPE fis_gMFI0Coeff4[] = { 0, 3.75, 7.5 };
FIS_TYPE fis_gMFI0Coeff5[] = { 3.75, 15, 50 };
FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2, fis_gMFI0Coeff3, fis_gMFI0Coeff4, fis_gMFI0Coeff5 };
FIS_TYPE fis_gMFI1Coeff1[] = { -50, -15, -3.75 };
FIS_TYPE fis_gMFI1Coeff2[] = { -7.5, -3.75, 0 };
FIS_TYPE fis_gMFI1Coeff3[] = { -3.75, 0, 3.75 };
FIS_TYPE fis_gMFI1Coeff4[] = { 0, 3.75, 7.5 };
FIS_TYPE fis_gMFI1Coeff5[] = { 3.75, 15, 50 };
FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2, fis_gMFI1Coeff3, fis_gMFI1Coeff4, fis_gMFI1Coeff5 };
FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff };

// Coefficients for the Input Member Functions
FIS_TYPE fis_gMFO0Coeff1[] = { 0, 2, 9 };
FIS_TYPE fis_gMFO0Coeff2[] = { 6, 9, 12 };
FIS_TYPE fis_gMFO0Coeff3[] = { 9, 12, 15 };
FIS_TYPE fis_gMFO0Coeff4[] = { 12, 15, 18 };
FIS_TYPE fis_gMFO0Coeff5[] = { 15, 22, 24 };
FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2, fis_gMFO0Coeff3, fis_gMFO0Coeff4, fis_gMFO0Coeff5 };
```

```
FIS_TYPE fis_gMFO1Coeff1[] = { 0, 2, 9 };
FIS_TYPE fis_gMFO1Coeff2[] = { 6, 9, 12 };
FIS_TYPE fis_gMFO1Coeff3[] = { 9, 12, 15 };
FIS_TYPE fis_gMFO1Coeff4[] = { 12, 15, 18 };
FIS_TYPE fis_gMFO1Coeff5[] = { 15, 22, 24 };
FIS_TYPE* fis_gMFO1Coeff[] = { fis_gMFO1Coeff1, fis_gMFO1Coeff2, fis_gMFO1Coeff3, fis_gMFO1Coeff4, fis_gMFO1Coeff5 };
FIS_TYPE fis_gMFO2Coeff1[] = { 0, 2, 9 };
FIS_TYPE fis_gMFO2Coeff2[] = { 6, 9, 12 };
FIS_TYPE fis_gMFO2Coeff3[] = { 9, 12, 15 };
FIS_TYPE fis_gMFO2Coeff4[] = { 12, 15, 18 };
FIS_TYPE fis_gMFO2Coeff5[] = { 15, 22, 24 };
FIS_TYPE* fis_gMFO2Coeff[] = { fis_gMFO2Coeff1, fis_gMFO2Coeff2, fis_gMFO2Coeff3, fis_gMFO2Coeff4, fis_gMFO2Coeff5 };
FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff, fis_gMFO1Coeff, fis_gMFO2Coeff };

// Input membership function set
int fis_gMFI0[] = { 0, 0, 0, 0, 0 };
int fis_gMFI1[] = { 0, 0, 0, 0, 0 };
int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1};

// Output membership function set
int fis_gMFO0[] = { 0, 0, 0, 0, 0 };
int fis_gMFO1[] = { 0, 0, 0, 0, 0 };
int fis_gMFO2[] = { 0, 0, 0, 0, 0 };
int* fis_gMFO[] = { fis_gMFO0, fis_gMFO1, fis_gMFO2};

// Rule Weights
FIS_TYPE fis_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Type
int fis_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Inputs
int fis_gRI0[] = { 1, 2 };
int fis_gRI1[] = { 1, 3 };
int fis_gRI2[] = { 1, 4 };
int fis_gRI3[] = { 2, 2 };
int fis_gRI4[] = { 2, 3 };
int fis_gRI5[] = { 2, 4 };
int fis_gRI6[] = { 3, 2 };
int fis_gRI7[] = { 3, 3 };
int fis_gRI8[] = { 3, 4 };
int fis_gRI9[] = { 4, 2 };
int fis_gRI10[] = { 4, 3 };
int fis_gRI11[] = { 4, 4 };
int fis_gRI12[] = { 5, 2 };
int fis_gRI13[] = { 5, 3 };
int fis_gRI14[] = { 5, 4 };
int* fis_gRI[] = { fis_gRI0, fis_gRI1, fis_gRI2, fis_gRI3, fis_gRI4, fis_gRI5, fis_gRI6, fis_gRI7, fis_gRI8, fis_gRI9, fis_gRI10, fis_gRI11, fis_gRI12, fis_gRI13, fis_gRI14 };

// Rule Outputs
int fis_gRO0[] = { 5, 1, 2 };
int fis_gRO1[] = { 5, 1, 1 };
int fis_gRO2[] = { 5, 2, 1 };
int fis_gRO3[] = { 5, 1, 2 };
int fis_gRO4[] = { 5, 1, 1 };
int fis_gRO5[] = { 5, 2, 1 };
int fis_gRO6[] = { 3, 1, 2 };
int fis_gRO7[] = { 3, 3, 3 };
int fis_gRO8[] = { 3, 2, 1 };
int fis_gRO9[] = { 3, 1, 2 };
int fis_gRO10[] = { 3, 4, 4 };
int fis_gRO11[] = { 3, 2, 1 };
int fis_gRO12[] = { 3, 4, 5 };
int fis_gRO13[] = { 3, 4, 4 };
int fis_gRO14[] = { 3, 5, 4 };
int* fis_gRO[] = { fis_gRO0, fis_gRO1, fis_gRO2, fis_gRO3, fis_gRO4, fis_gRO5, fis_gRO6, fis_gRO7, fis_gRO8, fis_gRO9, fis_gRO10, fis_gRO11, fis_gRO12, fis_gRO13,
fis_gRO14 };

// Input range Min
FIS_TYPE fis_gIMin[] = { -25, -25 };

// Input range Max
FIS_TYPE fis_gIMax[] = { 25, 25 };

// Output range Min
FIS_TYPE fis_gOMin[] = { 0, 0, 0 };

// Output range Max
FIS_TYPE fis_gOMax[] = { 24, 24, 24 };

//***********************************************************************
// Data dependent support functions for Fuzzy Inference System
//***********************************************************************
FIS_TYPE fis_MF_out(FIS_TYPE** fuzzyRuleSet, FIS_TYPE x, int o)
{
    FIS_TYPE mfOut;
    int r;

    for (r = 0; r < fis_gcR; ++r)
    {
        int index = fis_gRO[r][o];
        if (index > 0)
        {
            index = index - 1;
            mfOut = (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else if (index < 0)
        {
            index = -index - 1;
            mfOut = 1 - (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else
        {
            mfOut = 0;
        }

        fuzzyRuleSet[0][r] = fis_min(mfOut, fuzzyRuleSet[1][r]);
    }
    return fis_array_operation(fuzzyRuleSet[0], fis_gcR, fis_max);
}

FIS_TYPE fis_defuzz_centroid(FIS_TYPE** fuzzyRuleSet, int o)
{
    FIS_TYPE step = (fis_gOMax[o] - fis_gOMin[o]) / (FIS_RESOLUSION - 1);
    FIS_TYPE area = 0;
    FIS_TYPE momentum = 0;
    FIS_TYPE dist, slice;
    int i;

    // calculate the area under the curve formed by the MF outputs
    for (i = 0; i < FIS_RESOLUSION; ++i){
        dist = fis_gOMin[o] + (step * i);
        slice = step * fis_MF_out(fuzzyRuleSet, dist, o);
        area += slice;
        momentum += slice*dist;
    }

    return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o]) / 2) : (momentum / area));
}

//***********************************************************************
// Fuzzy Inference System
//***********************************************************************
void fis_evaluate()
{
    FIS_TYPE fuzzyInput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE fuzzyInput1[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyInput[fis_gcI] = { fuzzyInput0, fuzzyInput1, };
    FIS_TYPE fuzzyOutput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE fuzzyOutput1[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE fuzzyOutput2[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyOutput[fis_gcO] = { fuzzyOutput0, fuzzyOutput1, fuzzyOutput2, };
    FIS_TYPE fuzzyRules[fis_gcR] = { 0 };
    FIS_TYPE fuzzyFires[fis_gcR] = { 0 };
    FIS_TYPE* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };
    FIS_TYPE sW = 0;

    // Transforming input to fuzzy Input
    int i, j, r, o;
    for (i = 0; i < fis_gcI; ++i)
    {
        for (j = 0; j < fis_gIMFCount[i]; ++j)
        {
            fuzzyInput[i][j] =
                (fis_gMF[fis_gMFI[i][j]])(g_fisInput[i], fis_gMFICoeff[i][j]);
        }
    }

    int index = 0;
    for (r = 0; r < fis_gcR; ++r)
    {
        if (fis_gRType[r] == 1)
        {
            fuzzyFires[r] = FIS_MAX;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r], fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1);
            }
        }
        else
        {
            fuzzyFires[r] = FIS_MIN;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r], fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 0);
            }
        }
```

```
      fuzzyFires[r] = fis_gRWeight[r] * fuzzyFires[r];
      sW += fuzzyFires[r];
   }

   if (sW == 0)
   {
      for (o = 0; o < fis_gcO; ++o)
      {
         g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
      }
   }
   else
   {
      for (o = 0; o < fis_gcO; ++o)
      {
         g_fisOutput[o] = fis_defuzz_centroid(fuzzyRuleSet, o);
      }
   }
}

//**********************************************update sensors date****************************************
void updateSensors()
{
 float v;
 for (int n = 0; n < 6; n++)
 {
   v = 0.0;
   for (int i = 0; i < 5; i++)
     v = v + readSensors(n);
   sensorValue[n] = v / 5.0 - sensorZero[n];
 }
}

//**********************************************read sensors data*****************************************
float readSensors(int channel)
{

 if (mpu.getIntDataReadyStatus() == 1)
 {
   mpu.getAcceleration(&a1, &a2, &a3);
   ax = (float)a1;
   ay = (float)a2;
   az = (float)a3;
   mpu.getRotation(&g1, &g2, &g3);
   gx = (float)g1;
   gy = (float)g2;
   gz = (float)g3;
 }
 float mat[6] = {ax, ay, az, gx, gy, gz};
 return mat[channel];

}

//**********************************************get y axis & x axis deg/s*********************************
float getGyro_Y_Rate()
{
 return sensorValue[4] / 131.0;                //return gyro y
}

float getGyro_X_Rate()
{
 return sensorValue[3] / 131.0;                //return gyro x
}

//**********************************************get Acc y & x axis angle*********************************
float getAcc_Y_Angle()
{
 float a = 0.0;
 a = atan2(sensorValue[0], sensorValue[2]);           //atan2(y,x) return quid; degree = atan2(y,x) * 180.0 / 3.14; belong to (-180, 180)
 return (a * 180.0 / 3.14);
}


float getAcc_X_Angle()
{
 float b = 0.0;
 b = atan2(sensorValue[1], sensorValue[2]);
 return (b * 180.0 / 3.14);
}
```

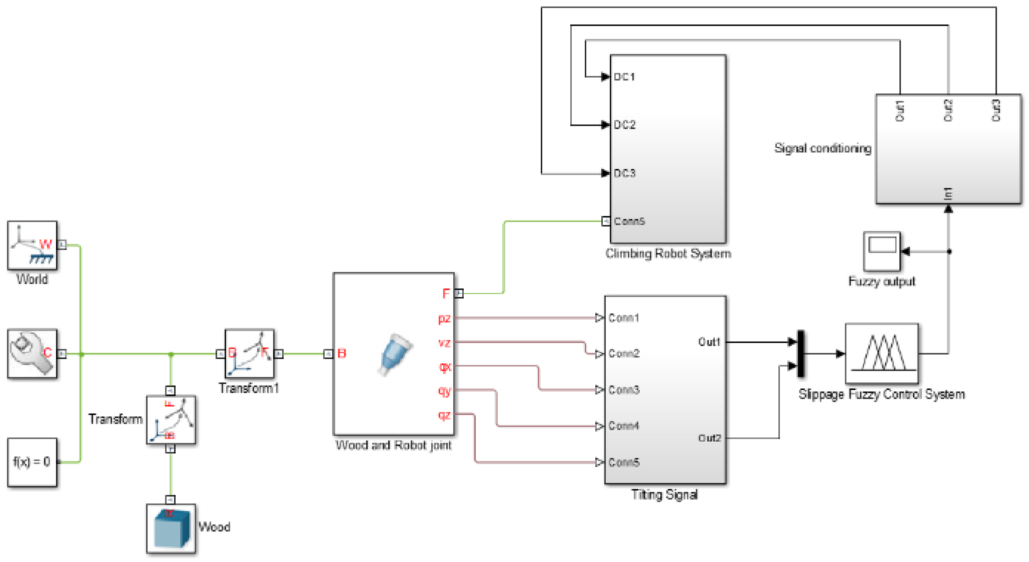
NARX Feedback Neural Network for Wheel Slip Control

```
function [y1,xf1,xf2] = myNeuralNetworkFunction(x1,x2,xi1,xi2)
%MYNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 01-Mar-2018 13:23:26.
%
% [y1,xf1,xf2] = myNeuralNetworkFunction(x1,x2,xi1,xi2) takes these arguments:
%   x1 = 3xTS matrix, input #1
%   x2 = 1xTS matrix, input #2
%   xi1 = 3x4 matrix, initial 4 delay states for input #1.
%   xi2 = 1x4 matrix, initial 4 delay states for input #2.
% and returns:
%   y1 = 1xTS matrix, output #1
%   xf1 = 3x4 matrix, final 4 delay states for input #1.
%   xf2 = 1x4 matrix, final 4 delay states for input #2.
% where TS is the number of timesteps.

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1_xoffset = [6;5;5];
x1_step1_gain = [0.111111111111111;0.142857142857143;0.133333333333333];
x1_step1_ymin = -1;

% Input 2
x2_step1_xoffset = 175;
x2_step1_gain = 0.08;
x2_step1_ymin = -1;

% Layer 1
b1 = [3.4652609932394545;-1.9348131781138858;0.88741159653409851;1.626233366133726;-0.93104937882842931;-
0.68914373554581676;0.4421368831599351;-
0.52729999460792576;0.31269982352440101;0.092148003823602137;0.21803259125411717;0.4305112235146048;-
0.59829251890806234;-0.77118122248419041;-0.52353751633725853;-1.2433039200435707;-
1.7290066069006087;1.1833894478448601;1.5982954100866376;-2.0658798298653198];
IW1_1 = [-1.1252539590255779 -0.17024785511313442 -0.60191738692960262 1.488319173654836 -0.49348121059945149 -
0.9975631552257429 0.52277416474681326 1.1334240273202902 -1.7039726624162645 -0.37696137317921019 0.56926637895958365
0.93524924858533576;0.56960444144478373 -0.32191424544290004 0.14604511936497652 0.26589160734833983 -
0.88331771655249625 0.86630244254967137 -0.074056145531645731 0.63532661545645652 0.20044059436407671
0.53888757386800923 -0.82374783516438677 0.27562667055794321;-0.98932559519559871 -0.80756570945220019
0.57371921737205511 0.19848534056452022 -0.70482065816438388 0.054006343610814807 -0.1896766364426698
0.46583857600027417 -0.18750373570355244 -0.11687264349186416 -0.55112436520161368
0.40535039391843131;0.35194269423564178 -0.97704843052782531 0.41505936821866807 -0.65375678996436926
0.88945946575277446 0.6313040007317251 -0.099296436384775866 0.47498666142232626 0.022098407922799719 -
0.30671423063572889 -0.45157706789517044 -0.0052482634053180599;0.13060659207767361 -1.033391180998666
0.25456944845315332 -0.22726743574152533 0.37099890283041359 -0.33988027542397625 0.32732418977738098 -
0.86643137604830234 0.15455814167472859 0.53838012949034453 -0.34735875563639257
0.10676696555822826;0.39774725145416273 -0.5015100378478522 -0.13068935754531358 -0.40042203710695345
0.41921469597677063 -0.15302545626059827 -0.33692863746773172 -0.33763546755312579 -0.29513585746268634
0.3227368612812439 -0.49531478808801116 -0.44150785659793068;-0.23096380917978712 -0.088112273419824513
0.35812831655728988 -0.14957525817269093 0.3401977511586366 -0.054205195240802327 -0.061340682656029294 -
0.28903662451053136 0.14838153520587569 0.60957451771640303 -0.0028863704412007661
0.08299823208798246;0.57616327521812882 0.23011658497155188 0.460291397247439 0.089087580571315328 -
0.69862835539646162 -0.10579044600983253 -0.19210709499747661 0.22639082259874144 0.41314066083165862 -
0.60536890055207149 0.41698969787439061 -0.24401511632647921;0.71448667009620503 0.16814310203969302
0.98393788604297416 -0.99361779503230685 0.032275819741424433 0.68138276516664398 -1.1049963782765375 -
0.76825506626700313 0.71572364354334472 -0.29722933200040785 -0.303010042749567
0.24519486898215367;0.047232759447995133 0.075181355004733177 0.031820952057788038 0.013995483805033862
0.017185478978475256 0.046500356393110565 -0.0045935767505659621 -0.039747812984054533 -0.029839381016155384 -
0.21354203260627633 0.11476328002599268 -0.18639200788185967;0.96108077093642519 1.4386242674343936 -
0.29311831984749315 0.2001511470543767 0.17843393832276946 0.061617938123036113 0.84887062641694355
0.33998028086923482 -0.27005844553327993 -0.42577841564347335 0.46143173298827694 -0.12627933002129849;-
0.25782860215017855 0.075777708501596902 0.036086997945202384 0.80934682486407439 -0.73759395160407337
0.59100014231087872 -0.31657060065967402 -0.035137666888678719 -0.52857704456380183 0.64088642956386743 -
0.38285958297447692 -0.064708556488764238;-0.15803890192260373 0.29466074542319654 0.21980842853059232 -
0.16048682220561905 0.055851624115891035 0.032886136862650442 -0.071169333786496883 -0.016300487355000629
0.18053597503878779 0.044767169661530271 -0.20817405922321103 -0.42494514358133667;-0.31866174857010759
0.78131634492399993 -0.37501229277834552 -0.12494655400565267 0.52473799661640619 -0.16249696095769048
0.23989527239762268 -0.4676810984009509 -0.070274941353196674 -0.11828180953436955 0.39987889679377564 -
0.10925051388411855;-0.34064827388589947 0.00086977204298363531 0.29362624828046591 -0.037881196083283393
0.24925845831729018 0.20034687299118425 -0.30304378487299977 -0.42123154082765224 -0.42950168369880859 -
0.32552850856247162 0.16327644605004923 0.41356363193760592;-0.68404031871178339 0.88066592372826169
0.56330191890574799 -0.29308055855791726 -0.18708154299881347 0.038130895170582005 -0.30326415819623065
0.0072903610412145547 -0.26220055155015953 -0.57418178494010652 -0.2580169472862649
0.24802112624075515;0.24312351532466489 -0.23853129616789037 0.47558146003008095 -0.47476817157686751
0.55526277174218519 0.43557097245924348 -0.52231523259380519 -0.10453056477869532 -0.48557872522764195
0.25383971800104743 -0.46753136610769724 0.83834276788550277;0.55018502486970589 0.055246207471839809
0.14276573414238203 0.62470106308589224 -0.22229072683116122 -0.22938227100000347 0.18479910951015155
0.011907991398727703 0.34920667958483742 0.39782302258853636 -1.2065690864943388
0.66333610119865638;0.1366939187085279 -0.50563518061004109 0.40473210625631817 -0.18497442974594855 -
0.6115104115167056 -0.2997576646057587 0.77986411659078025 0.19689743955532446 0.32312992117083728 0.48130689592213893
-0.09724615556865969 0.71117225189084388;-0.88590575182246956 -0.66317955897870817 0.21014096401482141 -
0.1014153701252976 0.020202093012584237 0.2060072445990139 -0.30917593460480425 0.46624805820548187 -
0.52683647719498039 -0.92591541780417164 -0.29582813805510072 0.74773391046639959];
IW1_2 = [0.36206006965465148 -0.46039745849517033 -0.053970837082655834 -0.138057475772494;0.60149592058326473
0.25367063523538819 0.31751540166674935 0.19081307791373969;-0.62074255087512331 0.23387476972206542 -
0.2218033495323985 0.16372480830170158;0.9660861457168417 0.42465980075402421 0.77440329532066654 -
0.13554409480065768;-0.012134915749129659 0.62214976959983948 -0.12040923295965408
0.45031919978462964;0.54275762780679226 0.68731681103479492 -0.17068837048281765 -0.57757896876495984;-
0.27406091895706214 0.22981037327552531 0.51483696492581887 -0.76081462619986318;0.49314216272432942 -
0.46743725151537996 0.79158720179556152 -0.21181488429611753;0.52989905501853318 -0.39977039427656808 -
0.31752742782605659 -0.040667777711143639;-0.63284045987071758 0.037170822107621099 0.15062197917507419
0.052655253360073345;0.5557235691715201 -0.16233576003190742 0.892155781433427
0.014144652910505508;0.031693751442002538 -0.20570663280409626 -0.04292693225286548 -
0.20885437464634965;0.19723208212280963 0.29127692315331222 0.075210816347086179 0.027378594971073505;-
0.66006118720661855 0.37563978757124572 -0.24078780125245638 0.076765751552907435;0.22896543797877045
0.077759076148801759 0.27744131142635642 -0.2759181460785109;0.58864305458826183 0.41755373695899617 -
0.27322109567800718 0.23243990251253402;-0.2274335754866289 -0.90643557522630513 0.22654769303768094
0.3282285411481265;-0.32210165414123426 -0.60948265956112191 0.56438780552610202 -
0.03115138326869113;0.24202592715058405 -0.27339608095951629 -0.23964340054535713 -0.38567678056921018;-
0.69437323750449875 0.32196149445657585 0.36230550190233762 -0.80341783543288359];

% Layer 2
b2 = -0.81502248133097099;
LW2_1 = [1.1387290544352762 0.072727552702546786 -0.097511281495121227 0.080124663546499608 0.017635965328615839 -
0.015967163044312307 -0.28943298910815979 0.10827097356678476 -0.053477362474633028 -1.1064604963888802
0.035084681353639659 -0.027396271482420069 0.60475369304155835 -0.065911066637865168 0.43873009511092798 -
0.19733932181024005 -0.21943221879840052 -0.035978454888549762 0.031317292589157289 -0.13949307091262456];

% Output 1
y1_step1_ymin = -1;
y1_step1_gain = 0.08;
```

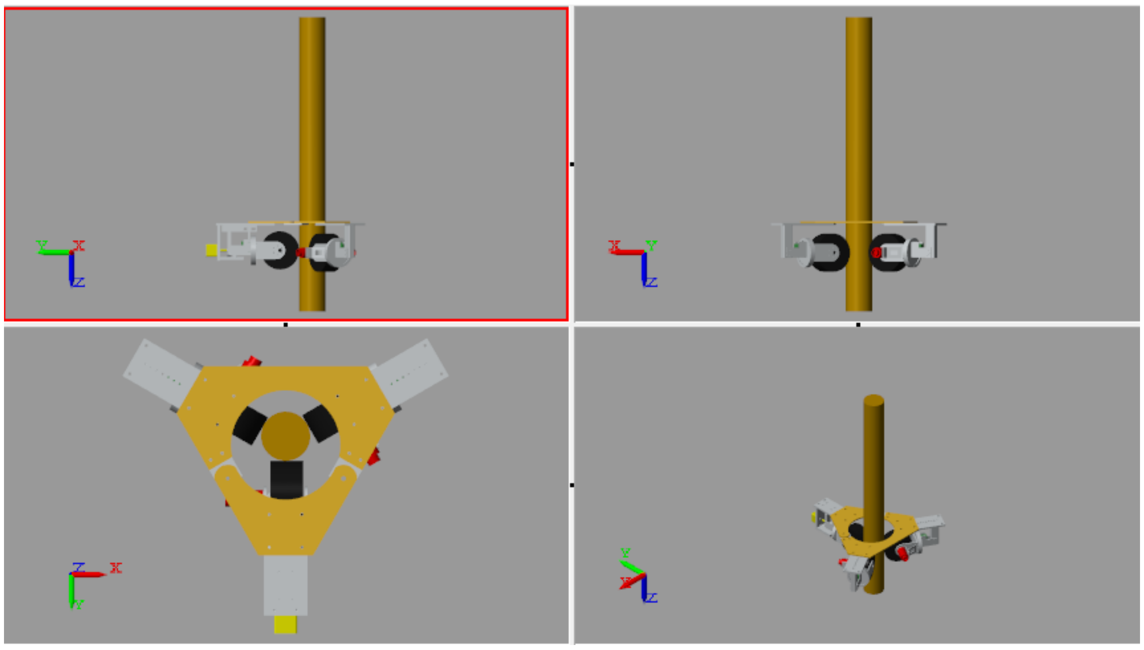
```
y1_step1_xoffset = 175;

% ===== SIMULATION ========

% Dimensions
TS = size(x1,2); % timesteps

% Input 1 Delay States
xd1 = mapminmax_apply(xi1,x1_step1_gain,x1_step1_xoffset,x1_step1_ymin);
xd1 = [xd1 zeros(3,1)];

% Input 2 Delay States
xd2 = mapminmax_apply(xi2,x2_step1_gain,x2_step1_xoffset,x2_step1_ymin);
xd2 = [xd2 zeros(1,1)];

% Allocate Outputs
y1 = zeros(1,TS);

% Time loop
for ts=1:TS

    % Rotating delay state position
    xdts = mod(ts+3,5)+1;

    % Input 1
    xd1(:,xdts) = mapminmax_apply(x1(:,ts),x1_step1_gain,x1_step1_xoffset,x1_step1_ymin);

    % Input 2
    xd2(:,xdts) = mapminmax_apply(x2(:,ts),x2_step1_gain,x2_step1_xoffset,x2_step1_ymin);

    % Layer 1
    tapdelay1 = reshape(xd1(:,mod(xdts-[1 2 3 4]-1,5)+1),12,1);
    tapdelay2 = reshape(xd2(:,mod(xdts-[1 2 3 4]-1,5)+1),4,1);
    a1 = tansig_apply(b1 + IW1_1*tapdelay1 + IW1_2*tapdelay2);

    % Layer 2
    a2 = b2 + LW2_1*a1;

    % Output 1
    y1(:,ts) = mapminmax_reverse(a2,y1_step1_gain,y1_step1_xoffset,y1_step1_ymin);
end

% Final delay states
finalxts = TS+(1: 4);
xits = finalxts(finalxts<=4);
xts = finalxts(finalxts>4)-4;
xf1 = [xi1(:,xits) x1(:,xts)];
xf2 = [xi2(:,xits) x2(:,xts)];
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings_gain,settings_xoffset,settings_ymin)
y = bsxfun(@minus,x,settings_xoffset);
y = bsxfun(@times,y,settings_gain);
y = bsxfun(@plus,y,settings_ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n)
a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing Function
function x = mapminmax_reverse(y,settings_gain,settings_xoffset,settings_ymin)
x = bsxfun(@minus,y,settings_ymin);
x = bsxfun(@rdivide,x,settings_gain);
x = bsxfun(@plus,x,settings_xoffset);
end
```

Appendix B – MATLAB SimMechanics Simulation

Robot Configuration in MATLAB SimMechanics:

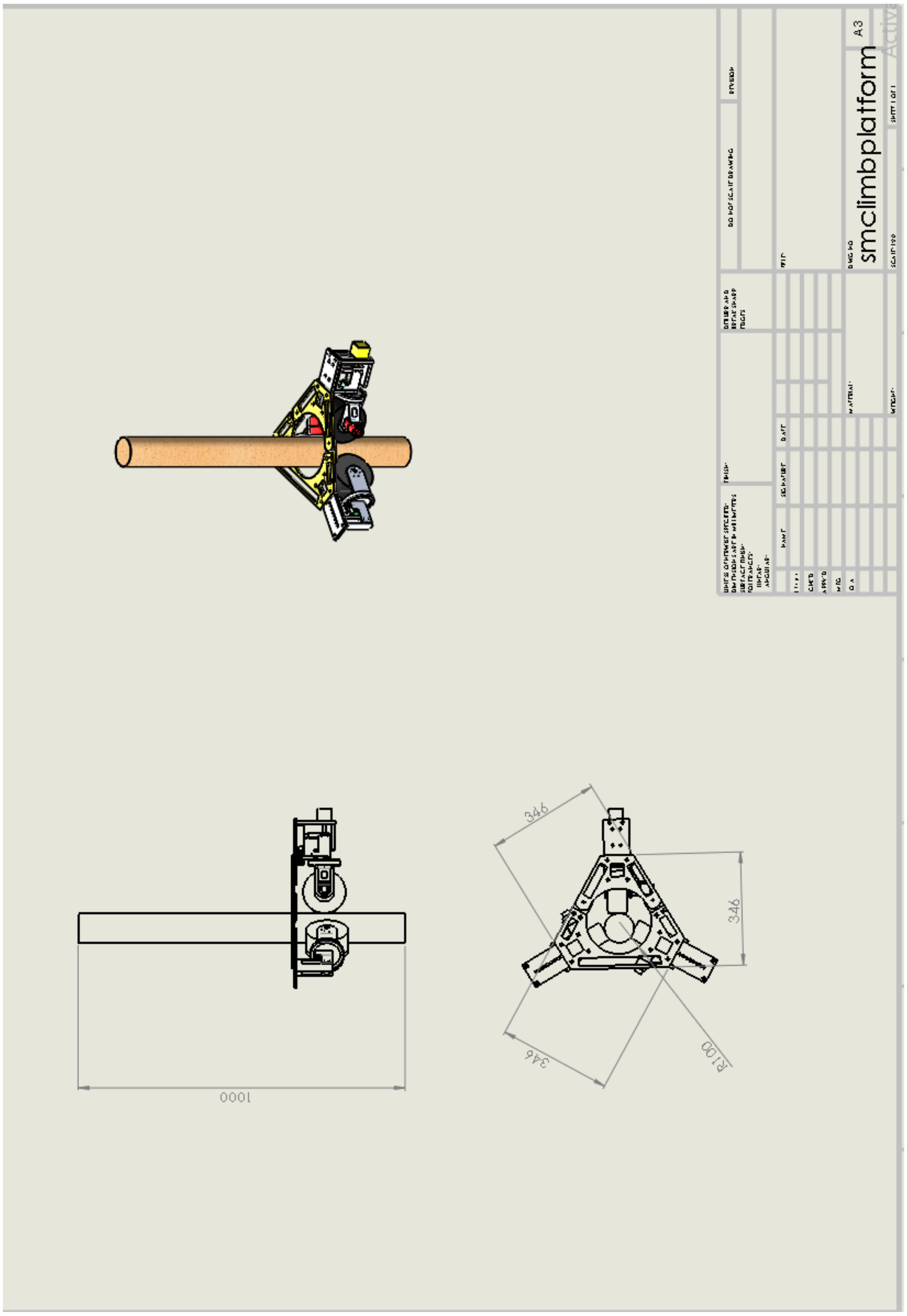


Robot 3D View in MATLAB SimMechanics Environment:

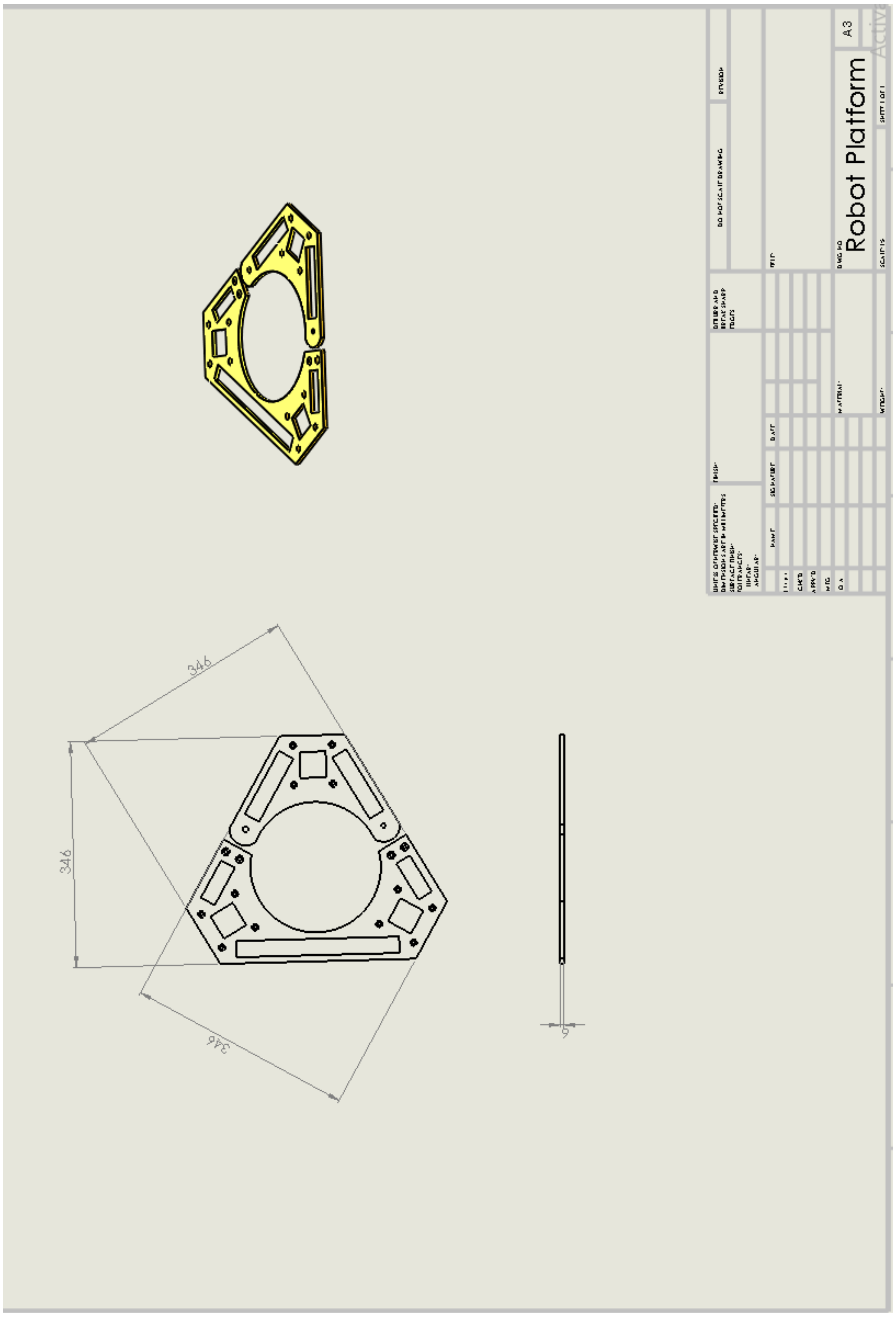


Appendix C – Tree Pruning Robot Mechanical Drawing

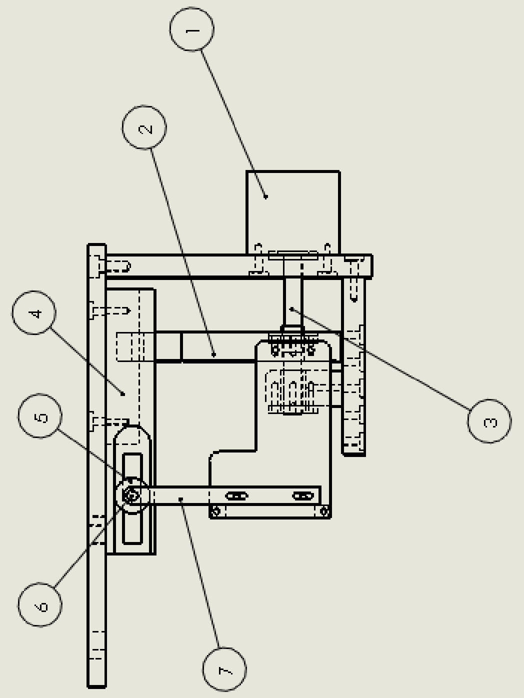
Tree Pruning Robot



Tree Pruning Robot Platform

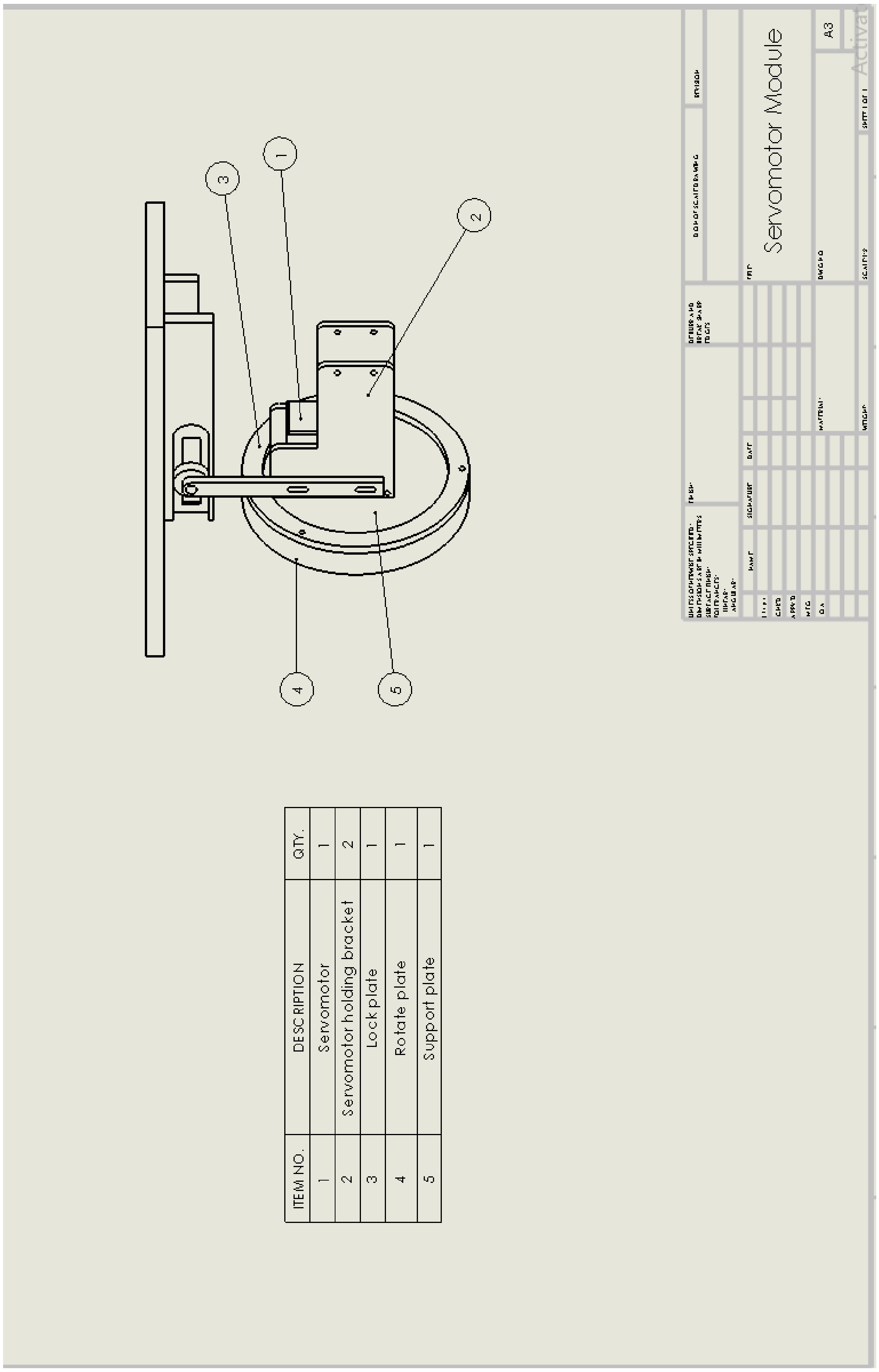


Stepper's Screw-Nut Unit and Bearing Support Mechanism



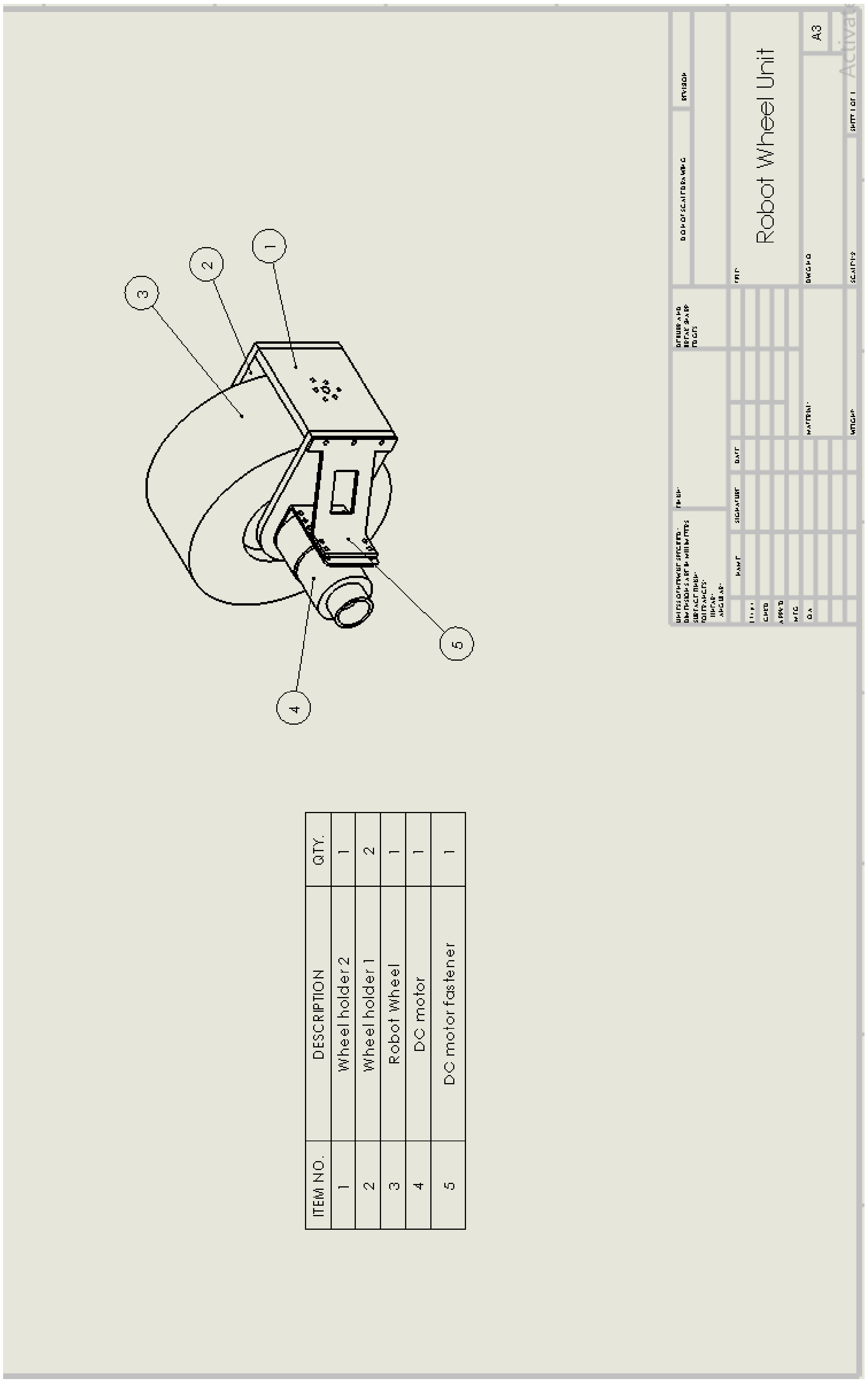
ITEM NO.	DESCRIPTION	QTY.
1	Stepper	1
2	Stepper Nut	1
3	Stepper Lead Screw	1
4	Bearing guide rail	2
5	Ball bearing	1
6	Bearing shaft	1
7	Bearing linkage	1

Servomotor Module



ITEM NO.	DESCRIPTION	QTY.
1	Servomotor	1
2	Servomotor holding bracket	2
3	Lock plate	1
4	Rotate plate	1
5	Support plate	1

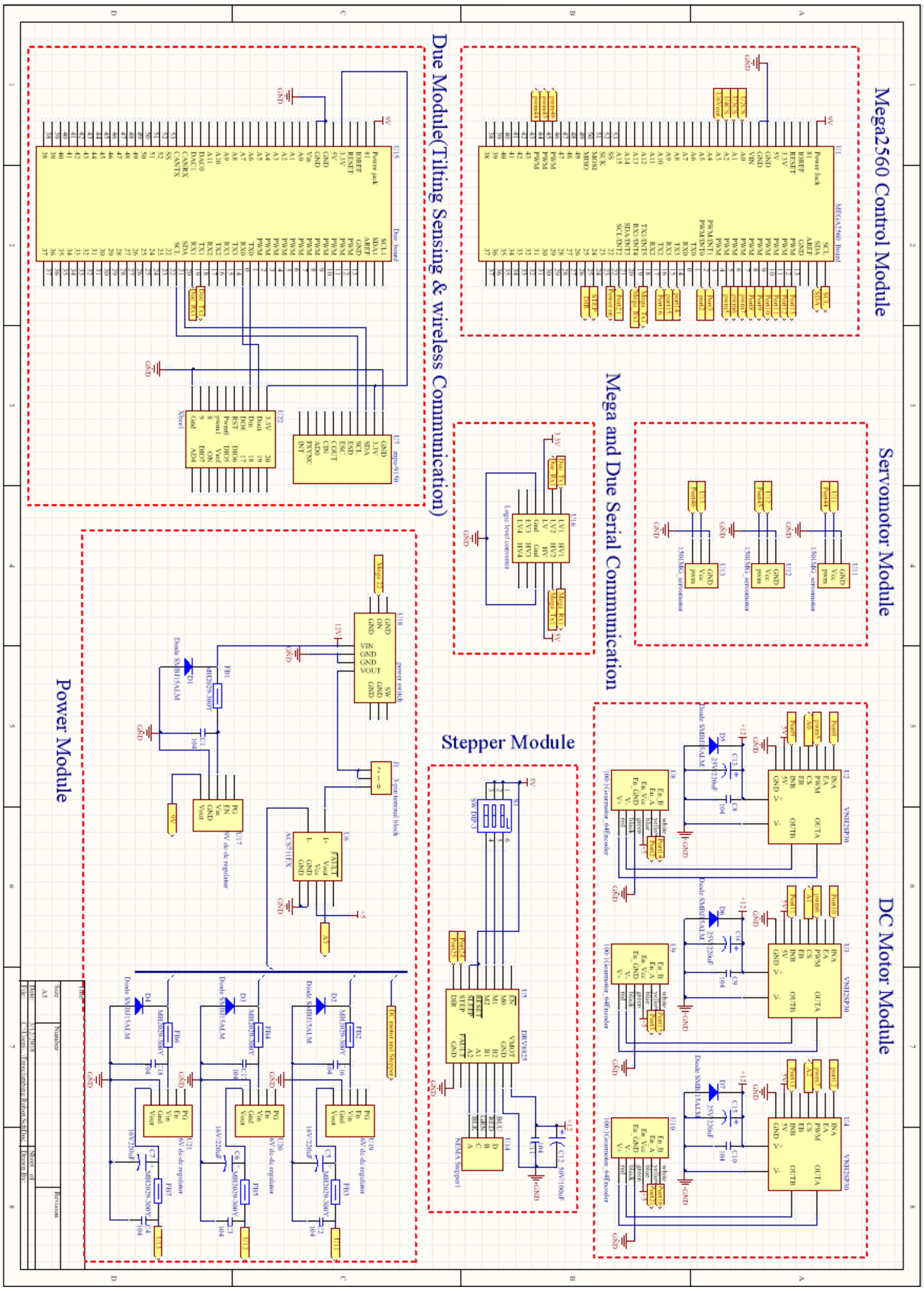
Tree Pruning Robot Wheel Unit



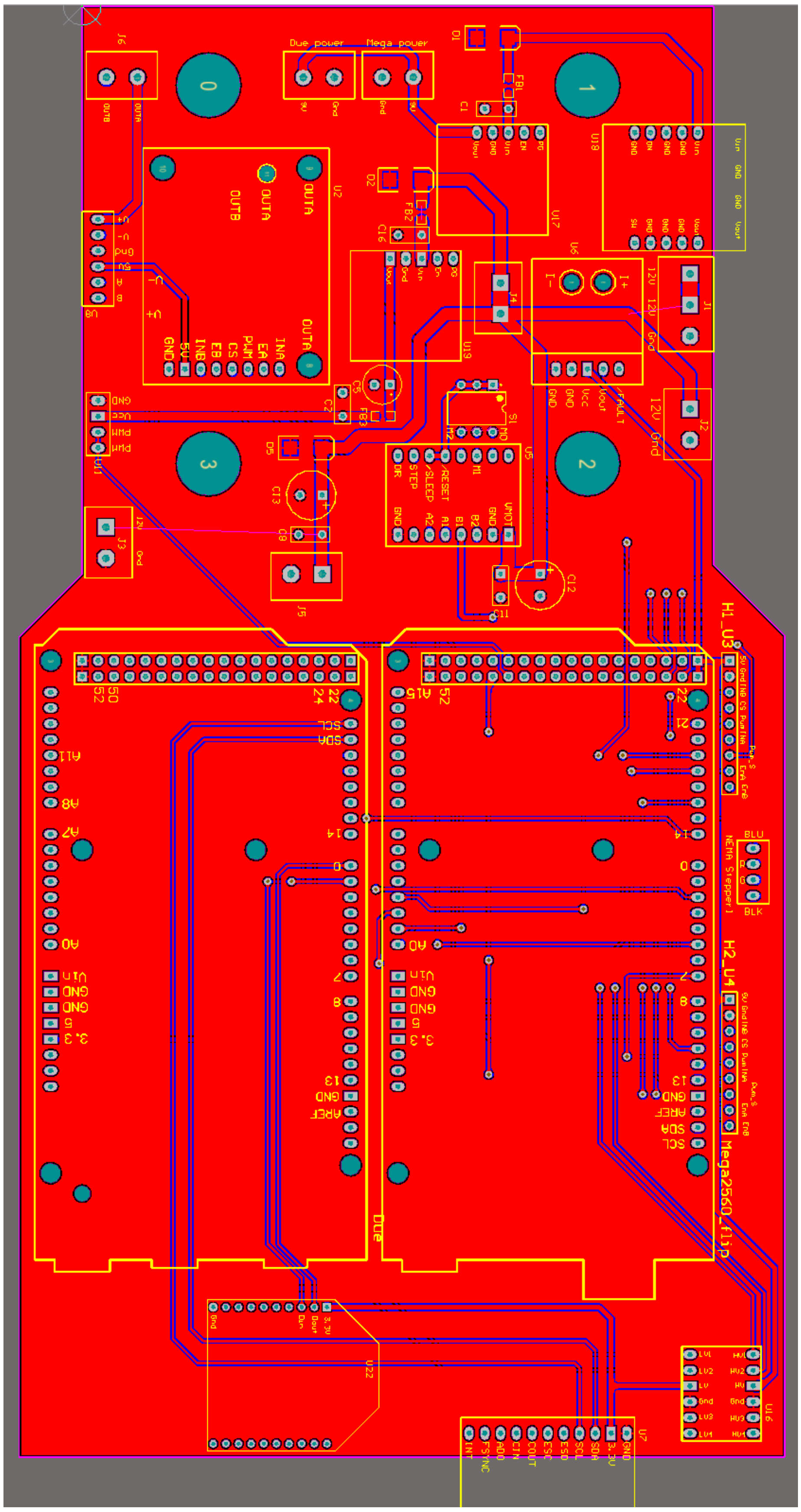
ITEM NO.	DESCRIPTION	QTY.
1	Wheel holder 2	1
2	Wheel holder 1	2
3	Robot Wheel	1
4	DC motor	1
5	DC motor fastener	1

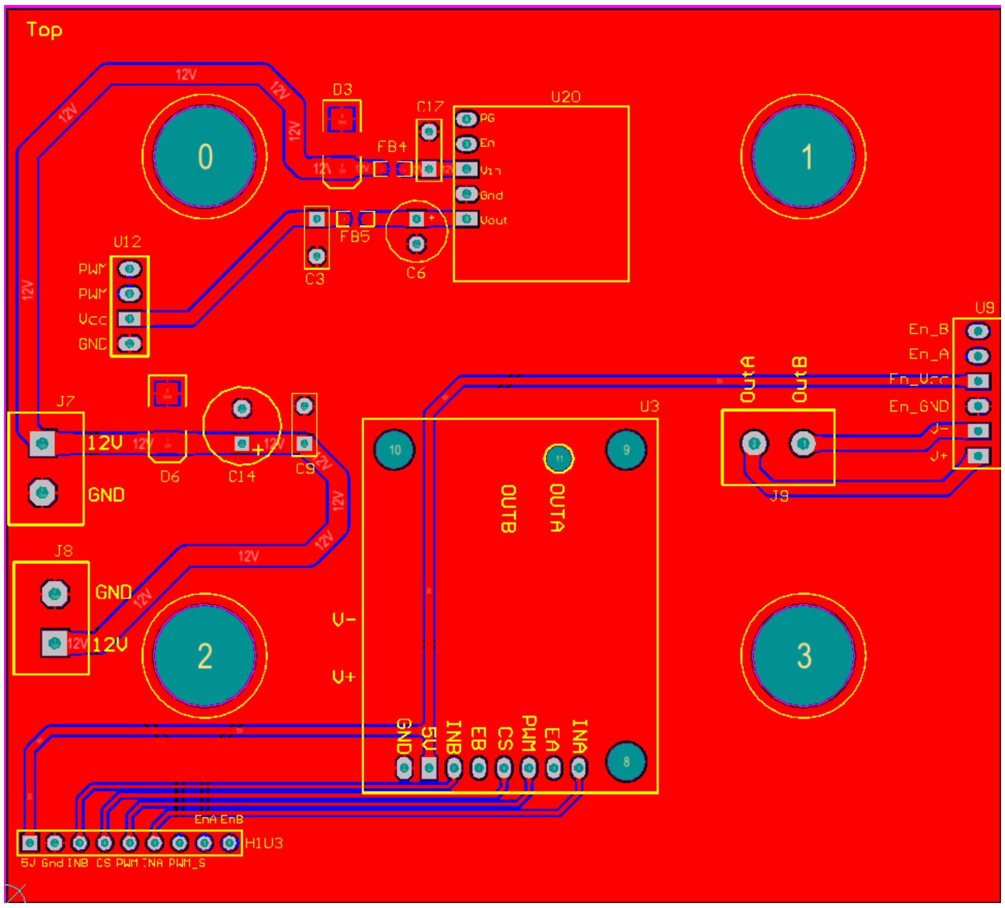
Appendix D – Tree Pruning Robot Hardware Design

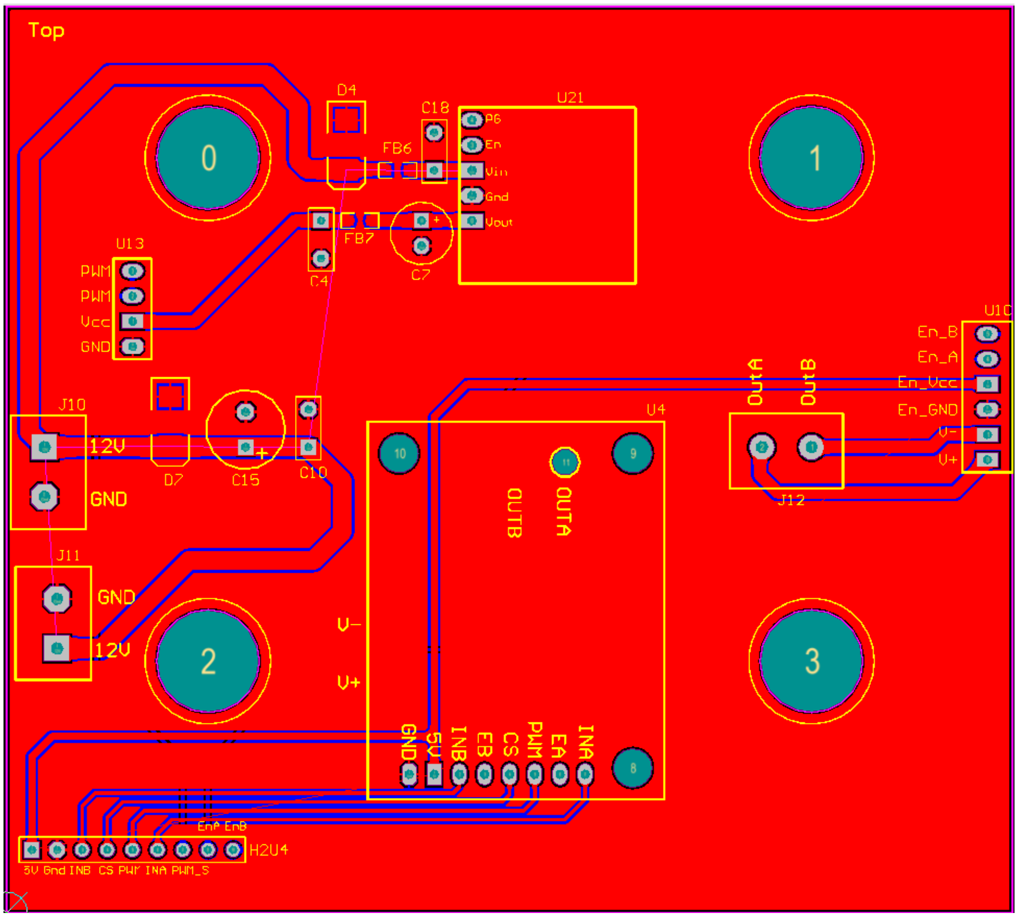
Tree Pruning Robot Schematic Diagram Design

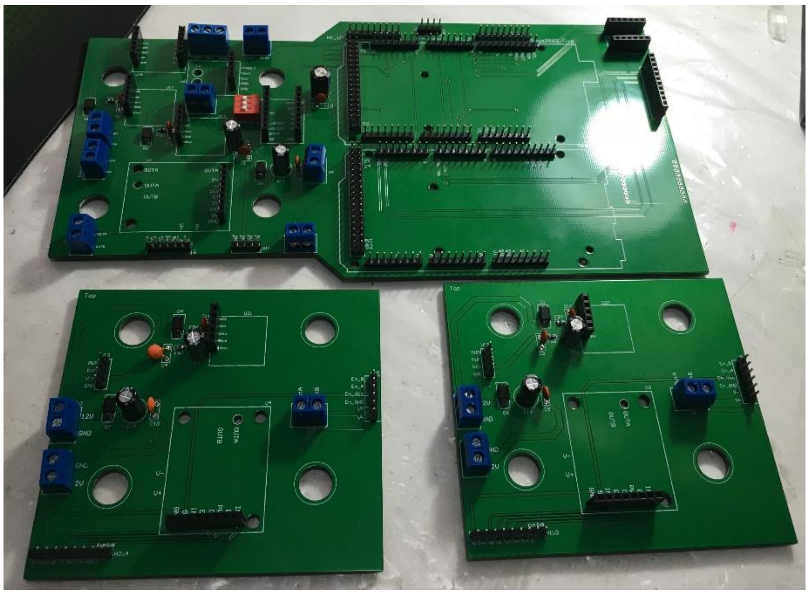
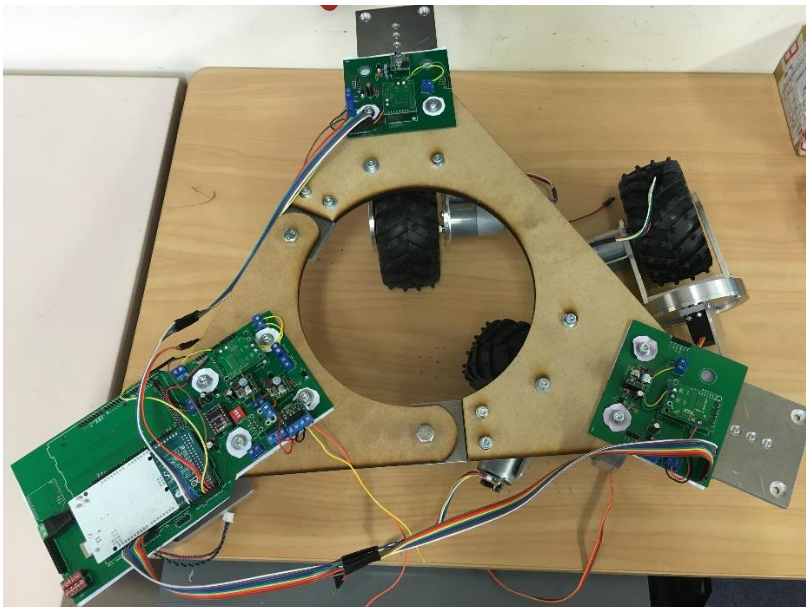
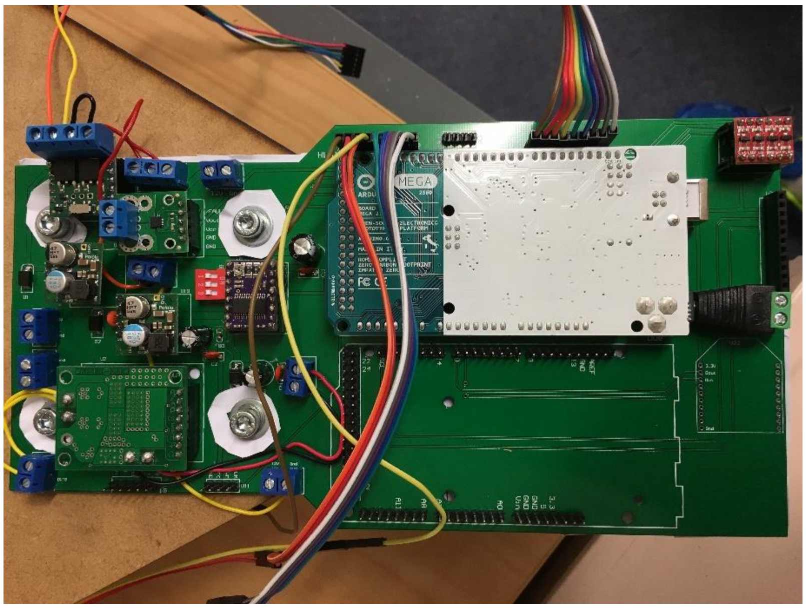

Mega2560 Control Module
Servomotor Module
DC Motor Module
Due Module(Tilting Sensing & wireless Communication)
Mega and Due Serial Communication
Stepper Module
Power Module

Tree Pruning Robot PCB Design

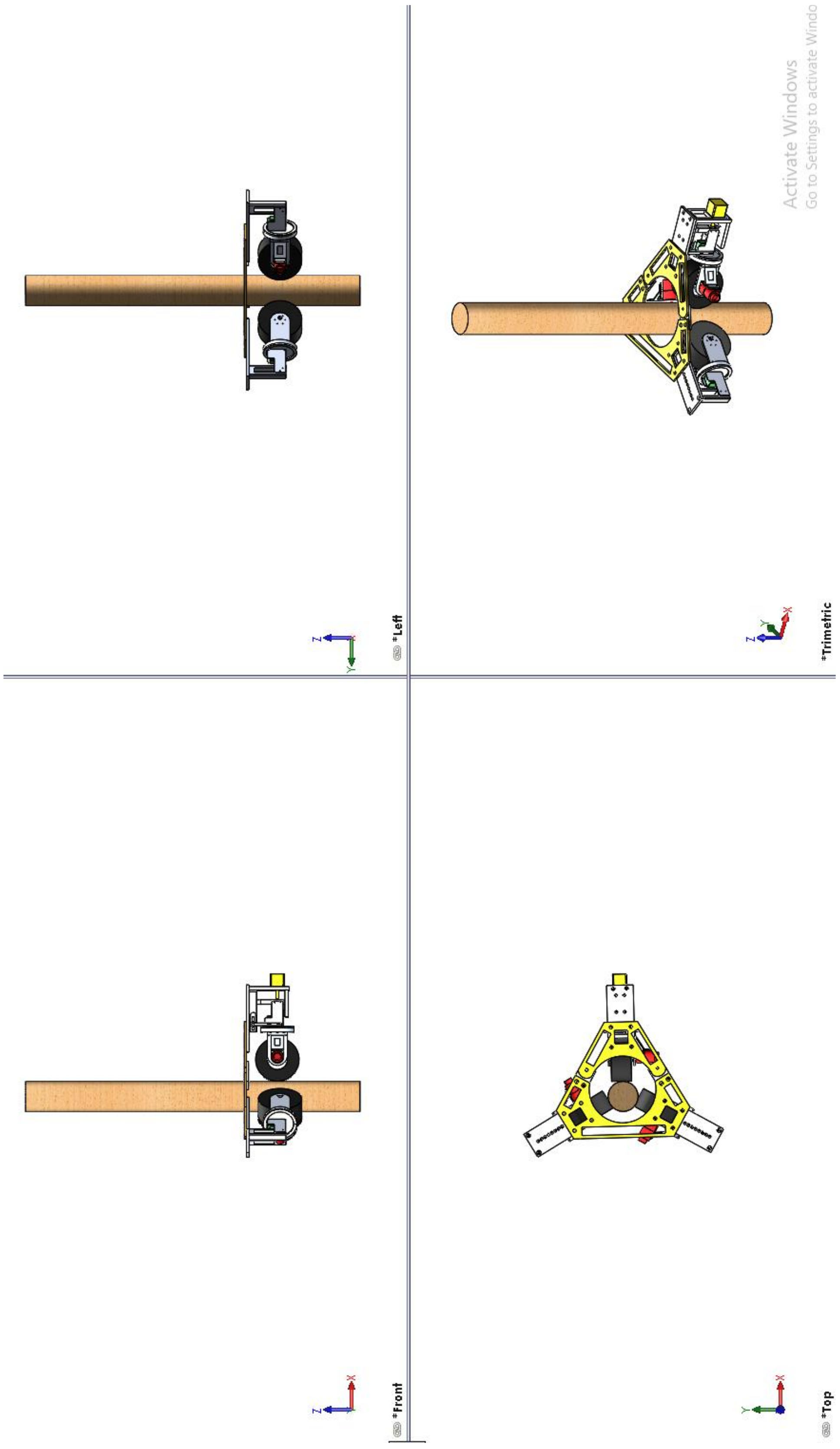



Top
0
1
2
3
12V
D3
C17
U20
FB4
FB5
C3
C6
PG
En
Vin
Gnd
Vout
U12
PWM
PWM
Vcc
GND
J7
12V
GND
D6
C14
C9
U3
J8
GND
12V
OUTB
OUTA
V-
V+
GND
5V
INB
EB
CS
PWM
EA
INA
OutA
OutB
J9
U9
En_B
En_A
En_Vcc
En_GND
V-
V+
H1U3
EnA EnB
5V Gnd INB CS PWM INA PWM_S

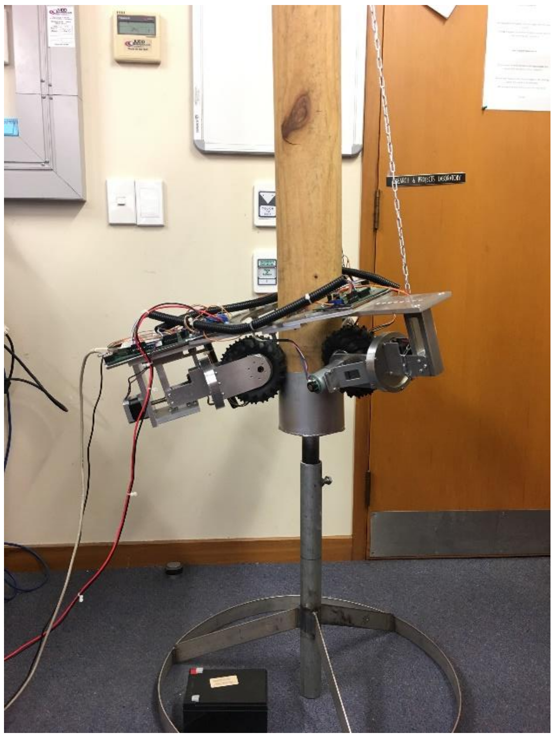

Top
0
1
2
3
D4
C18
U21
FB6
FB7
C4
C7
PG
En
Vin
Gnd
Vout
U13
PWM
PWM
Vcc
GND
J10
12V
GND
D7
C15
C10
U4
J11
GND
12V
OUTB
OUTA
V-
V+
GND
5V
INB
EB
CS
PWM
EA
INA
OutA
OutB
J12
U10
En_B
En_A
En_Vcc
En_GND
V-
V+
H2U4
EnA EnB
5V Gnd INB CS PWM INA PWM_S

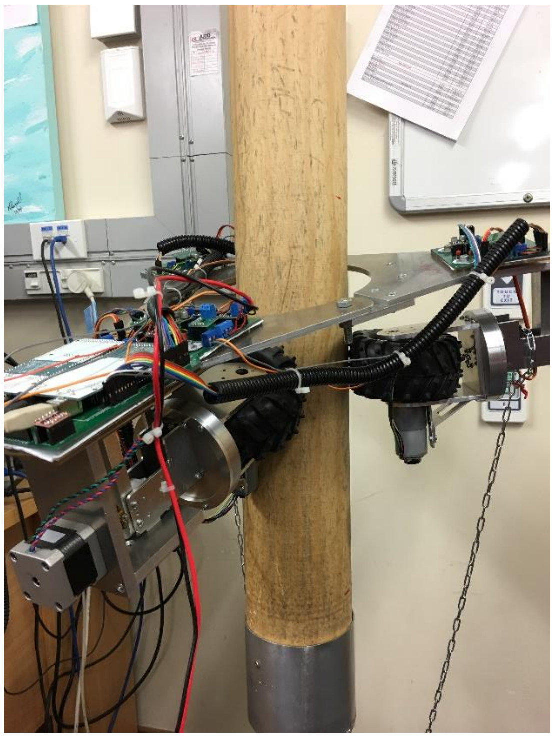




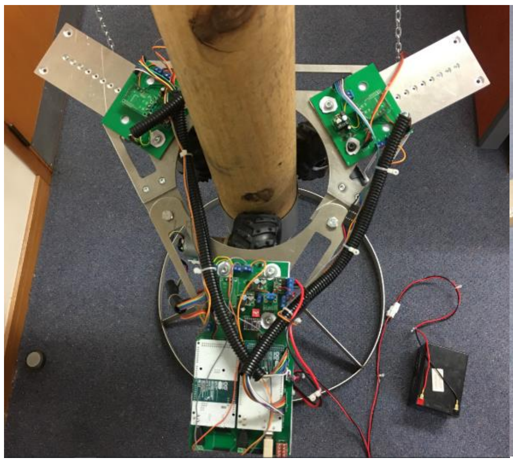
Appendix E – Tree Pruning Robot 3D CAD Model in SolidWorks

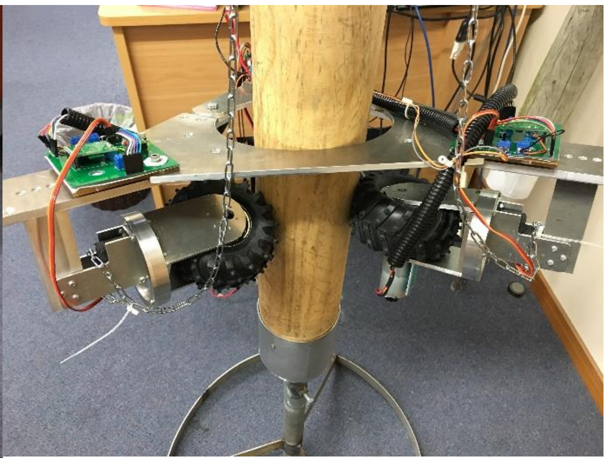


Appendix F – Tree Pruning Robot Prototype









Reference

[1] Hon Jo Goodhew, 2016, "New Zealand Plantation Forest Industry: Facts and Figures 2015/2016," Forest Owners Association & Ministry for Primary Industries, Wellington, New Zealand.

[2] Hon Jo Goodhew, 2015, "New Zealand Plantation Forest Industry: Facts and Figures 2014," Forest Owners Association & Ministry for Primary Industries, Wellington, New Zealand.

[3] Hon Jo Goodhew, 2014, "New Zealand Plantation Forest Industry: Facts and Figures 2012/2013," Forest Owners Association & Ministry for Primary Industries, Wellington, New Zealand.

[4] Hon Nathan Guy, 2013, "New Zealand Plantation Forest Industry: Facts and Figures 2011/2012," Forest Owners Association & Ministry for Primary Industries, Wellington, New Zealand.

[5] H. Kawasaki, S. Murakami, H. Kachi, S. Ueki, 2008, "Novel Climbing Method of Pruning Robot," SICE Annual Conference, Japan, August 20–22, 2008, pp. 160-163.

[6] "A Roadmap for U.S. Robotics: From Internet to Robotics," Georgia Institute of Technology et al., 2013 Edition, March 20, 2013.

[7] "Executive Summary: 1. World Robotics 2013 Industrial Robots 2. World Robotics 2013 Service Robots".

[8] "Executive Summary: 1. World Robotics 2014 Industrial Robots 2. World Robotics 2014 Service Robots".

[9] "Robotics 2020 Multi-Annual Roadmap: for Robotics in Europe," Initial Release B 15/01/2014.

[10] M. J. Spenko, G. C. Haynes et al., 2008, "Biologically Inspired Climbing with a Hexapedal Robot," Journal of Field Robotics, 25(4-5), pp: 223-242.

[11] G. C. Haynes and A. A. Rizzi, 2006, "Gaits and Gait Transitions for legged Robots," Proceedings of the 2006 IEEE International Conference on Robotics and Automation, 15-19 May, Orlando, USA, pp: 1117-1122.

[12] G. C. Haynes and A. A. Rizzi, 2006, "Gait Regulation and Feedback on a Robotic Climbing Hexapod," Robotics: Science and Systems II, 16-19 August, Pennsylvania, USA.

[13] G. C. Haynes, A. Khripin et al., 2009, "Rapid Pole Climbing with a Quadrupedal Robot," IEEE International Conference on Robotics and Automation, May 12-17, Japan, pp. 2767-2772.

[14] Sugano Laboratory, Waseda University, Japan. "Robot Assisting Forestry Work," 2003. http://www.sugano.mech.waseda.ac.jp/project/forest/index.html.

[15] O. Reinoso, R. Aracil, R. Saltaren, 2007, "Using parallel platforms as climbing robots," INDUSTRIAL ROBOTICS: Programming, Simulation and Applications, pp. 663-676.

[16] R. Saltaren, R. Aracil, O. Reinoso et al., 2005, "Climbing Parallel Robot: A computational and Experimental Study of its Performance Around Structural Nodes," IEEE Transactions on Robotics, Vol. 21, No. 6, pp. 1056-1066.

[17] T. L. Lam, Y. S. Xu, 2011, "A Flexible Tree Climbing Robot: Treebot–Design and Implementation," IEEE International Conference on Robotics and Automation, 9-13 May, Shanghai, China, pp. 5849-5854.

[18] T. L. Lam and Y. S. Xu, 2011, "Climbing Strategy for a Flexible Tree Climbing Robot-Treebot," IEEE Transactions on Robotics, Vol. 27, No. 6, pp: 1107-1117.

[19] T. L. Lam and Y. S. Xu, 2011, "Treebot: Autonomous Tree Climbing by Tactile Sensing," IEEE International Conference on Robotics and Automation, 9-13 May, Shanghai, China, pp: 789-794.

[20] S.C. Lau, W.A.F.W. Othman, E.A. Bakar, 2013, "Development of Slider-Crank based Pole Climbing Robot," IEEE International Conference on Control System, Computing and Engineering, Penang, Malaysia, November 29 – 1 December, pp. 471-476.

[21] C. Wright, A. Johnson, A. Peck et al., 2007, "Design of a Modular Snake Robot," Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 29-Nov.2, USA, pp. 2609-2614.

[22] Yisheng Guan et al., 2013, "A Modular Biped Wall-Climbing Robot with High Mobility and Manipulationg Function," IEEE/ASME TRANSACTIONS ON MECHATRONICS, December 2013, Vol.18, No. 6, pp. 1787-1798.

[23] M. Tavakoli et al, 2008, "3DCLIMBER: A climbing robot for inspection of 3D human made structures," 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, September 22-26, pp. 4130-4135.

[24] M. Tavakoli, L. Marques and A. T. Almeida, 2010, "3DCLIMBER: Climbing and manipulation over 3D structures," Mechatronics, 21(2011), pp. 48-62.

[25] M. Nili Ahmadabadi et al., 2005, "Kinematics Modeling of a Wheel-Based Pole Climbing Robot (UT-PCR)," Proceedings of the 2005 IEEE international Conference on Robotics and Automation, Spain, April 2005, pp. 2099-2104.

[26] S. Mahdavi, E. Noohi and M. N. Ahmadabadi, 2007, "Basic movements of a nonholonomic wheel-based pole climbing robot," 2007 IEEE/ASME international conference on advanced intelligent mechatronics, Switzerland, September 4-7, 2007.

[27] M. N. Ahmadabadi et al., 2010, "The Evolution of UT Pole Climbing Robots," 2010 1st International Conference on Applied Robotics for the Power Industry, Canada, October 5-7, 2010.

[28] W. Chonnaparamutt, H. Kawasaki, S. Ueki, et al, 2009, "Development of a Timberjack-like Pruning Robot: Climbing Experiment and Fuzzy Velocity Control," ICROS-SICE International Joint Conference, Japan, August 18-21, 2009, pp. 1195-1199.

[29] Y. Ishigure, K. Hirai and H. Kawasaki, 2013, “A Pruning Robot with a Power-Saving Chainsaw Drive,” Proceedings of 2013 IEEE International Conference on Mechatronics and Automation, Takamatsu, Japan, August 4–7, 2013, pp. 1223-1228.

[30] W. Chonnaparamutt and H. Kawwasaki, 2009, “Fuzzy Systems for Slippage Control of a Pruning Robot,” IEEE International Conference on Fuzzy Systems, South Korea, August 20-24, 2009, pp. 1270-1275.

[31] Seirei Industry Co., http://www.seirei.com/products/fore/b232r/ab232r.html, LED Web Site.

[32] J. Fauroux and J. Morillon, 2010, “Design of a climbing robot for cylindro-conic poles based on rolling self-locking,” Industrial Robot: An International Journal, vol. 37/3, pp. 287–292.

[33] H. Dulimarta and R.L. Tummala, 2002, “Design and control of miniature climbing robots with nonholonomic constrains,” IEEE International Conference on Intelligent Control and Automation, Shanghai, China, June 10-14, 2002, pp. 3267-3271.

[34] A. Nagakubo and S. Hirose, 1994, “Walking and running of the quadruped wall-climbing robot,” IEEE International Conference on Robotics and Automation, San Diego, CA, USA, May 8-13, 1994, pp. 1005-1012.

[35] M. Rachkov, 2002, “Control of climbing robot for rough surfaces,” Proceedings of the Third International Workshop on Robot Motion and Control, Bukowy Dworek, Poland, November 11-11, 2002, pp. 101-105.

[36] Yu Zhisheng, 2009, “Automobile Theory,” China Machine Press, China, ISBN: 9787111020769, 03-2009.

[37] Gérard-Philippe Zéhil, Henri P. Gavin, 2013, “Three-dimensional boundary element formulation of an incompressible viscoelastic layer of finite thickness applied to the rolling resistance of a rigid sphere,” International Journal of Solids and Structures, Volume 50, Issue 6, 15 March 2013, pp. 833-842.

[38] Roberts, 1992, “A guide to estimating the friction of rubber,” Rubber chemistry and technology, 65.3 (1992), pp. 673-686.

[39] David B. Fancher, 1999, “ILB, ILBB, Ferrite Beads,” VISHAY Engineering Note, Document Number 34091.

[40] Charles Coles, 2007, “How to control input ripple and noise in buck converters,” EE Times-India, Feb. 2007.

[41] C. Richardson and R. Bramanpalli, 2015, “Selecting and Using Ferrite Beads for Ringing Control in Switching Converters,” WÜRTH ELEKTRONIK APPLICATION NOTE, 15 Feb. 2015.

[42] Robert Keim, 2015, “Clean Power for Every IC, Part 3: Understanding Ferrite Beads,” Sept. 30, 2015, https://www.allaboutcircuits.com/technical-articles/clean-power-for-every-ic-part-3-understanding-ferrite-beads/.

[43] Kent Walters, “How to Select Transient Voltage Suppressors,” MicroNote 125, Rev. 1.0, 7/99.

[44] Steve hayes, 2017, “What’s the difference? EMC vs EMI,” https://www.element.com/nucleus/2017/07/18/10/46/whats-the-difference-emc-vs-emi.

[45] L. Shawneh and M. A. Jarrah, 2008, “Development and calibration of low cost MEMS IMU for UAV applications,” Proceeding of the 5th International Symposium on Mechatronics and its Applications, Amman, Jordan, May 27-29, 2008.

[46] S. Sukkarieh, E. M. Nebot and H. F. Durrant-Whyte, 1999, “A High Integrity IMU/GPS Navigation Loop for Autonomous Land Vehicle Applications,” IEEE Transactions on Robotics and Automation, vol. 15, no. 3, June 1999, pp. 572 – 578.

[47] F. Alam, Z. Z. He and H. J. Jia, 2014, “A Comparative Analysis of Orientation Estimation Filters using MEMS based IMU,” 2nd International Conference on Research in Science, Engineering and Technology, March 21-22, 2014 Dubai.

[48] H. Zhou and H. Hu, 2008, “Human motion tracking for rehabilitation – A Survey,” Biomedical Signal Processing and Control, vol. 3, no. 1, pp. 1– 18.

[49] M. Clifford, L. Gomez, 2005, “Measuring Tilt with Low-g Accelerometers,” Freescale Semiconductor Application Note, AN3107, 05, 2005.

[50] Mark Pedley, 2013, “Tilt Sensing Using a Three-Axis Accelerometer,” Freescale Semiconductor Application Note, AN3461, 03, 2013.

[51] M. Euston, P. Coote, R. Mahony, J. Kim, T. Hamel, 2008, “A Complementary Filter for Attitude Estimation of a Fixed-wing UAV,” IEEE/RSJ International Conference on Intelligent Robots and Systems, France, September 22-26, 2008.

[52] Y. F. Ren and X. Z. Ke, 2010, “Particle Filter Data Fusion Enhancements for MEMS-IMU/GPS,” Intelligent Information Management, Feb 2010, pp. 417–421.

[53] H. G. Min and E. T. Jeung, “Complementary Filter Design for Angle Estimation using MEMS Accelerometer and Gyroscope,” www.academia.edu/6261055.

[54] J. F. Vasconcelos, C. Silvestre, P. Oliveira, P. Batista, B. Cardeira, 2009, “Discrete Time-Varying Attitude Complementary Filter,” 2009 American Control Conference, June 10-12, 2009.

[55] S. P. Tseng, W. L. Li, C. Y. Sheng, J. W. Hsu, C. S. Chen, 2011, “Motion and Attitude Estimation Using Inertial Measurements with Complementary Filter,” Proceedings of 2011 8th Asian Control Conference, Taiwan, May 15-18, 2011.

[56] M. Filiashkin and M. Novik, 2012, “Combined Complementary Filter For Inertial Navigation System,” 2012 2nd International Conference “Methods and Systems of Navigation and Motion Control”, October 9-12, 2012 Ukraine, pp. 59-62.

[57] D. Cao, Q. Qu, C. T. Li and L. C. He, 2009, "Research of Attitude Estimation of UAV Based on Information Fusion of Complementary Filter," 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology, IEEE Computer Society, pp. 1290-1293.

[58] F. M. Mirzaei and S. I. Roumeliotis, 2008, "A Kalman Filter-Based Algorithm for IMU-Camera Calibration: Observability Analysis and Performance Evaluation," IEEE Transaction on Robotics, vol.23, no. 5, October 2008.

[59] P. F. Zhang, J. Gu, E. E. Milios, P. Huynh, 2005, "Navigation with IMU/GPS/Digital Compass with Unscented Kalman Filter," Proceedings of the IEEE, International Conference on Mechatronics & Automation, Canada, July 2005.

[60] F. Caron, E. Duflos, D. Pomorski, P. Vanheeghe, 2006, "GPS/IMU data fusion using multisensory Kalman filtering: introduction of contextual aspects," INFORMATION FUSION, 7 (2006), pp. 221–230.

[61] H. Ferdinando, H. Khoswanto and D. Purwanto, 2012, "Embedded Kalman Filter For Inertial Measurement Unit (IMU) on the Atmega8535," 2012 International Symposium on Innovations in Intelligent Systems and Applications, July 2-4.

[62] S. H. Won, W. Melek and F. Golnaraghi, 2008, "Position and Orientation Estimation Using Kalman Filtering and Particle Filtering with One IMU and One Position Sensor," 34th Annual Conference of IEEE on Industrial Electronics, November 10-13, 2008.

[63] S. Sabatelli, M. Galgani, L. Fanucci and A. Rocchi, 2012, "A double stage Kalman filter for sensor fusion and orientation tracking in 9D IMU," Sensors Applications Symposium, February 7-9, 2012.

[64] Walter T. Higgins, 1975, "A Comparison of Complementary and Kalman Filtering," IEEE Transaction on Aerospace and Electronic Systems, vol. AES–11, no. 3, May 1975, pp. 321–325.

[65] Shane Colton, 2007, "The Balance Filter," Massachusetts Institute of Technology, Tech. Rep., June 25, 2007.

[66] G. Welch and G. Bishop, 2006, "An Introduction to the Kalman Filter," UNC-Chapel Hill, TR 95-041, July 24, 2006.

[67] Pengfei Gui, Liqiong Tang and Subhas Mukhopadhyay, 2015, "A novel design of anti-falling mechanism for tree pruning robot," 2015 IEEE 10th conference on Industrial Electronics and Applications (ICIEA), Auckland, New Zealand, 15-17 June 2015, pp. 812-816.

[68] Pengfei Gui, Liqiong Tang and Subhas Mukhopadhyay, 2015, "MEMS based IMU for tilting measurement: Comparison of complementary and Kalman filter based data fusion," 2015 IEEE 10th conference on Industrial Electronics and Applications (ICIEA), Auckland, New Zealand, 15-17 June 2015, pp. 812-816.

[69] Winai Chonnaparamutt and Haruhisa Kawasaki, 2009, "Fuzzy Systems for Slippage Control of a Pruning Robot," IEEE International Conference on Fuzzy Systems, South Korea, August 20-24, 2009, pp. 1270-1275.

[70] Pierre Lamon, Ambroise Krebs, et al., 2004, "Wheel torque control for a rough terrain rover," IEEE International Conference on Robotics and Automation, New Orleans, USA, April 2004, pp. 4682-4687.

[71] Chris C. Ward and Karl Iagnemma, 2008, "A Dynamic-Model-Based Wheel Slip Detector for Mobile Robots on Outdoor Terrain," IEEE Transactions on Robotics, Vol. 24(4), August 2008, pp. 821-831.

[72] Giulio Reina, et al., 2008, "Vision-based Estimation of Slip Angle for Mobile Robots and Planetary Rovers," IEEE International Conference on Robotics and Automation, Pasadena, USA, May 19-23, 2008, pp. 486-491.

[73] Mohamed S. Masmoudi, et al., 2016, "Fuzzy logic controllers design for omnidirectional mobile robot navigation," Applied Soft Computing, Vol. 49, December 2016, pp. 901-919.

[74] Kevin M. Passino and Stephen Yurkovich, "Fuzzy Control," Addison Wesley Longman, Inc., California, USA. ISBN: 0-201-18074-X, pp. 21-94.

[75] E.H. Mamdani, 1974, "Application of fuzzy algorithms for control of simple dynamic plant," Proceedings of the Institution of Electrical Engineers, Control and Science, Vol. 121, No. 12, December 1974, pp. 1585-1588.

[76] Pengfei, G., Liqiong, T., and Subhas, M., 2017, "A Novel Robotic Tree Climbing Mechanism with Anti-Falling Functionality for Tree Pruning," ASME Journal of Mechanisms and Robotics, 10(1), pp. 014502-1 – 014502-8.

[77] Harifi, A., Aghagolzadeh, A., Alizadeh, G., and Sadeghi, M., 2008, "Designing a sliding mode controller for slip control of antilock brake systems," Transportation Research Part C, 16, pp. 731-741.

[78] Velimir Ćirović, Dragan Aleksendrić, Dušan Smiljanić, 2013, "Longitudinal wheel slip control using dynamic neural networks," Mechatronics, 23, pp. 135-146.

[79] Velimir Ćirović, Dragan Aleksendrić, 2013, "Adaptive neuro-fuzzy wheel slip control," Expert Systems with Applications, 40, pp. 5197-5209.

[80] Lei Yuan, Haiyan Z., Hong Chen, and Bingtao R., 2016, "Nonlinear MPC-based slip control for electric vehicles with vehicle safety constraints," Mechatronics, 38, pp. 1-15.

[81] Pranav, P. K., Tewari, V. K., Pandey, K. P., and Jha, K. R., 2012, "Automatic wheel slip control system in field operations for 2WD tractors," Computers and Electronics in Agriculture, 84, pp. 1-6.

[82] Daisuke, C., Hajime, A., Kuniaki, K., Hayato, K., and Taketoshi, M., 2006, "Plural Wheels Control based on Slip Estimation," IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China, Oct. 9-15, pp. 5558- 563.

[83] Pierre, L., Ambroise, K., Michel, L., and Roland, S., 2004, "Wheel torque control for a rough terrain rover," IEEE International Conference on Robotics and Automation, New Orleans, LA, USA, April 26 – May 1, pp. 4682-4687.

[84] William, P., Antonio, L., and Mathieu, G., 2012, “Design and experimental validation of a nonlinear wheel slip control algorithm,” Automatica, 48, pp. 1852-1859.

[85] Hossein, M., and Mehdi, M., 2010, “A novel method for non-linear control of wheel slip in anti-lock braking systems,” Control Engineering Practice, 18, pp. 918-926.

[86] Klaus, K., and Jörg-Michael, H., 1995, “An Embedded Fuzzy Anti-Slippage System for Heavy Duty Off Road Vehicles,” Information Sciences, 4, pp. 1-27.

[87] Haibo, G., Kerui, X., Liang, D., Zongquan, D., Zhen, L., and Guangjun, L., 2015, “Optimized control for longitudinal slip ratio with reduced energy consumption,” Acta Astronautica, 115, pp. 1-17.

[88] Valentin, I., Dzmitry, S., Klaus, A., Phil, B., Bernhard, K., and Josef, Z., 2015, “Wheel slip control for all-wheel drive electric vehicle with compensation of road disturbances,” Journal of Terramechanics, 61, pp. 1-10.

[89] Giulio, R., Genya, I., Keiji, N., and Kazuya, Y., 2008, “Vision-based Estimation of Slip Angle for Mobile Robots and Planetary Rovers,” IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, May 19-23, pp.486-491.

[90] Mingyuan, B., Long, C., Yugong, L., and Keqiang, L., 2014, “A Dynamic Model for Tire/Road Friction Estimation under Combined Longitudinal/Lateral Slip Situation,” The Society of Automotive Engineers’ (SAE) 2014 World Congress & Exhibition, Detroit, Michigan, USA, April 8-10.

[91] Martin T. Hagan et al, 2014, “Neural Network Design (2nd Edition),” Martin Hagan, ISBN-10: 0-09717321-1-6, 09-2014.