DESIGN AND EXPERIMENTAL VALIDATION OF 6 COMPONENT WIND TUNNEL FORCE BALANCE

EYÜP GÖZÜKARA

AUGUST 2025

ÇANKAYA UNIVERSITY

GRADUATE SCHOOL
MECHANICAL ENGINEERING DEPARTMENT
MECHANICAL ENGINEERING MASTER'S THESIS

DESIGN AND EXPERIMENTAL VALIDATION OF 6 COMPONENT WIND TUNNEL FORCE BALANCE

EYÜP GÖZÜKARA

AUGUST 2025

ABSTRACT

DESIGN AND EXPERIMENTAL VALIDATION OF 6 COMPONENT WIND TUNNEL FORCE BALANCE

GÖZÜKARA, EYÜP
MECHANICAL ENGINEERING MASTER'S THESIS

Supervisor: Prof. Dr. Sıtkı Kemal İDER
Co-Supervisor: Assoc. Prof. Dr. Ekin ÖZGİRGİN YAPICI
August 2025, 151 pages

Fluid mechanics, a crucial domain in engineering science, underpins numerous engineering principles and necessitates the resolution of the nonlinear and intricate Navier-Stokes equations for flow analysis. Owing to the complexity of these equations, experimental and numerical methodologies are important for investigating the influence of fluids on objects and for precise measurement of these effects. In this area, wind tunnels are often used to create a controlled flow setting that is important for studying how three-dimensional models behave in flow and learning about flow characteristics.

The aim of this study is to design a force balance mechanism capable of measuring forces along six axes for use in wind tunnels. This mechanism will precisely measure lift, drag, and lateral forces, as well as pitching, roll, and yaw moments, on models placed in the test chamber of subsonic wind tunnels. A Stewart Platform-type balance mechanism with high crosstalk sensitivity will be used in the design process.

The experimental apparatus will accurately quantify the lift, drag, and lateral forces of three-dimensional models, in addition to the torque produced on each axis. The system orientation can be readily altered by computer commands and buttons, with measurement data transmitted to concurrent digital displays and logged over time.

This feature will facilitate the generation of significant graphs post-experiment and enhance the precision of data utilized in scientific research.

During this study, various force balance mechanisms in the literature were investigated, and a unique design approach was adopted. A system inspired by the Stewart platform, yet possessing a different degree of freedom, was developed for use within an experimental chamber. Designed for this system, a control algorithm allows the electromechanical components to run in line with the algorithm, therefore enabling the user to change the orientation of the model housed within the wind tunnel chamber.

Load cells integrated into the system enabled the measurement of aerodynamic forces acting on the model. Sensor outputs were analyzed and converted into meaningful force components. This method ensures the accuracy of force measurements and increases the reliability of experimental results.

Following the research, the system was exhaustively assessed. System accuracy was found by extensive study and assessments using position control and measurement. Multiple tests were conducted on the system to see whether it could remain still or move under control criteria. Calibration of sensors and control systems has been carefully done to minimize the margin of error in practical applications.

Keywords: Wind tunnel, Stewart platform, Force balance, Sensor fusion, Multi-axial force measurement

ÖZET

6 BİLEŞENLİ RÜZGAR TÜNELİ KUVVET DENGE SİSTEMİ TASARIMI VE DENEYSEL DOĞRULAMASI

GÖZÜKARA, EYÜP
MAKİNE MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ

Danışman: Prof. Dr. Sıtkı Kemal İDER
Ortak Danışman: Doç. Dr. Ekin ÖZGİRGİN YAPICI
Ağustos 2025, 151 sayfa

Akışkanlar mekaniği, mühendislik biliminin temel bir alanı olup, birçok mühendislik ilkesinin temelini oluşturur ve akış analizleri için doğrusal olmayan ve karmaşık Navier-Stokes denklemlerinin çözülmesini gerektirir. Bu denklemlerin karmaşıklığı nedeniyle, akışkanların nesneler üzerindeki etkilerini incelemek ve bu etkileri hassas bir şekilde ölçmek için deneysel ve sayısal yöntemler büyük önem taşımaktadır. Bu alanda, üç boyutlu modellerin akış altındaki davranışlarını incelemek ve akış karakteristiğini anlamak için kontrollü bir akış ortamı sağlayan rüzgar tünelleri yaygın olarak kullanılmaktadır.

Bu çalışmanın amacı, rüzgar tünellerinde kullanılabilecek altı eksende kuvvet ölçümü gerçekleştirebilen bir kuvvet denge mekanizması tasarlamaktır. Bu mekanizma, ses altı rüzgar tünellerinin test bölmesine yerleştirilen modeller üzerindeki kaldırma, sürükleme ve yanal kuvvetleri, ayrıca yunuslama, yuvarlanma ve sapma momentlerini hassas bir şekilde ölçebilecektir. Tasarım sürecinde, tek eksenli ve üç eksenli yük hücreleri ile yüksek crosstalk hassasiyetine sahip Stewart Platform tipi bir denge mekanizması kullanılacaktır.

Geliştirilecek deney düzeneği, üç boyutlu modellerin kaldırma, sürükleme ve yanal kuvvetlerini, bunlara ek olarak her eksendeki torkları doğru bir şekilde ölçebilecektir. Sistem oryantasyonu, bilgisayar komutları ve butonlar yardımıyla

kolayca değiştirilebilecek, ölçüm verileri eş zamanlı olarak dijital ekranlara aktarılacak ve zaman bazlı olarak kaydedilecektir. Bu özellik, deney sonrasında anlamlı grafiklerin oluşturulmasını kolaylaştıracak ve bilimsel araştırmalarda kullanılan verilerin doğruluğunu artıracaktır.

Bu çalışma sırasında, literatürde yer alan çeşitli kuvvet denge mekanizmaları incelenmiş ve özgün bir tasarım yaklaşımı benimsenmiştir. Stewart platformundan esinlenilmiş ancak farklı serbestlik derecesine sahip bir sistem, deney bölmesi içinde kullanılmak üzere geliştirilmiştir. Bu sistem için tasarlanan kontrol algoritması, elektromekanik bileşenlerin algoritma doğrultusunda çalışmasını sağlayarak kullanıcının rüzgar tüneli içinde bulunan modelin oryantasyonunu değiştirmesine olanak tanımaktadır.

Sisteme entegre edilen yük hücreleri, model üzerinde etki eden aerodinamik kuvvetlerin ölçülmesini mümkün kılmıştır. Sensör girişleri analiz edilerek anlamlı kuvvet bileşenlerine dönüştürülmüştür. Bu yöntem, kuvvet ölçümlerinin doğruluğunu garanti altına almakta ve deney sonuçlarının güvenilirliğini artırmaktadır.

Çalışma sonucunda, geliştirilen sistem detaylı bir şekilde değerlendirilmiştir. Sistem doğruluğu, kapsamlı testler ve ölçümler ile incelenmiş, pozisyon kontrolü ve ölçüm hassasiyeti belirlenmiştir. Sistem, kontrol kriterleri altında sabit kalma veya hareket etme yeteneğini test etmek için birçok deneyden geçirilmiştir. Sensörlerin ve kontrol sistemlerinin kalibrasyonu, pratik uygulamalarda hata payını en aza indirmek için dikkatlice yapılmıştır.

Anahtar Kelimeler: Rüzgar tüneli, Stewart platformu, Kuvvet denge sistemi, Sensör füzyonu, Çok eksenli kuvvet ölçümü

ACKNOWLEDGEMENT

Many people have contributed to my thesis writing and the success I've achieved in my education, but among these individuals, there are some who have been profoundly influential not only in my academic life but also in my personal development. While I don't know how to repay this support, I feel an immense sense of gratitude. It's a source of great pride for me to have embraced the experiences, ideas, and perspectives of these individuals who have left their mark on my life, and to have followed their path.

The person who has been with me most on this journey and contributed most to my writing this thesis is my beloved wife, Aybüke Öztürk Gözükara. She is not only my life partner, but also a guiding light when times were tough, a source of hope when I felt hopeless, and a unique person who kept me moving forward. The writing of this thesis coincided with those early years of our newlywed life, as we were building our lives. While I may have neglected many things during this time, my beloved wife always filled this gap. She made our house a warm and welcoming home in which we would spend years, and she handled so much on her own throughout this process. Without her, I doubt I would have been able to complete this thesis.

I have endless respect and gratitude to all the academic staff at Çankaya University's Mechanical Engineering faculty from 2017 to 2025. Thanks to this team, my undergraduate and graduate studies were fulfilling and I gained a wealth of experience. This team has also contributed significantly to my current career. Of course, the people I must first and foremost thank my advisor, Prof. Dr. Sıtkı Kemal İder, and my co-advisor, Assoc. Prof. Dr. Ekin Özgirgin Yapıcı. They never failed to support me and provided invaluable assistance whenever I needed it. Thanks to their experience, I was able to find the paths that would lead me back in the right direction, even though I sometimes got lost.

Finally, I would like to thank my colleagues at work. They had my back at many times during my thesis writing process and supported me with my responsibilities at work while I was working on my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOLS

λ	: Maximum freedom in selected space dimensions
l	: Total number of links
j	: Total number of joints
f_i	: The degree of freedom provided by each joint
Point P	: Point of aerodynamic forces affected
Force P	: Aerodynamic resultant force
R0	: Reaction force at origin
R1	: Reaction force at joint 1
R2	: Reaction force at joint 2
R3	: Reaction force at joint 3
d	: Distance
$\overrightarrow{r_1}$	: Position vector to the point A
$\overrightarrow{r_2}$	: Position vector to the point B
$\overrightarrow{r_3}$	: Position vector to the point C
$\overrightarrow{r_p}$	: Position vector to the point P
$\overrightarrow{r_{cg}}$	: Position vector to the center of gravity of model
$\overrightarrow{M_o}$	: Moment around origin
$\overrightarrow{M^G}$	: Applied moment based on ground coordination system
$\vec{M}$	: Applied moment based on top plate coordination system
T	: Transformation matrix
A	: Ball joint A center point
B	: Ball joint B center point
C	: Ball joint C center point
A_g	: Ball joint A ground center point
B_g	: Ball joint B ground center point
C_g	: Ball joint C ground center point

ABBREVIATIONS

UAV	: Unmanned Aerial Vehicle
μN	: micro-Newtons
kN	: kilo-Newtons
N	: Newtons
μNm	: micro-Newton-meters
Nm	: Newton-meters
SPS	: Spherical-Prismatic-Spherical
UPS	: Universal-Prismatic-Spherical
I&I	: Immersion and Invariance
DOF	: Degree-of-freedom
ROM	: Range of motion
VR	: Virtual reality
IMU	: Inertial Measurement Unit
mm	: millimeter
m	: meter
FBD	: Free body diagram
CM	: Center of mass
kgf	: kilogram-force
kg	: kg
%FS	: Percentage based on full scale
GND	: Ground
DC	: Direct current
VDC	: Direct current voltage
PWM	: Pulse width modulation
I2C	: Inter-Integrated Circuit
SPI	: Serial Peripheral Interface
SDA	: Serial data line
SCL	: Serial clock line
EMC	: Electromagnetic compatibility

CHAPTER I
LITERATURE REVIEW

1.1 INTRODUCTION

People often use wind tunnels to test the shapes of airplane wings, the designs of their fuselages, and how well they fly. These tests help make sure that planes can take off, land, cruise, and turn more smoothly. Wind tunnels are also used to test the aerodynamic forces that spaceships may face when they enter the atmosphere [1].

Figure 1: The Mercury re-entry capsule in wind tunnel test at NASA Langley Research Center in 1959 [1].

Wind tunnels are used to make cars sleeker and get the most out of their gas mileage. These studies are done on car designs to make them more stable, lower their air resistance, and cut down on wind noise. [2]. Optimizing the aerodynamics of fast cars like Formula 1 cars is very important.

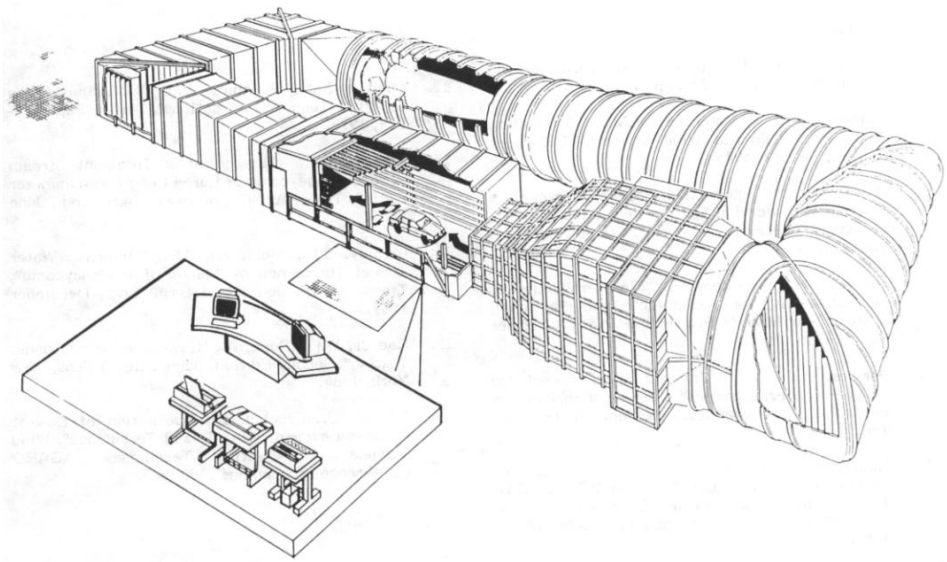

Figure 2: Design of the volvo wind tunnel [2].

Wind tunnels are often used to evaluate the wind resistance of skyscrapers and bridges. These tests are very important for making sure that buildings are safe and stable [3]. With this in mind, it is possible to simulate the effects of natural events like earthquakes and storms in these tunnels.

Wind tunnel tests are performed to enhance the aerodynamic efficiency of athletes and sporting apparatus. Tests are performed in these tunnels to improve the performance of sportsmen, including bikers, skiers, and swimmers (Figure 3) [4]. Furthermore, the aerodynamic drag of race cars, bicycles, and various sports apparatus is refined.

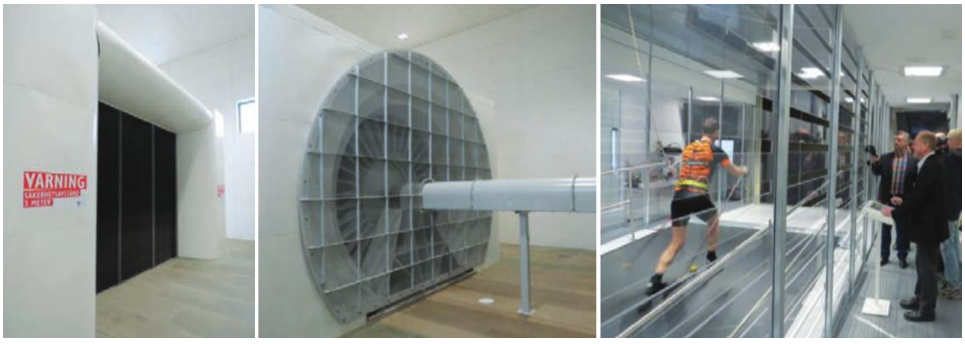

Figure 3: Wind tunnel test of ski athlete [4].

Wind tunnels are utilized for the aerodynamic evaluation of military vehicles, missiles, and unmanned aerial vehicles (UAVs). Evaluating the aerodynamic efficacy

of military systems, particularly those functioning at elevated velocities, is essential in defensive initiatives. Figure 4 [5].

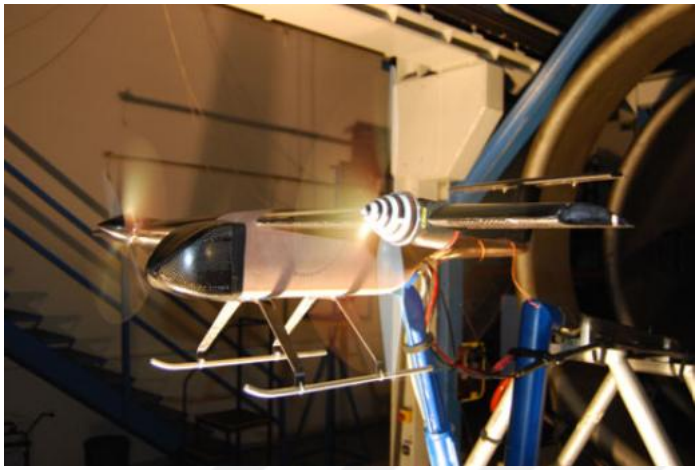

Figure 4: Wind tunnel UAV examination [5].

Wind tunnels are utilized in the design of sailing yachts and big marine crafts. Aerodynamic analyses are performed to enhance the interactions between sailing vessels and the wind [6].

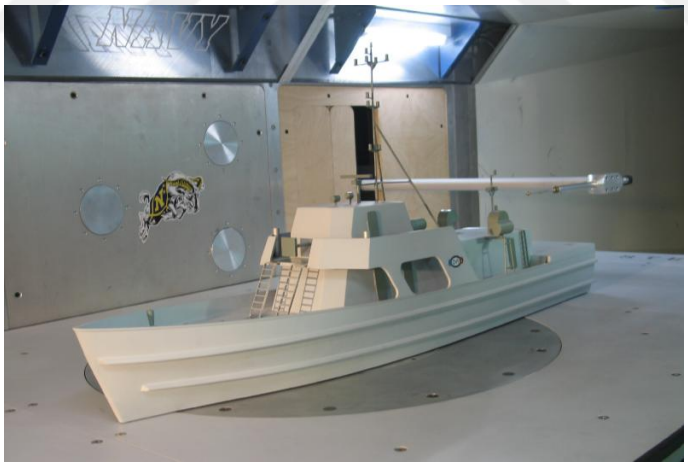

Figure 5: YP676 model vessel examination at 4% scale [6].

Wind tunnels are used to optimize wind turbine design and efficiency. In these tunnels, the aerodynamic performance of turbine blades can be evaluated in order to maximize energy production capacity [7].

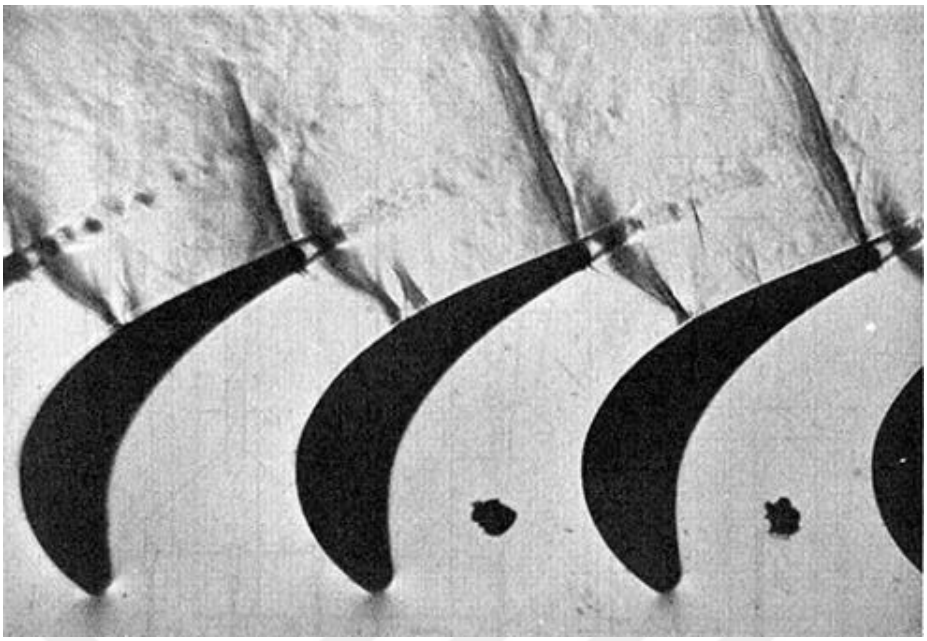

Figure 6: Flow visualization of turbine blades [7].

1.2 FORCE TRANSDUCERS IN WIND TUNNELS

In wind tunnels, force sensors are very important for measuring aerodynamic forces and moments very accurately. You can use these sensors to find out the model's lift, drag, and side forces. Most people choose metal strain gauges as sensors because they can accurately turn mechanical strains into electrical signs. In wind tunnels, force transducers built into internal balance systems keep the model from being affected by outside forces, which lets measurements be more exact. These sensors are used to evaluate the aerodynamic performance of airplanes, vehicle prototypes, and other constructions. Furthermore, because of their nonlinear features, semiconductor-based transducers are less commonly utilized to record dynamic and rapid force changes than metal strain gauges.

The development of force transducers (power converters) has a lengthy history, beginning with fundamental physical principles and progressing to modern strain gauge technology. The process started in 1678 when Robert Hooke came up with the law of elasticity. This law helped us understand how stress and bending are connected. Hooke's work led to many of the tools we use today to measure strain and test materials. Georg Simon Ohm helped with measuring electrical resistance in the early 1800s by showing how voltage, current, and resistance are connected in electrical systems [8]. Charles Wheatstone later devised a method for detecting minor electrical resistances based on Samuel Hunter Christie's Wheatstone Bridge circuit, which was

invented in 1833 [9]. It was during this time that William Thomson (Lord Kelvin) did more research on how mechanical strain affects electrical resistance [8].

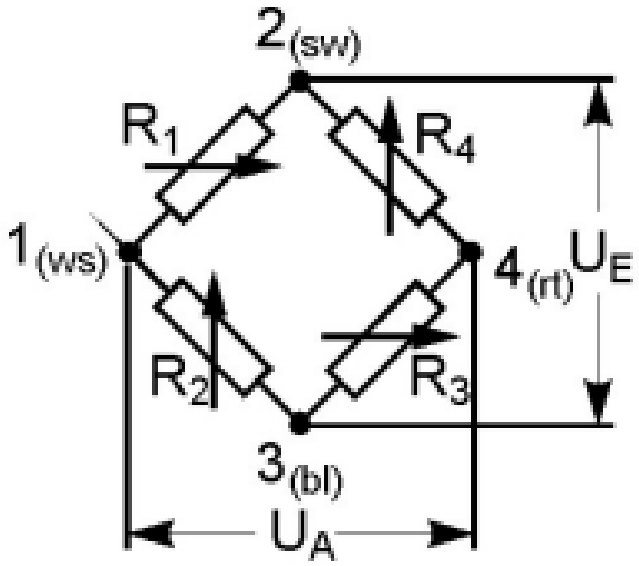

Figure 7: Wheatstone bridge circuit [9].

In the 1950s, P. Jackson pioneered the photochemical etching technique in the manufacturing of metal strain gauges, facilitating their mass production. This advancement, by lowering expenses and augmenting production capacity, facilitated the broader application of strain gauges. Companies such as Baldwin-Southwark Company and Hottinger Baldwin Messtechnik (HBM) have innovated the industrial manufacturing of strain gauges, facilitating their extensive application in sectors including aerospace, automotive, and wind tunnel testing [8].

In wind tunnel tests, metal strain gauges are often used to get accurate readings of force and moment. In dynamic tests, however, semiconductor strain gauges (Figure 8) have been chosen because they respond more quickly. However, semiconductors' non-linear properties and temperature sensitivity have limited their widespread application [10]. Today, strain gauge technology is one of the most important tools for measuring force with great accuracy.

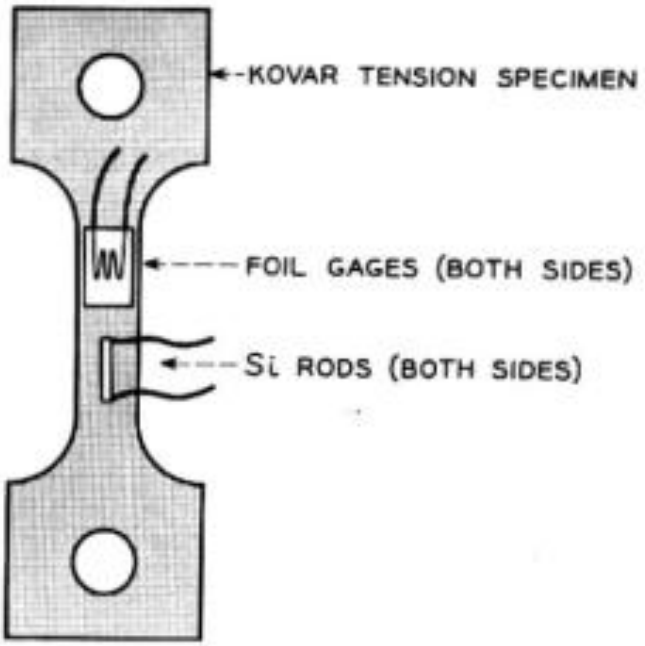

Figure 8: Semiconductor strain gauge [10].

Currently, strain measurement is predominantly conducted using load cells. Load cells are regarded as dependable instruments for measuring force and moment, extensively utilized throughout diverse sectors, particularly in robotics, agriculture, medicine, and industrial weighing.

Load cells are categorized into many types according to force measurement requirements. The fundamental classification consists of uniaxial, biaxial, and triaxial load cells. Uniaxial load cells detect force along a single axis, whereas biaxial and triaxial load cells offer two- or three-dimensional force measurements [11].

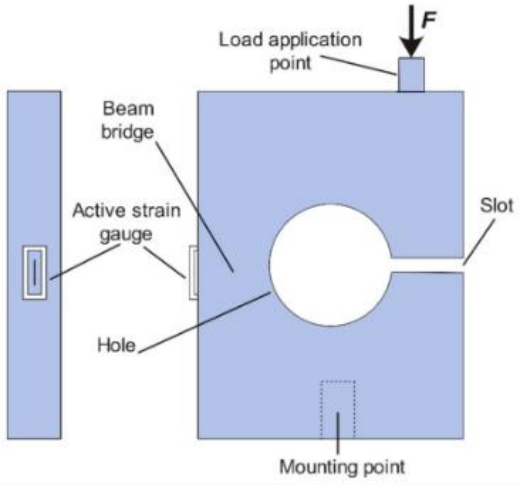

Figure 9: Uniaxial load cell [11].

Three-axis load cells, seen in Figure 10, are frequently employed in robotic and biomechanical applications, especially in contexts requiring accurate and precise force monitoring. The interactions between several axes in multi-axis load cells are termed crosstalk. The crosstalk value must be minimized through optimizations in the load cell design. The precision of the measurements is crucial in this setting (Figure 11) [11, 12].

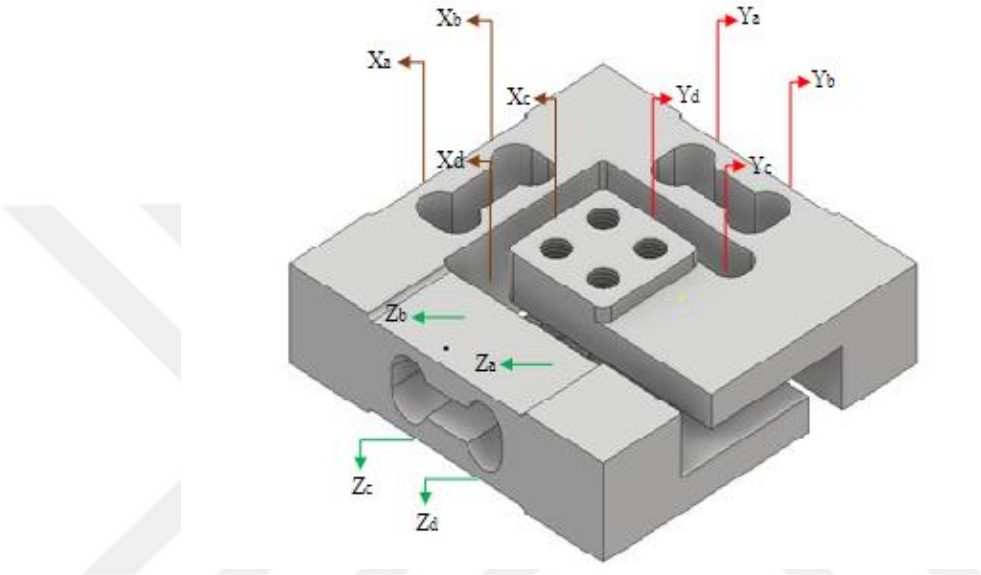

Figure 10: Design of a three-axis load cell [12].

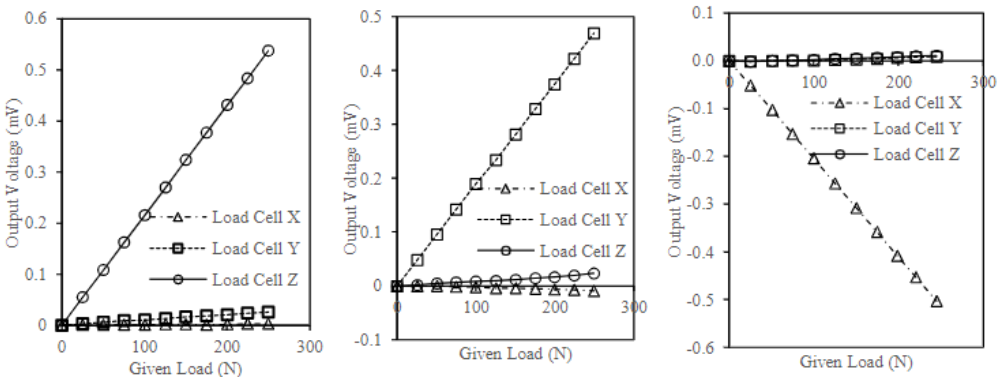

Figure 11: Cross-axis output interaction (crosstalk) of the multiaxis load cell [12].

A prevalent variety of load cell is the bending beam load cell. Most platform scales use this type because it measures force by how much the beam bends. On the other hand, tension and compression load cells can measure both tensile and compressive forces and are often used in devices that test materials. Furthermore,

diaphragm load cells are used to measure pressure, whereas ring load cells are utilized to measure high-capacity force.

For more specialized applications, tiny load cells are used to measure small forces, while torque load cells are used to measure torque. Furthermore, hydraulic and pneumatic load cells are widely used in heavy industrial settings [11, 12] and measure force using liquid or gas pressure.

1.3 BALANCE SYSTEMS FOR WIND TUNNELS

The balance systems employed in wind tunnels guarantee the accurate measurement of forces and moments on the model during aerodynamic evaluations. These systems are categorized according to both internal and external configurations. External balance systems are situated outside the wind tunnel, whilst internal balance systems are incorporated within the model. The appropriate choice of balancing type is essential for precise measurement of aerodynamic loads in wind tunnel experiments [8].

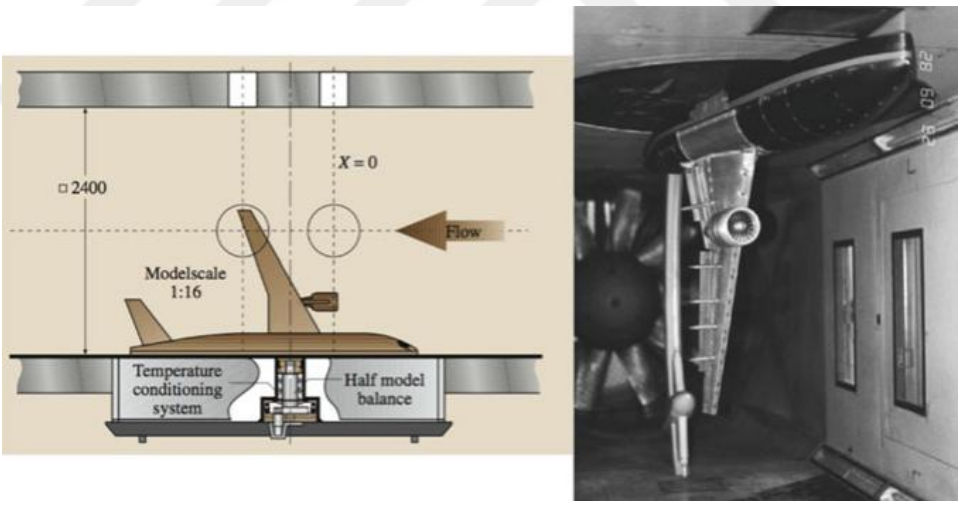

Figure 12: Semi-model equilibrium system [8].

1.3.1 External Balance Systems

External balances are apparatus located outside the wind tunnel test section and typically necessitate additional room. These balances quantify the pressures and moments exerted by the tested model across various axes. External balances are categorized into distinct groups according to their structures, each with unique advantages and disadvantages [8].

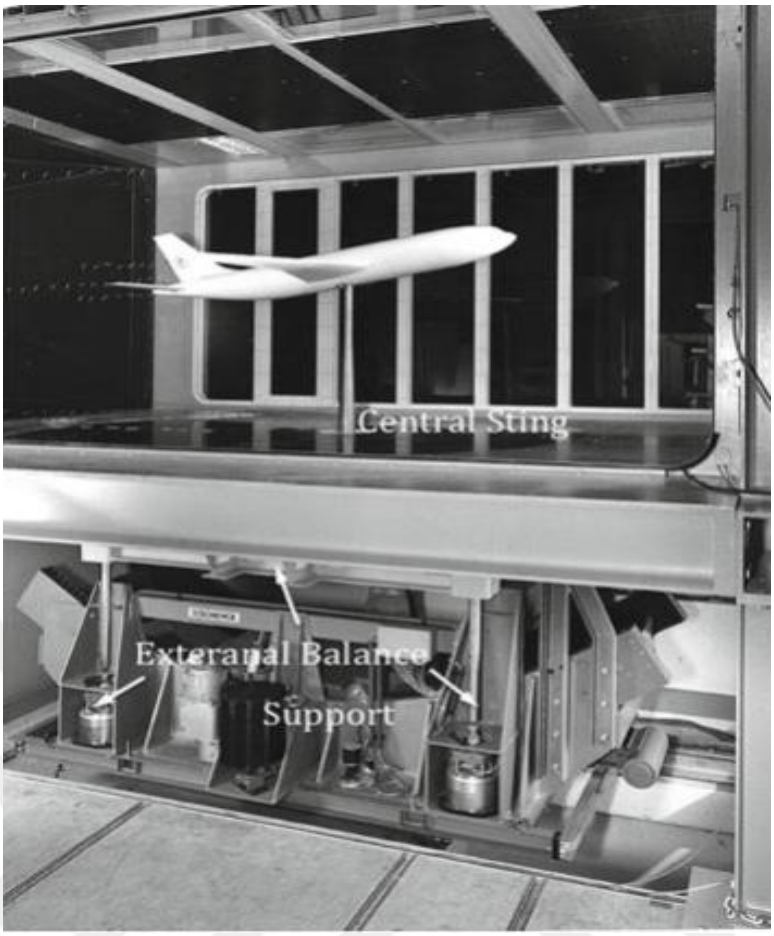

Figure 13: External balance system [8].

1.3.2 Weigh Beam Balances

Beam balance scales are one of the most ancient forms of external balance systems. They are constructed according to the principles of mechanical beam balances utilized in ancient marketplaces. These balances assess by equilibrating the forces and moments on the model. Beam balance scales quantify aerodynamic loads through six distinct beams, with each beam corresponding to various force components within the wind tunnel. Nevertheless, the wire support systems connecting the beams and the model result in a low natural frequency, potentially leading to sluggish measurements. Historically, the beams were balanced and read manually; however, contemporary versions have automated these operations [8].

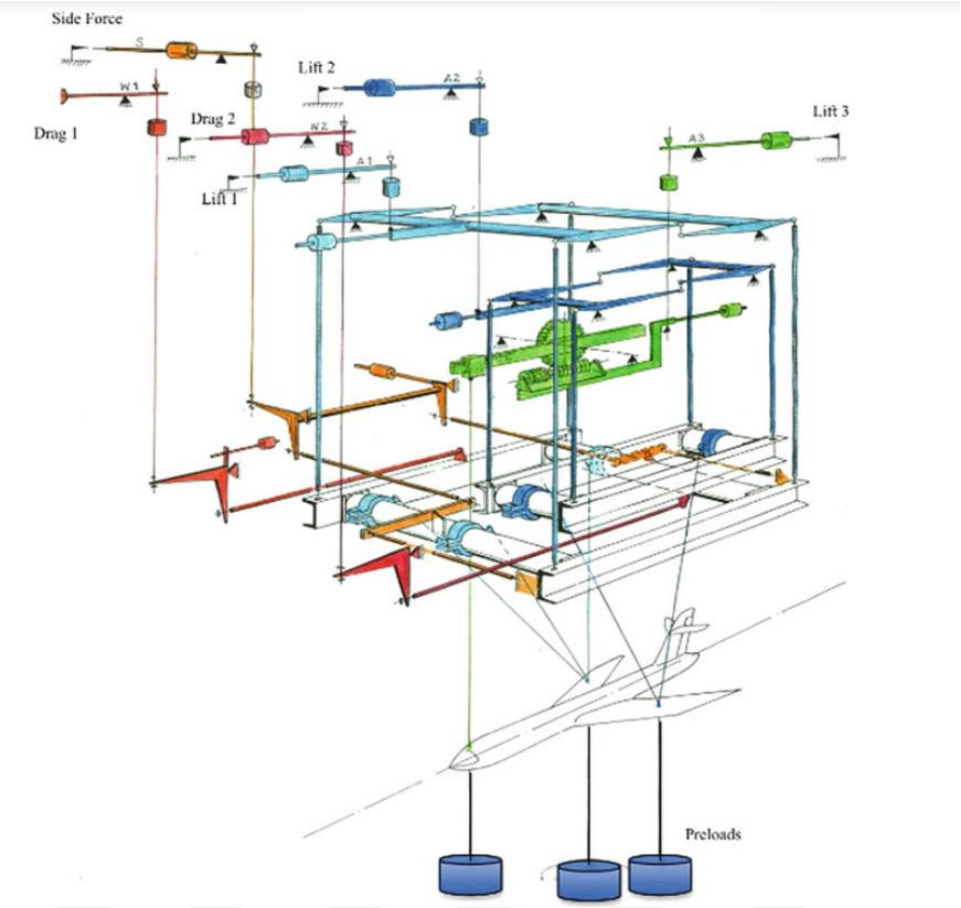

Figure 14: Schematic of weigh beam balance [8].

1.3.3 Pyramid Balances

Pyramid balances (Figure 15) derive its name from the four slanted rods in their construction that resemble a pyramid form. The fundamental operating principle of these balances involves the concentration of forces (X, Y, Z) at a designated reference point and the assessment of tensile or compressive forces via the bars. Load cells are positioned on these bars to measure forces with exceptional accuracy. Pyramid balances are distinguished by their ability to minimize interactions among forces [8]. Thus, forces and moments are nearly entirely decoupled, with signals immediately linked to force and moment components, facilitating measurements without intricate calculations.

The drawback of pyramid balances is their structural intricacy. The accurate alignment and careful installation of the rods and load cells are crucial. Minor alignment discrepancies might adversely impact the precise delineation of forces and moments, hence augmenting calibrating demands. Pyramid balances provide an

optimal solution for high-precision assessments requiring the separate measurement of forces and moments.

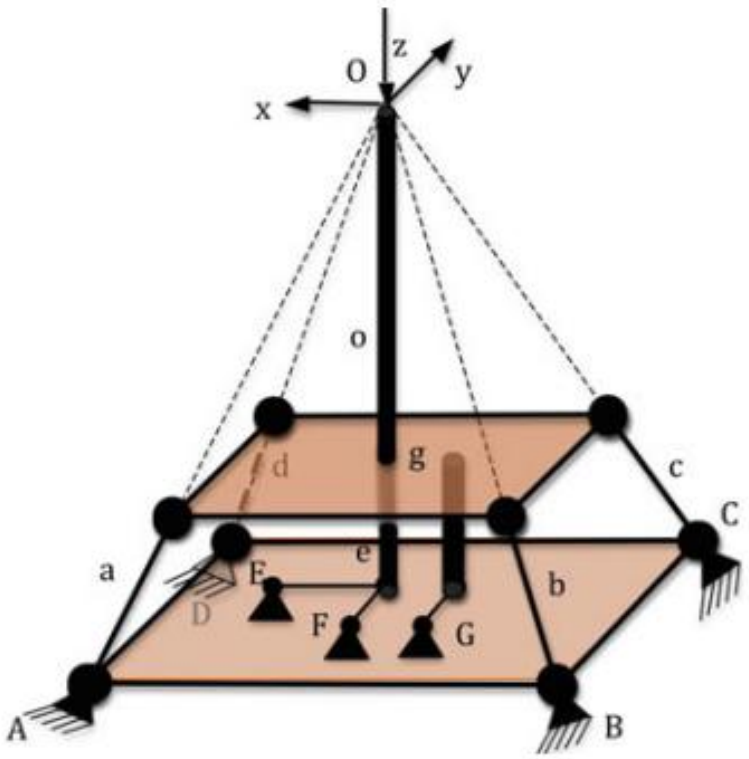

Figure 15: Pyramid balance system [8].

Platform balances (Figure 16) are one of the most prevalent forms of external balances now in use. This system, characterized by a simplified design, quantifies the pressures and moments exerted by the model positioned on a platform. Load cells monitor forces via rods connected to the platform, and they are designed to assess forces along designated axes. The utilization of load cells with flexible joints facilitates the decoupling of forces on the platform, hence ensuring precise measurements.

The primary distinction of platform balances is that force and moment values are derived from the integration of signals from several load cells. This design determines a force component through the integration of information from several load cells, resulting in interaction among certain components. These interactions must be meticulously assessed and analyzed during calibration. Platform balances can often measure six force components concurrently, rendering them optimal for comprehensive model testing. Moreover, platform balances are utilized in side wind

and crosswind assessments. Owing to its exceptional accuracy and extensive applicability, platform balances have become essential in contemporary wind tunnel experiments [8].

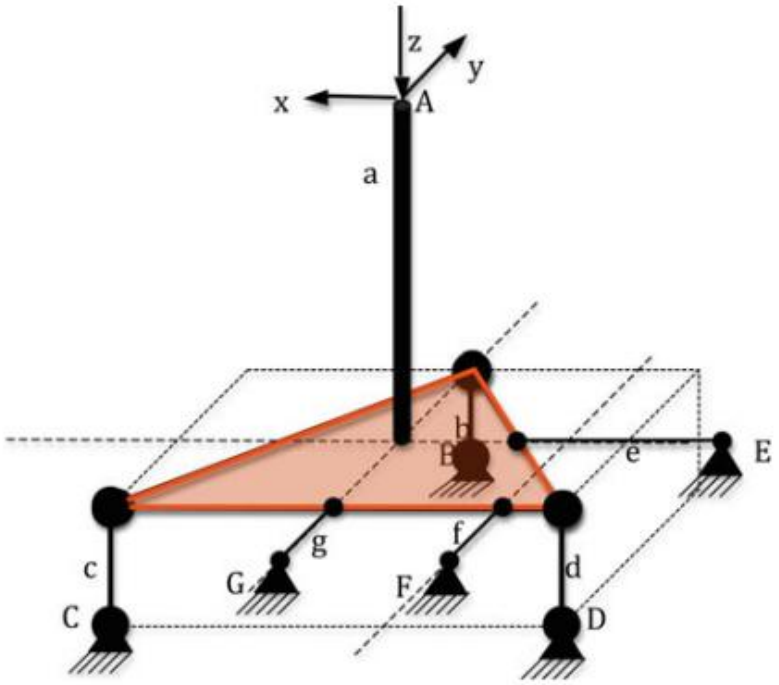

Figure 16: Platform balance system [8].

1.3.4 Internal Balance Systems

Internal Balance Systems are complex measuring tools used in wind tunnel tests. They help get accurate readings of aerodynamic forces and moments by putting sensors inside the model being tested. Most internal balance systems are made up of precise force and moment sensors that are placed on the body or certain parts of a model. These methods are mostly liked because they keep the flow around the model unhindered and help get more accurate results. These hidden parts inside the model make testing in a wind tunnel easier without getting in the way of the airflow. Sensors that use piezoelectric, strain gauge, or capacitive technology are often used in internal balance systems.

These sensors convert the pressures applied to the model into electrical impulses and relay them to a computer environment. The operational foundation of internal balance systems depends on the precise measurement of aerodynamic forces and moments by force and moment sensors in the tested model. A fundamental

attribute of internal balance systems is their precise functionality. Regular calibration is necessary to uphold this precision. Internal balancing systems are adjusted by applying exact force and moment values before testing. Calibration ensures the correct functionality of sensors and the accuracy of measurement values. During calibration, the forces applied to the model's center of gravity and symmetry axes are calculated, and the precision of the sensor outputs is validated. Internal balance systems can precisely identify even the slightest forces. In these systems, forces in the range of micro-Newtons (μN) and moments in the range of micro-Newton-meters (μNm) can be measured. Thus, internal balance devices provide a significant benefit in the aerodynamic evaluation of small models [8].

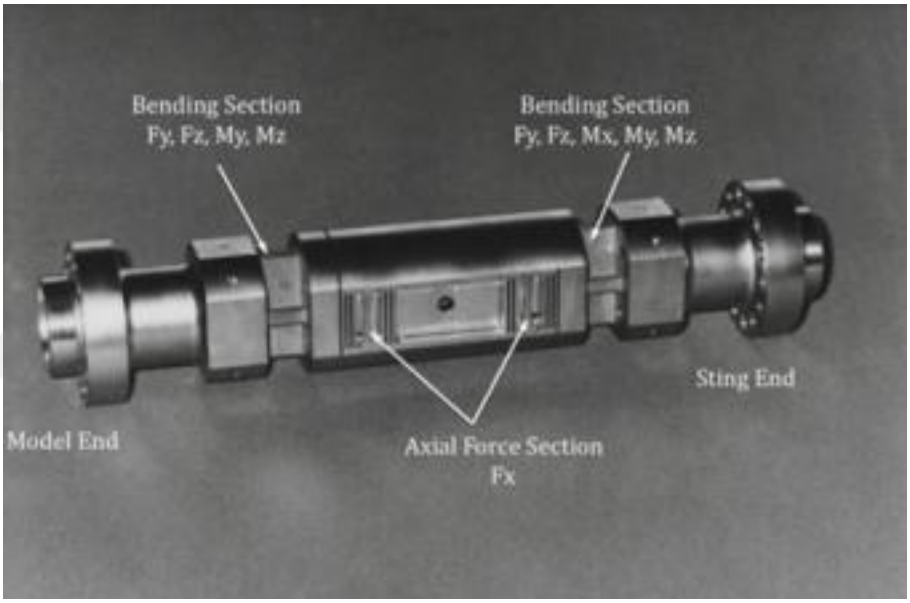

Figure 17: Internal wing balance for a cryogenic wind tunnel with double axial force section [8, 13].

1.4 STEWART PLATFORMS AND THEIR APPLICATIONS

Defining the Stewart platform as a parallel manipulator with six independent legs linking an upper and lower plate gives six degrees of freedom in space. Originally proposed by Gough in 1949 for aircraft tire testing, this idea gained popularity first when D. Stewart demonstrated its use as a flight simulator in 1965 [14]. Since then, it has been referred to as the "Stewart Platform." Its advantages, such as rigidity, precision, and high load-carrying capacity, have made it an essential tool in various applications, including simulation systems, machining tools, telescopic observation instruments, and vibration isolation units [14, 15].

Two kinds of Stewart Platforms are common. Their leg joints set SPS (Spherical-Prismatic-Spherical) and UPS (Universal-Prismatic-Spherical) Stewart platforms apart. Whereas a prismatic (P) joint provides linear mobility, spherical (S) joints link both ends of the legs of an SPS Stewart platform. While this structure offers great mobility, in some orientations the degrees of freedom could be uncontrolled. By contrast, the UPS Stewart platform has a prismatic (P) joint in the middle, a universal (U) joint at one end, and a spherical (S) joint at the other end. Using a universal joint to rotate in two dimensions boosts system controllability and extends its workspace. While more reasonably priced solutions usually call for the SPS architecture, high-precision applications usually demand for the UPS configuration [14, 16, 17].

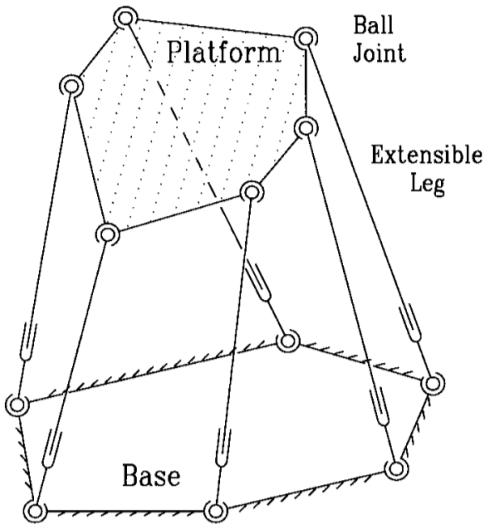

Figure 18: SPS Stewart Platform [14].

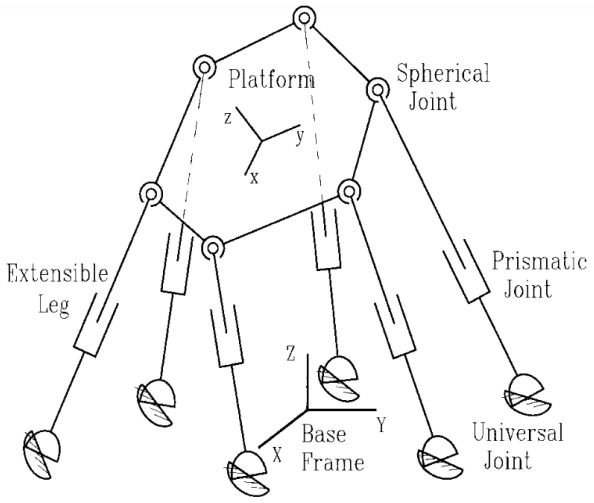

Figure 19: UPS Stewart Platform [14].

Compared to serial manipulators, parallel manipulators offer several advantages, including the distribution of the platform's mass among the legs, a more compact structure, and consequently, higher positioning accuracy [14, 15, 16]. However, their closed-loop kinematic structure makes forward kinematic analysis and defining design constraints quite challenging. Studies in the literature on the Stewart platform focus on improving kinematic and dynamic analysis methods while also exploring optimization techniques and flexible joint applications [14, 17, 18].

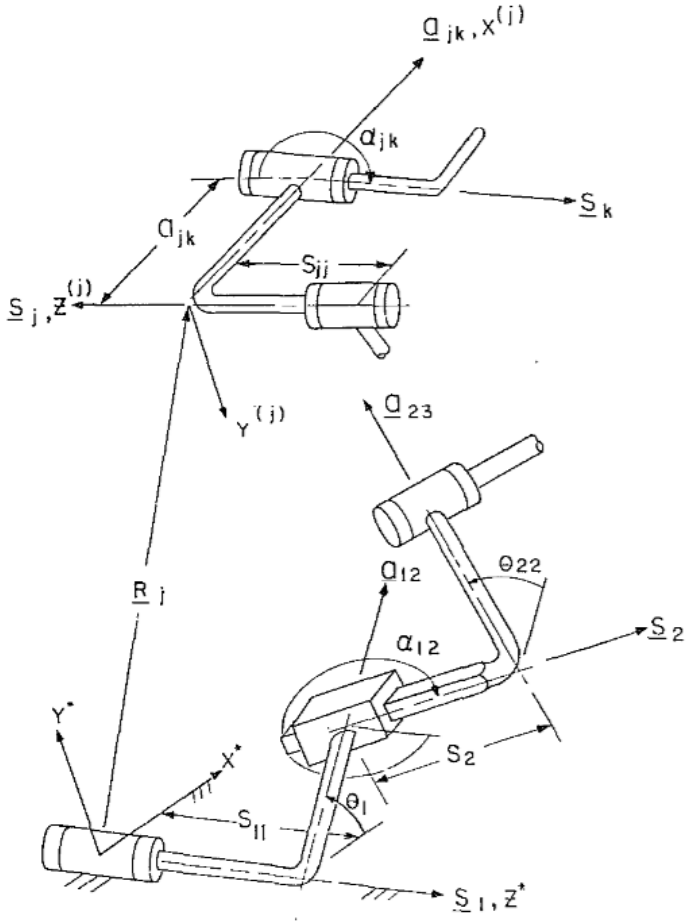

Figure 20: Kinematic representation of a serial manipulator [16].

Stewart platforms offer accurate motion control, making them useful in various industry and scientific areas.

The Stewart Platform is used in various fields that need accurate movement control because it can move in six different ways. This system is very important, especially in the aerospace business. For instance, in the study titled “Development of a Pneumatically Driven Flight Simulator Stewart Platform Using Motion and Force Control” by Pradipta et al. In 2013, the designers and controllers of a reasonably priced

Stewart platform flying simulator driven by pneumatic cylinders show their work. Under the proposed system, "ServoFlight," (Figure 21) seven pneumatic actuators—six for motion control and one central actuator for vertical force compensation—are used. This arrangement seeks to combine precision control appropriate for flight simulation uses with the lightweight and fast-response benefits of pneumatic systems. Two different control systems are proposed to handle the particular kinematic limitations and dynamic response needs. An accurate input-output linearization technique for force control governs the central vertical actuator; an Immersion and Invariance (I&I)-based motion controller controls the outer six actuators. A hyperbolic tangent-based friction adjuster is also used to balance out the nonlinear friction effects that are common in pneumatic systems. The platform can accurately follow reference paths, with angular position errors of less than 0.3°, and respond much more quickly when the friction adjustment method is used, as shown in experiments. These results show that pneumatic actuation can work well instead of hydraulic systems in high-performance motion models as long as the right control methods are used [19].

Figure 21: ServoFlight Platform [19]

In the car business, Stewart Platforms help simulate how vehicle suspension systems work, how cars handle on the road, and perform crash tests. For example, in some test centers, there are dynamic test platforms to determine how vehicles behave under different road conditions [20].

It’s clear that Stewart platforms are a great approach to simulating how cars behave in real life. In 2023, Tytuła and his team used a Stewart platform in a racing simulator to see how well it could mimic the forces drivers feel when making fast turns. We checked how real the system was by comparing virtual driving data to real-world data from the platform. Alkhedher et al. (2020) also studied a 6-degree-of-freedom (6-DOF) Stewart platform that may change shape to keep self-driving cars stable. Their work showed that the system was stable even when things changed, such when the road was bumpy, when it was sloped, or when it had to make rapid turns. Both studies show that Stewart platforms have a lot of potential for driving simulation, vehicle safety, and autonomous control systems [21, 22].

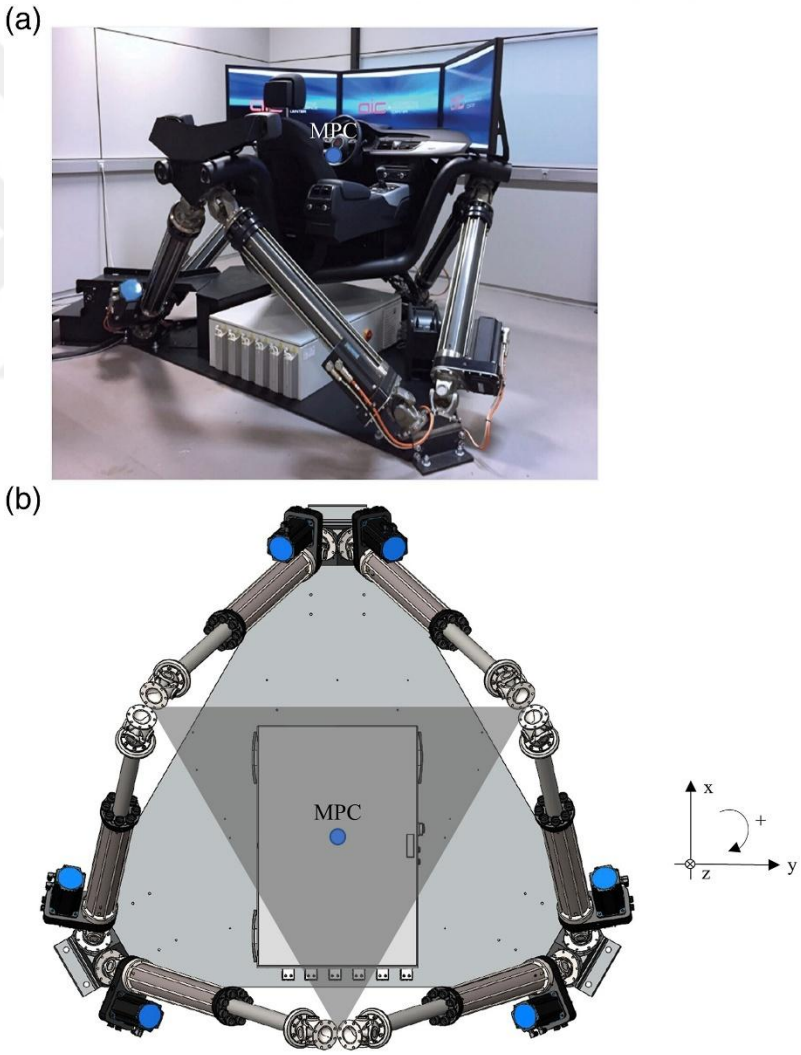

Figure 22: Stewart Platform driving test setup. (a) overview (b) upper view [20].

CNC machining and assembly jobs that are hard to do can be done accurately on Stewart platforms by cutting, drilling, and grinding. Several studies have shown

this to be true. In 2004, Ting, Chen, and Cha used Euler-Lagrange equations and cutting force dynamics to make a Gough-Stewart CNC machine dynamic model. They used an adaptive task-space control law to accurately track and position the trajectory. Harib and Srinivasan (2003) completed a detailed study of Stewart-based machine tools examining both their kinematics and dynamics. Using inverse Jacobian matrices and Newton-Eulerian dynamics that account for actuator inertia and joint friction, they showed that the platform is structurally sound and good for high-precision manufacturing operations. These studies suggest that Stewart platforms are widely used in precision areas such as making aircraft parts and electronic components such as camera modules and semiconductor assembly, because they can cut complex parts with micrometer precision [23, 24].

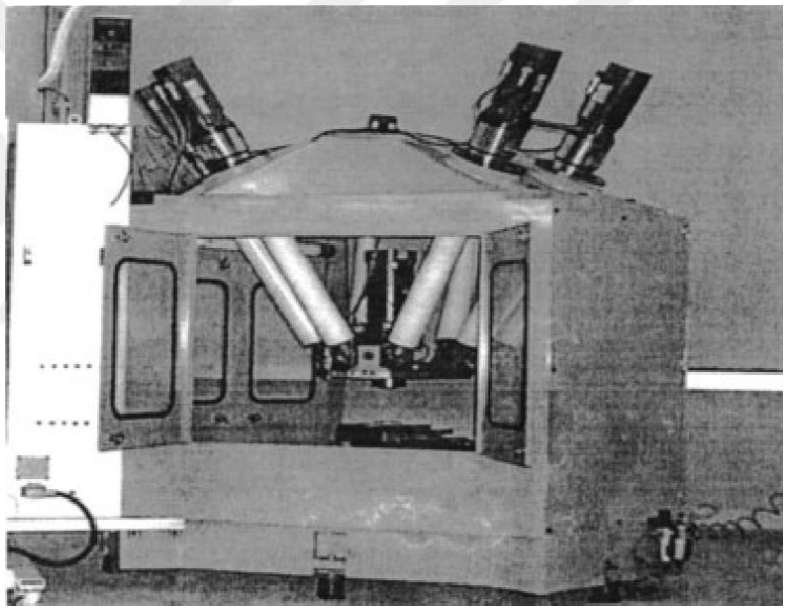

Figure 23: Gough–Stewart-type CNC machine [23]

In medicine, Wapler et al. (2003) – A Stewart Platform for Precision Surgery study introduces a Stewart platform (hexapod) robotic system designed for applications requiring high precision such as microsurgery and image-guided surgeries. The system developed at Fraunhofer IPA consists of a high-precision robotic arm that can move surgical instruments and endoscopes in six degrees of freedom (6-DOF) and a user interface that controls this arm. The selected Physik Instrumente M-850 type hexapod platform can carry loads up to 500N, provides 1µm resolution and 2 arc-sec repeatability. The study specifically targeted neuroendoscopy applications; intracranial applications were performed with three-axis rotation centered on a fixed

anatomical point. Supraorbital, fronto-latero-basal and retromastoidal approaches were tested on a cadaver as a clinical scenario. The system, controlled by voice commands, increases operational safety by limiting instrument movements within a predetermined volume. The voice command system was developed with IBM and Linguatec software and it was observed that the commands were perceived with high accuracy. As a result, it was stated that this robotic system has high potential for surgical areas requiring microscopic precision intervention and can be easily adapted to areas such as eye surgery, spine surgery and orthopedic implant placement in the future [25].

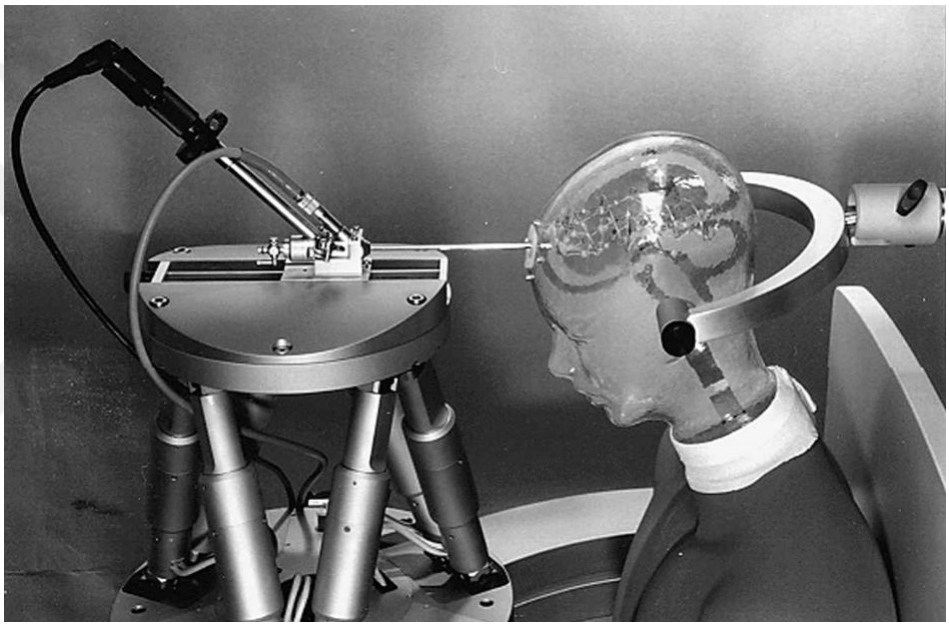

Figure 24: Precision surgery setup with Stewart Platform [25].

Girone et al. (2001) talk about the "Rutgers Ankle," a haptic device for telerehabilitation that was made at Rutgers University and is based on the Stewart platform. The Stewart platform with a pneumatic actuator creates a 6-DOF resistive force that helps in ankle rehabilitation. It also features a built-in computer that runs the platform and a database that can be reached over the internet. The device's goal is to improve things like ROM (range of motion), muscular strength, and coordination by giving the patient's foot feedback force. It employs VR (virtual reality)-based game-like simulations to make people more motivated and make their actions feel more natural. For example, in a virtual environment representing the gas pedal of a race car, the user interacts with the simulation as he moves his ankle. The force and position data measured on the platform are recorded in a database to analyze both the treatment

process and to provide remote evaluation. The first clinical tests were conducted at New Jersey Medical School and were tested on patients of different ages and disorders. The system has been evaluated positively by both patients and physiotherapists; it has been reported that it provides advantages especially in terms of comfort, feedback and traceability. In future studies, a dual platform system and force-controlled applications are planned for balance exercises [26].

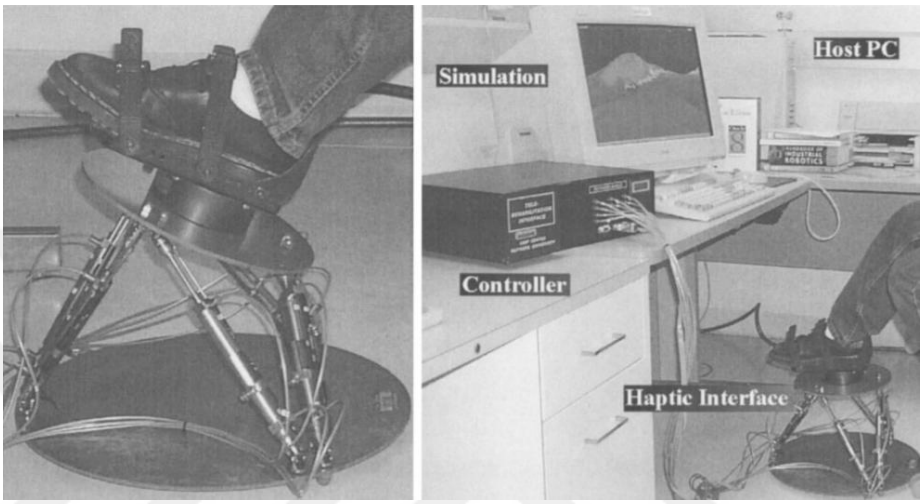

Figure 25: The haptic interface and the setup of the Rutgers Ankle [26].

1.4.1 Kinematic Analysis and Solution Methods

The kinematic analysis of the Stewart platform is primarily divided into two subcategories: inverse kinematics and forward kinematics. In the inverse kinematic problem, the goal is to determine the lengths of the six legs when the platform's position (x, y, z) and orientation (e.g., Euler angles or other rotational parameters) are given. This is rather simple since numerous strategies have been established in the literature and the equations involved are usually solved in a closed-form manner [14, 15].

On the other hand, the forward kinematic problems are more challenging. Here the leg lengths are given; the objective is to determine the direction and position of the platform. The closed-loop structure of the system and the existence of several nonlinear equations—often leading to several real or complicated solutions—make this problem regarded as difficult [14, 15].

The Newton-Raphson methodology is among the most often applied techniques for forward kinematics problem solvers. This approach obtains an accurate solution in a short time by use of its quadratic convergence rate and, given a suitable

starting estimate, generates fast convergence. It does have certain disadvantages, too, including sensitivity to starting values and the possibility of finding oneself caught in local solutions. Changing leg lengths in little increments or choosing an initial estimate around a recognized platform design will help to somewhat offset these problems [14, 15].

The homotopy method—which progressively converts a known solvable system into the intended system via a homotopy function—is another powerful technique. This addresses the convergence sensitivity problem seen in direct solution techniques such as Newton-Raphson, therefore drastically lowering the reliance on the first guess. When there are several answers, the homotopy method is especially helpful since it increases the possibility of locating the most exact or preferred option [27].

1.4.2 Dynamic Analytical Methods

The dynamic study of the Stewart platform seeks to grasp the interactions among the legs, joints, and platform mass of the manipulator. Design and control depend on this, particularly in applications needing high-speed and high-acceleration motions where forces and torques operating on the platform must be correctly described. Usually found in the literature are two main techniques: Lagrange and Newton-Euler [14, 15].

The Newton-Euler method methodically incorporates limitations resulting from the closed-loop structure while individually analyzing each component of the system—legs, upper and lower plates, etc.). Leg mass and inertia help one to ascertain the stresses in the legs and the response forces at the joints. This method produces a huge set of equations that call for the development of position, velocity, acceleration connections, and force-torque equations for every leg. Although this results in a computationally demanding system, contemporary computing capability lets one control and simulate real-time [14, 15, 27].

Conversely, the Lagrange technique develops motion equations from the kinetic and potential energy of the system. It employs Lagrange multipliers or like techniques to integrate loop limitations. Some studies simplify dynamic models by assuming frictionless joints or omitting leg mass, therefore lowering computational complexity. Such simplifications, meanwhile, can cause unacceptable mistakes in applications requiring great accuracy. Consequently, the particular application needs should guide the careful selection of the assumptions in both approaches [27, 28, 29].

1.4.3 Workspace, Control, and Singularity

Singularity remains a major limiting factor even if parallel manipulators have advantages over serial manipulators in terms of stiffness and load capacity. When the system loses or adds degrees of freedom, a single configuration results in either uncontrollably or unbounded motions. The movement of the platform is limited in single configurations, hence too strong tensions in the joints could arise. Therefore, establishing the workspace—that is, the area the platform can travel safely without singularities—is absolutely important [14, 15].

The workspace is a complicated six-dimensional region difficult to perceive since it comprises of both location and orientation elements. Analyzing constant-orientation slices—where one rotational parameter (e.g., rotation around the z-axis) is maintained fixed—helps one to reduce the complexity to three dimensions. For general uses, however, knowing the whole six-dimensional workspace and spotting singularity surfaces still presents constant hurdles. Usually used to investigate the leg lengths and joint limits of the platform, numerical search methods and interval analysis approaches help to find single configurations [31, 32].

Two basic types of control systems for the Stewart platform are dynamic-based control and kinematic-based control. Kinematic control corrects positioning mistakes using feedback from leg lengths. Force/torque measurements are included into the feedback loop in dynamic control therefore facilitating more sophisticated closed-loop control. Applications involving vibrations or high-speed movements depend primarily on dynamic control since acceleration-induced force distributions greatly affect system accuracy [31, 32].

1.4.4 Vibration Isolation, Control, and Future Developments

Where sub-micron accuracy is needed, the Stewart platform shines in vibration isolation applications including semiconductor wafer fabrication, military or aerospace sensor stability, and space telescopes. Its multi-leg support mechanism allows it to operate as an active isolation device when coupled with under controllable actuators. Research on Stewart platforms with piezoelectric, magneto strictive, or voice coil actuators—each with special benefits—in the literature has found [14, 15, 33].

Often employed are passive isolation methods include springs, dampers, or elastomer-based materials. Passive techniques are less effective for high-frequency vibrations or variable external disturbances even if they are straightforward and

affordable. Modern designs so lean towards hybrid (semi-active) solutions, so combining the advantages of active and passive approaches [14, 15, 33, 34].

1.4.5 Flexible Joints and the Future of Stewart Platforms

Flexible joints have lately attracted attention as a replacement for conventional joints—e.g., spherical or universal joints. These improve dynamic performance while also lowering friction and backlash. Nonetheless, in dynamic modeling the elastic deformation of flexible joints adds further complexity. Flexible joints have been found to improve stability in near-singular structures and help to reduce some of the difficulties presented by singularities [35].

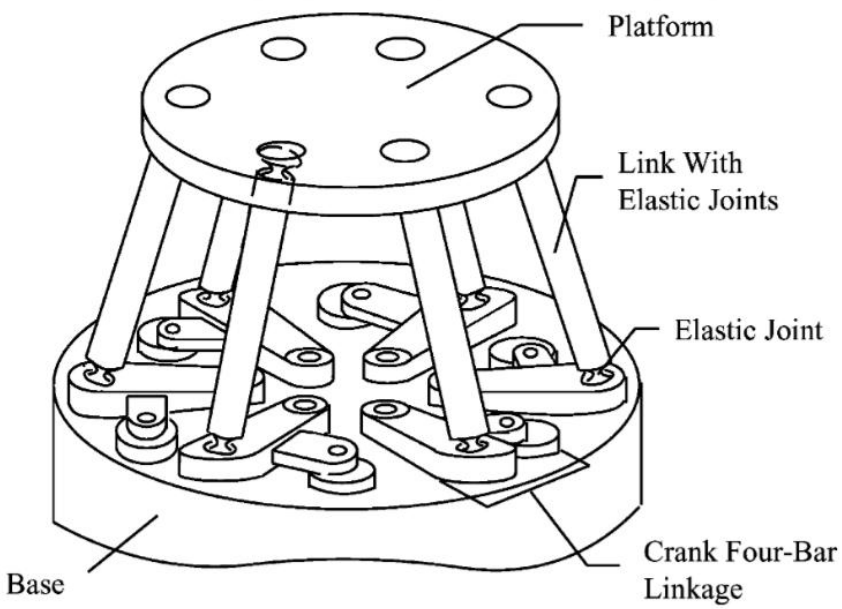

Figure 26: Stewart Platform with elastic joints [35].

1.4.6 Conclusion

Stewart platform research keeps changing in several fields: kinematic solutions, dynamic modeling, singularity analysis, vibration isolation, and flexible joint designs. Future research is supposed to concentrate on multi-objective optimization by include vibration isolation criteria, rigidity, and precision.

Along with rising experimental validation to validate theoretical benefits, flexible-jointed platforms and hybrid passive-active control systems are expected to advance. Modern simulation tools (e.g., ADAMS, SimMechanics, Abaqus) and fast prototyping technologies are projected to speed innovation in Stewart platform design, hence increasing their utility in aircraft, industrial automation, and beyond [14, 15].

CHAPTER II

DESIGN OF 6 COMPONENT WIND TUNNEL FORCE BALANCE

2.1 MECHANICAL DESIGN OF THE SYSTEM

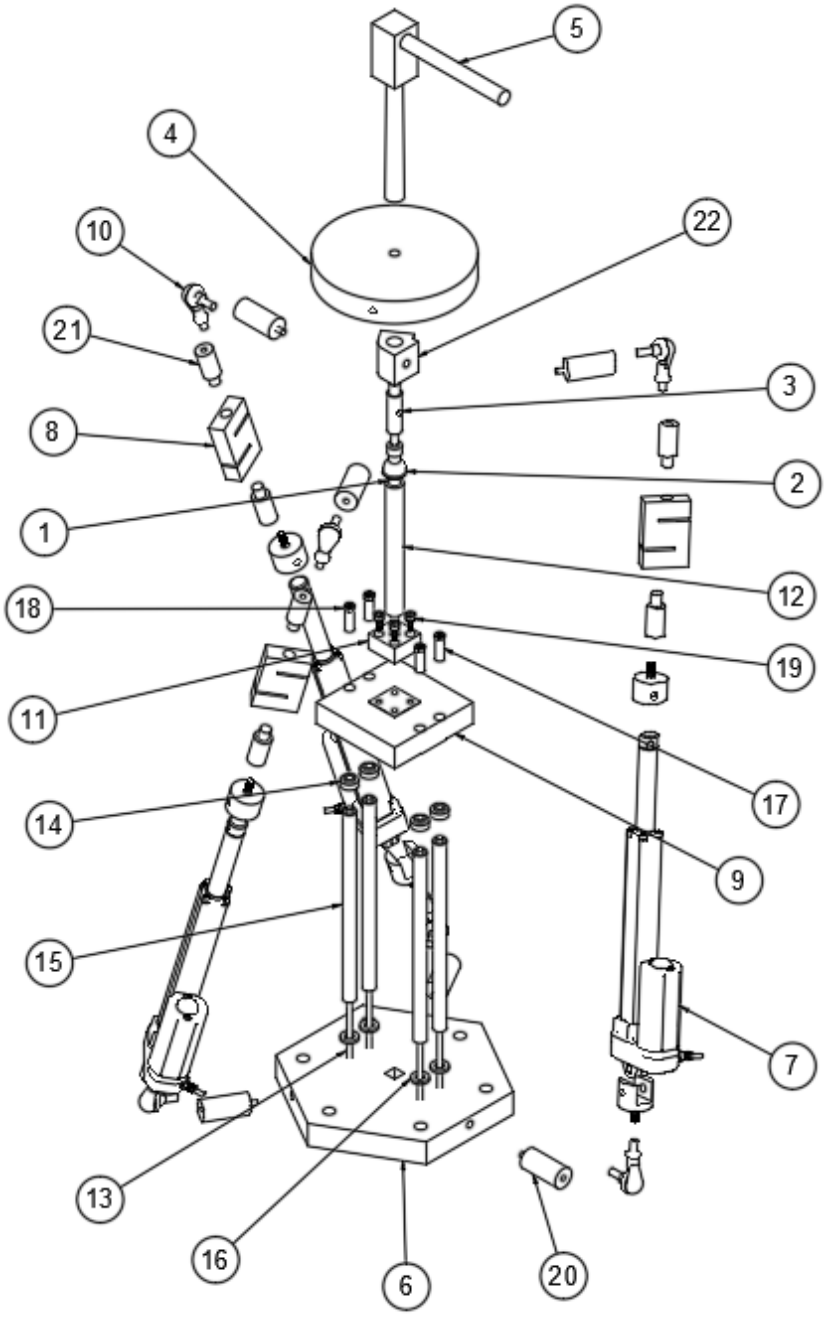

Figure 27: Exploded view of the design

The system basically consists of Base, Top Platform, linear actuators, spherical joints, S-Type single axis loadcells, a loadcell capable of measuring three axis force. Various apparatus / connector parts that will make the connections of these basic parts suitable for sensor connection types and cabling movements are also specified via exploded view and bill of materials. In addition, there is a connection part for IMU that can be used to control the angle of the Top Platform on the system. This part is placed on the Platform Ball Joint Connector located just below the Top Platform and is designed to exhibit the same behavior as the Top Platform in terms of angle. Thus, the Top Platform movement can be followed as a result of the data to be received from the IMU.

Table 1: Bill of materials of the design

Item	Part Number	Qty
22	IMU_Adapter	1
21	Joint_Spacer	6
20	Joint_Spacer	6
19	91290A320_Alloy Steel Socket Head Screw	4
18	90591A151_Zinc-Plated Steel Hex Nut	4
17	Spacer	4
16	Lower Washer	4
15	Tube	4
14	Upper Washer	4
13	Screw Rod	4
12	Upperadapter_Rod	1
11	Upperadapter	1
10	RBL8D	3
9	3axis_weilo	1
8	STKA_S_weilo	3
7	SLA019 150mm	3
6	Base	1
5	Strut	1
4	Top Platform	1
3	Platform Ball Joint Connector	1
2	Inline Ball Case	1
1	Inline Ball Insert	1
Item	Part Number	Qty

Parts List | Parts List

2.1.1 General Considerations

2.1.1.1 Creating Unique Design Different from Stewart Platforms

The mechanism is basically a platform balance system. It consists of a design inspired by the Stewart Platform, the movement of which is restricted for the purpose. In this design, unlike the Stewart Platform, there are three legs between the lower and upper platforms instead of six. In order to reduce the degree of freedom to three, an additional spherical joint is used at the origin point. With the use of this additional spherical joint, the translational movement freedoms of the upper platform of the mechanism are eliminated and it is possible to perform rotation in only three axes, namely pitch roll and yaw movements. The main purpose of this restriction is to reduce

the number of equations of motion of the mechanism and to prevent the strut structure to be located in the wind tunnel from moving in a tendency that will go beyond the boundaries of the experimental chamber of the wind tunnel. In addition, this situation provides an advantage such as using three actuators instead of six. This restriction in the movement capability does not prevent the mechanism from measuring force and moment in six axes. As explained in the mathematical model of the system, in addition to the tension or compression data generated in the two force members to be taken from the three S-Types loadcells, all necessary equations can be solved as a result of measuring the forces by loadcell in the three axes acting on the spherical joint located at the origin of the system.

2.1.1.2 System Overview

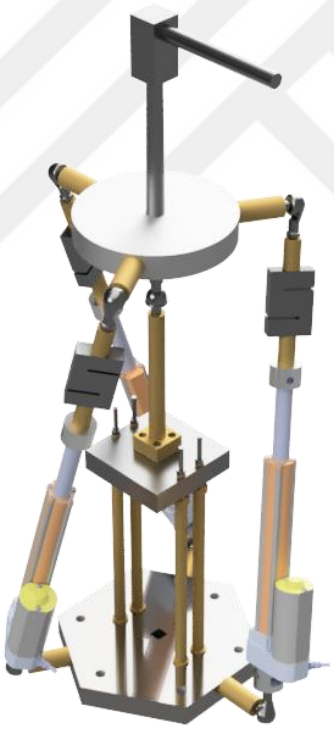

Figure 28: (0,0,0) position

The image of the system after the connections are completed is shown in the (0,0,0) position. The indication in parentheses (pitch, roll, yaw) shows the position of the mechanism at that moment. The (0,0,0) position is specifically planned so that the legs will stop at a certain angle. The reason for this is to eliminate the slack in the

mechanism that will arise from tolerance gaps. When the movement conditions of the mechanism are considered, if the legs are perpendicular to the ground surface and have equal stroke lengths, the legs are in the shortest length they can be. This position is the position where the system creates singularity and does not know exactly which direction it will move in its next position.

2.1.2 Degree of Freedom

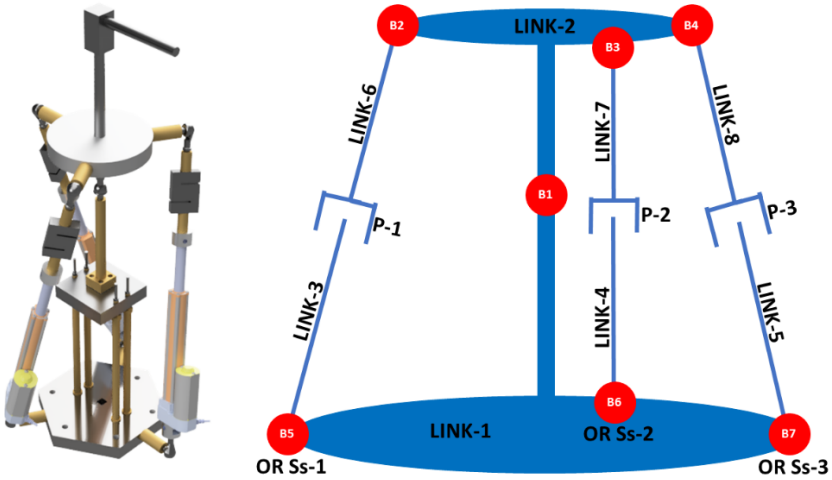

Figure 29: DOF calculation schematic

$$DOF = \lambda(\mathrm{l} - j - 1) + \sum f_i \tag{2.1}$$

$* \ DOF: Degree\ of\ Freedom$

$* \ \lambda: Maximum\ freedom\ in\ selected\ space\ dimentions$

$* \ l: Total\ number\ of\ links$

$* \ j: Total\ number\ of\ joints$

$* \ f_i: The\ degree\ of\ freedom\ provided\ by\ each\ joint\ (e.g., revolute, prismatic, etc.)$

There are a total of 8 links and 10 joints in the system. 7 of the joints are ball joints with three axes of rotational freedom. The remaining three joints are prismatic joints with a single axis of translational freedom. When calculating $\sum f_i$ the joint types are multiplied by their degrees of freedom and all products are added together. For example, in our case, $\sum f_i$ will be calculated as follows.

$$\sum f_i = 3(1) + 7(3) = 24 \tag{2.2}$$

If we take into account the number of links and joints and the fact that there are 6 degrees of freedom in 3D space in the continuation of this calculation, the following calculation will emerge.

$$DOF = 6(8 - 10 - 1) + 24 = 6 \tag{2.3}$$

According to this result, the mechanism has a 6-axis movement capability. When these 6 axes are examined, it is observed that 3 of them have rotational motion in the pitch, roll and yaw axes around the ball joint located at the origin point. The remaining three axes are the degrees of freedom that allow the linear actuators to rotate around their own axes. The rotation of the linear actuators around their own axes does not constitute a situation against the basic movement principle of the mechanism because the basic principle in actuating the mechanism will be provided by performing stroke controls of the linear actuators. Therefore, the actual degrees of freedom in which the mechanism is used functionally will be reduced to 3.

If slotted spherical joints were used instead of ball (spherical) joints on the base plate, the three axes that cause the linear actuators to rotate around their own axes would have been eliminated. In its calculation, the $\sum f_i$ value would be calculated as follows.

$$\sum f_i = 3(1) + 4(3) + 3(2) = 21 \tag{2.4}$$

In this case, since the degree of freedom of the slotted spherical joint is 2, the value of $\sum f_i$ decreases from 24 to 21. Therefore, the degree of freedom calculation is reshaped as follows.

$$DOF = 6(8 - 10 - 1) + 21 = 3 \tag{2.5}$$

2.1.3 Movement Capability

The rotation capability of the system in three axes is kept within ±20° angular limits. Since the spherical joints used in the system have ±22.5° pitch, roll and yaw axes movement capabilities, the safe limit of ±20° has been determined accordingly.

It is anticipated that this much angular change will be sufficient in wind tunnel studies in each axis. The movement limit points of the mechanism are shown with the help of images as (0,0, -20), (20,0,0) and (0,20,0) positions. The same capability is also available in the opposite directions.

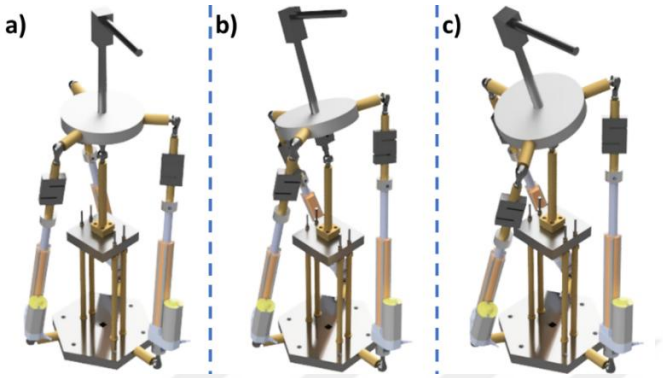

Figure 30: a) (0,0, -20) position, b) (20,0,0) position, c) (0,20,0) position

2.1.4 Mathematical Model

In order to create the motion and force equations of the mechanism, vectorial calculations were performed using free body diagrams. First of all, the effect of the forces acting on an aerodynamic model placed at point P of the system on the load cells was calculated. In order to achieve this, the free body diagram of the Top Platform was drawn.

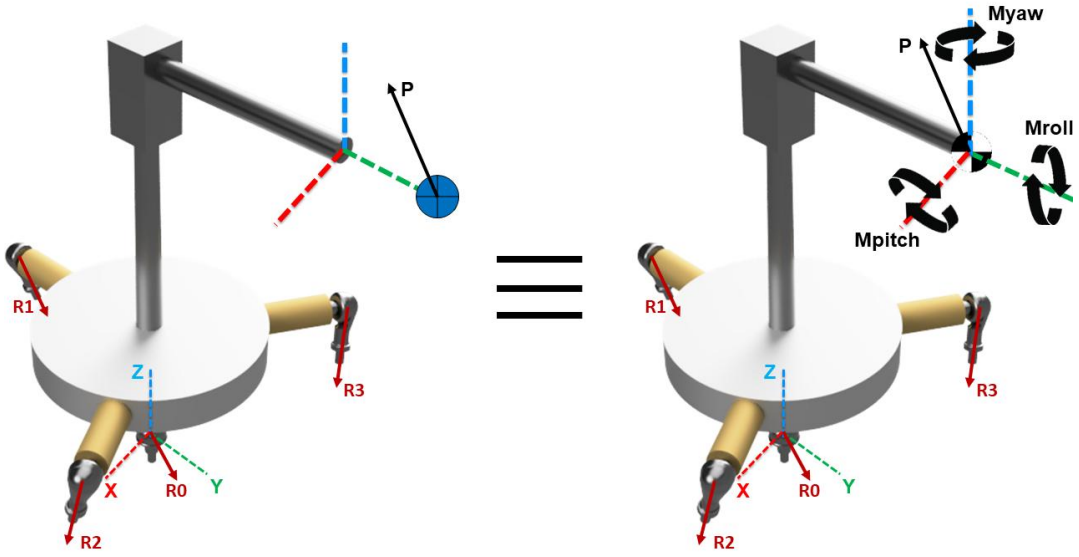

Figure 31: Free Body Diagram of the Top Platform

The force acting on point P (center of pressure point) in the free body diagram of the Top Platform is the force P. If we carry the force from P point to the center of gravity of the test model which is the tip of the sting, external moments around center of gravity will be included. The resultants of this force consist of side force, drag force and lift force. These forces are the forces in the x, y and z directions, respectively, according to the coordinate system determined. With the effect of the P force and moments (External forces), reaction forces occur in the system. These reaction forces occur on the ball joints as R0, R1, R2 and R3. Due to the structure of the ball joints, when friction is neglected on these joints, no reaction moment occurs and they only affect the equations as forces. Since the reaction forces R1, R2 and R3 are located on the legs between two ball joints, they only occur in the same direction as the leg in compression or tension due to the two-force member principle. For this reason, if the directions of the legs in space, i.e., unit vectors, are calculated, since the data from the load cell will also express the magnitudes of these forces, these forces can be calculated both in magnitude and direction. The R0 force is not located on a two-force member. For this reason, the components in all three axes must be measured separately. A three-axis load cell can provide this. The ratio of the components in the three axes to each other will also determine the orientation (direction) of the R0 vector a known value.

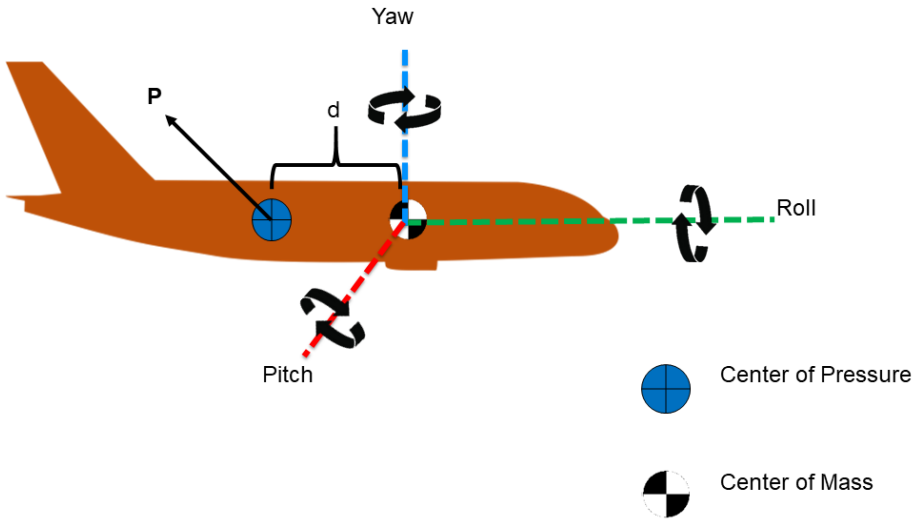

Figure 32: Moment generation on an aerodynamic model

The fact that point P is off the center of gravity of the model to be tested causes the force $\vec{P}$ to create a moment around the center of gravity.

$$\sum \overrightarrow{F_x} = \sum \vec{F}_y = \sum \overrightarrow{F_z} = 0 \tag{2.6}$$

$$\sum \vec{F} = \overrightarrow{R_0} + \overrightarrow{R_1} + \overrightarrow{R_2} + \overrightarrow{R_3} + \vec{P} = 0 \tag{2.7}$$

$$\vec{P} = -\left(\overrightarrow{R_0} + \overrightarrow{R_1} + \overrightarrow{R_2} + \overrightarrow{R_3}\right) \tag{2.8}$$

$$\vec{P} = (F_{side})\vec{\imath} + \left(F_{drag}\right)\vec{\jmath} + \left(F_{lift}\right)\vec{k} \tag{2.9}$$

When the calculations are made assuming that the system is in static conditions, the force on the three axes can be easily calculated thanks to the above balance equation. The sum of the x resultants of the reaction forces will be equal to the negative side force, the sum of the y resultants will be equal to the negative drag force, and the sum of the z resultants will be equal to the negative lift force.

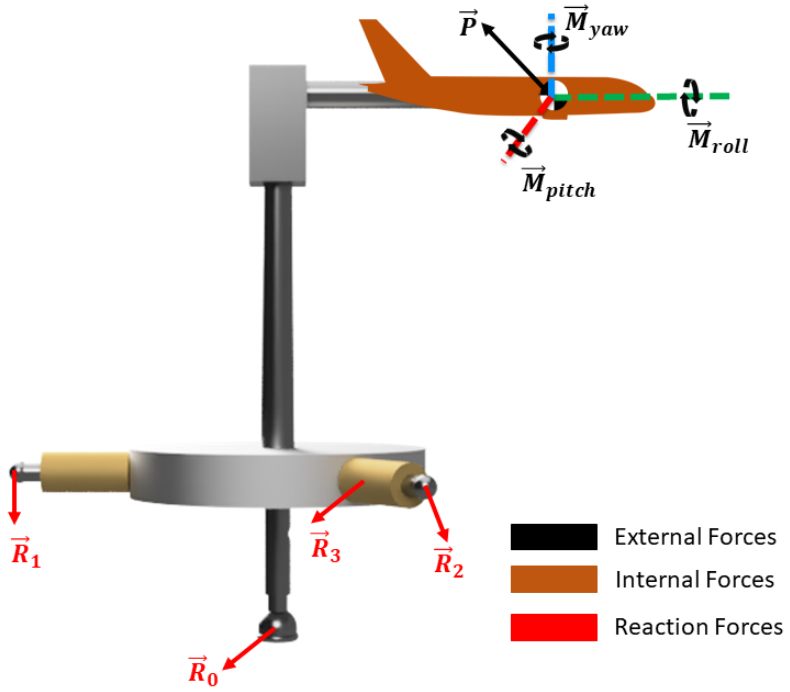

Figure 33: FBD when model mounted to the sting, static condition

Another point to consider in the calculations is the direction of the calculated force/moments and the type of force they represent. The P and M moments in the equations are calculated as the external forces shown in Figure 33. If we draw the two FBDs separately, as in Figure 34, the sum of the internal forces/moments in the sting structure and the internal forces/moments on the model must be zero. Therefore, these internal forces are the negative of each other.

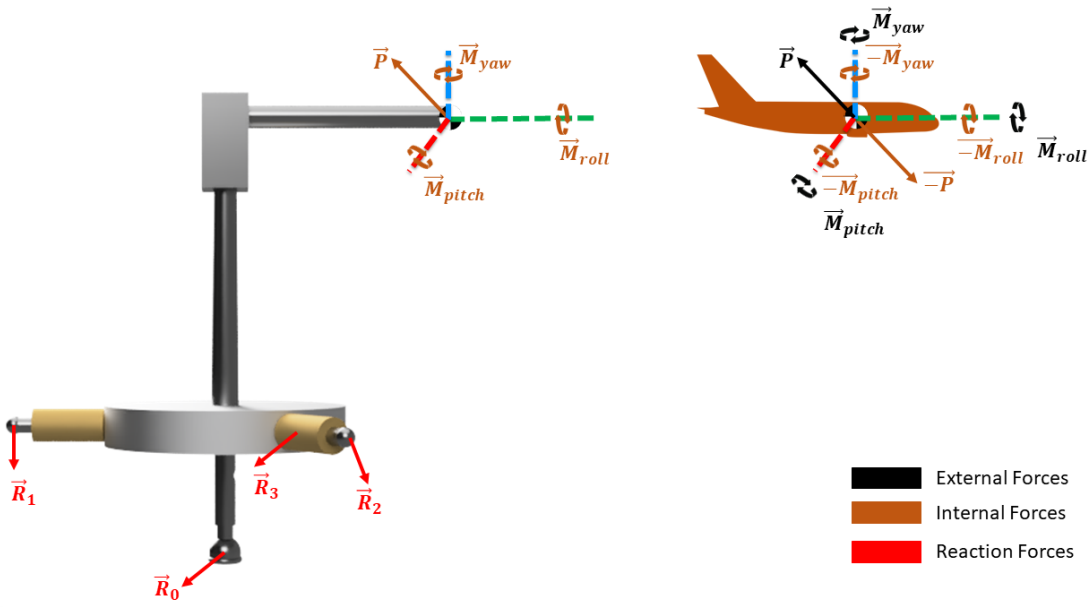

Figure 34: FBD when model separated from the sting, static condition

$$\sum \overrightarrow{M_o} = \overrightarrow{r_1} \times \overrightarrow{R_1} + \overrightarrow{r_2} \times \overrightarrow{R_2} + \overrightarrow{r_3} \times \overrightarrow{R_3} + \overrightarrow{r_p} \times \vec{P} = 0 \tag{2.10}$$

$$\sum \overrightarrow{M_o} = \overrightarrow{r_1} \times \overrightarrow{R_1} + \overrightarrow{r_2} \times \overrightarrow{R_2} + \overrightarrow{r_3} \times \overrightarrow{R_3} + \overrightarrow{r_{cg}} \times \vec{P} + \overrightarrow{M^G} = 0 \tag{2.11}$$

$$\overrightarrow{M^G} = M^G_{pitch}\,\vec{\imath} + M^G_{roll}\,\vec{\jmath} + M^G_{yaw}\,\vec{k} \tag{2.12}$$

$$\overrightarrow{M^G} = \begin{pmatrix} M^G_{pitch} \\ M^G_{roll} \\ M^G_{yaw} \end{pmatrix} \tag{2.13}$$

$$\vec{M} = T^T \overrightarrow{M^G} \tag{2.14}$$

The moment calculation is performed around the origin point. The distances of the action points of all forces to the origin create a moment on the origin. If the cross-product operation of the force at the action point with the vectors having the tip at the action points is performed, the moment created by that force on the origin point will be calculated.

When the calculation is performed in such a way that the sum of all moments is zero under static conditions, since all forces and position vectors except moments have been calculated or measured, the resultant $\overrightarrow{M^G}$ vector, which has separate components on three axes, can be easily calculated on the ground coordinate system. However, the following should be taken into consideration; the mechanism is designed to move the orientation of the Top Platform and to take measurements in different

positions. Therefore, Euler-Angles will change in each position. It is logical to calculate the measured side, drag and lift forces according to the ground coordinate system. Because, if an aircraft makes a pitch angle, the lift force that will occur is again calculated according to the ground coordinate system. However, the moments that will occur on the model must be converted to the model's own coordinate system. For example, a moment that will occur in the roll direction on a model should result in the roll axis of that model's own coordinate system, so that it can be determined how this moment will provide a dynamic movement to that model. To achieve this, the moment vector in the ground coordinate system will be processed with the transformation matrix and the moment vector on the model coordinate system will be obtained.

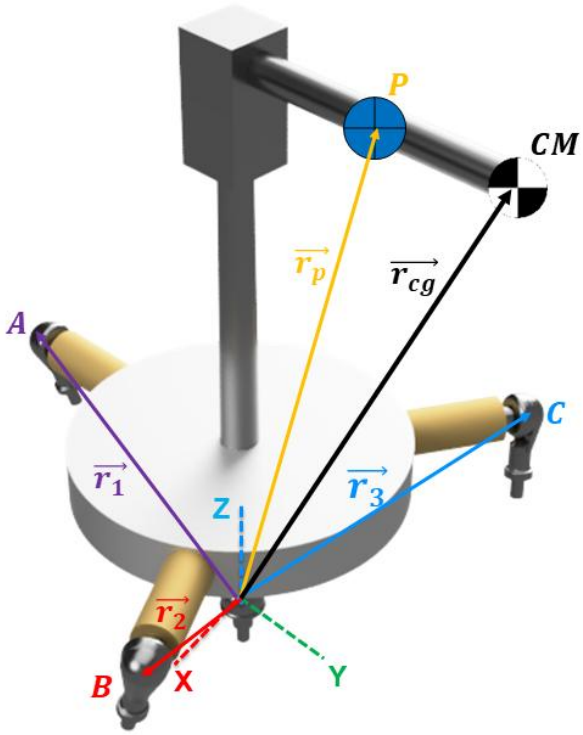

Figure 35: Position vectors

The position vectors that appear in the moment equations are the vectors located on the Top Platform. The orientations of these vectors do not change on the moving Top Platform. However, when the Top Platform moves, the vector orientations with respect to the ground coordinate system also change. This change can be calculated with the help of transformation matrices when the angles of the Top Platform on the pitch, roll and yaw axes are known, and the projections of the vectors whose orientations change on the ground coordinate system can be calculated.

$$\overrightarrow{r'} = T_3 T_2 T_1 \vec{r} = T\vec{r} \tag{2.15}$$

$$\vec{r} = T^T \overrightarrow{r'} \tag{2.16}$$

$* \overrightarrow{r'}: constant$

$* \vec{r}: updated$

$$T = \begin{bmatrix} cos\theta_{x'x} & cos\theta_{x'y} & cos\theta_{x'z} \\ cos\theta_{y'x} & cos\theta_{y'y} & cos\theta_{y'z} \\ cos\theta_{z'x} & cos\theta_{z'y} & cos\theta_{z'z} \end{bmatrix} \tag{2.17}$$

$$T = T_3 T_2 T_1 \tag{2.18}$$

The basic equation of the transformation matrix is given above. Three different transformation matrices are calculated on three different axes and the general transformation matrix is created by multiplying these matrices with each other in the correct order. This matrix allows a vector in space to switch between two different coordinate systems.

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta_{pitch} & sin\theta_{pitch} \\ 0 & -sin\theta_{pitch} & cos\theta_{pitch} \end{bmatrix} \tag{2.19}$$

$$T_2 = \begin{bmatrix} cos\theta_{roll} & 0 & -sin\theta_{roll} \\ 0 & 1 & 0 \\ sin\theta_{roll} & 0 & cos\theta_{roll} \end{bmatrix} \tag{2.20}$$

$$T_3 = \begin{bmatrix} cos\theta_{yaw} & sin\theta_{yaw} & 0 \\ -sin\theta_{yaw} & cos\theta_{yaw} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.21}$$

Pitch (Around “x-axis”) → Roll (Around “y-axis”) → Yaw (Around “z-axis”)

(2-1-3 Sequence)

When the transformation matrices are calculated in a way that follows this order in the Pitch, Roll and Yaw directions, the above T_1, T_2 and T_3 matrices will be found. The multiplication order of the matrices is critical. They can be multiplied in any desired order, but the order to be determined in the calculation of the parameters that will enter the same equations must be identical. In the framework of this study, the order referred to as the 2-1-3 sequence in the literature was used.

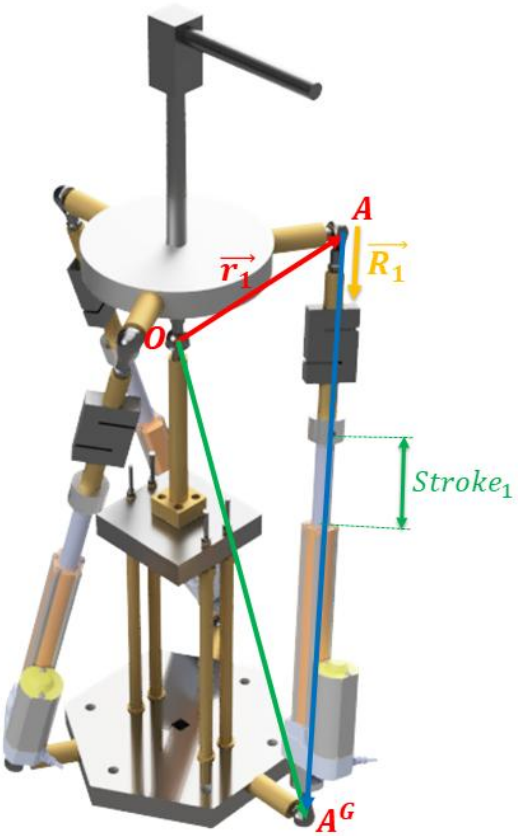

Figure 36: Stroke and unit vector calculation

$$\overrightarrow{AA^G} = \overrightarrow{OA^G} - \overrightarrow{r_1} \tag{2.22}$$

$$Stroke_1 = \left\|\overrightarrow{AA^G}\right\| - constant \tag{2.23}$$

$$Unit\ vector\ of\ \overrightarrow{AA^G} = \frac{\overrightarrow{AA^G}}{\left\|\overrightarrow{AA^G}\right\|} \tag{2.24}$$

$$\overrightarrow{R_1} = R_1 \cdot \frac{\overrightarrow{AA^G}}{\left\|\overrightarrow{AA^G}\right\|} \tag{2.25}$$

After finding the $\overrightarrow{AA^G}$ vector, the $Stroke_1$ vector can be calculated. There is only a constant coefficient difference between the $Stroke_1$ vector and the norm of the $\overrightarrow{AA^G}$ vector. This coefficient originates from the geometric parameters in the length direction of the actuator. The value of the distance between points A and A_g, except for the stroke, originates from the fixed geometric values of the mechanical parts located between these two points. Since the dimensions of all parts are known, this fixed value can be measured. Since the directions of the reaction forces will also be

found by finding the unit vectors, the vectors of these forces will also be calculated exactly. These unit vectors are calculated by dividing the vector between the two points by its own norm. As a result of the above equations, the $\overrightarrow{R_1}$ vector will be calculated. The representation in the calculations is carried out over the points A and A_g. If the same calculations are made on $B, B_g, C \; and \; C_g$, the values of $Stroke_2, Stroke_3, \overrightarrow{R_2}$ and $\overrightarrow{R_3}$ can also be calculated.

2.1.5 Singularity Position

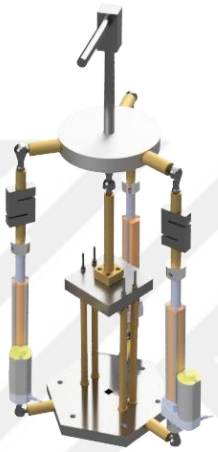

Figure 37: Singularity position

In the singularity position, the mechanism behavior is unstable in the yaw axis. This causes the mechanism to move out of control in the yaw axis with a small external factor. Preventing this situation is critical to prevent movement fluctuations that may occur due to air flow under the flow in the wind tunnel. The main reason for this behavior is that when the mechanism makes a small rotation from the singularity position to another yaw direction, there is a very small difference between the initial and final stroke lengths of the linear actuators. This difference is so small that it is smaller than the gap amounts found in the structure of the joints due to their nature. For this reason, the mechanism can move freely in the yaw axis to a certain extent by using these gaps in the joints when it is in the singularity position. In light of these reasons, the mechanism should be used at points far from the singularity position. In order to ensure this, the mechanism was moved +60° in the yaw axis relative to the singularity position and the (0,0,0) position was determined. This position was determined by considering the maximum stroke length of the linear actuators, which

is 150 mm, and was selected in a way that would limit the movement capability of the mechanism at the least level.

Table 2: Stroke values according to yaw value (pitch = 0 and roll = 0)

Yaw (º)	Stroke 1, 2 and 3 (mm)	Difference (mm)
0	84,9818	0,6905
-1	84,2913	-
-10	78,3946	0,6158
-11	77,7788	-
-20	72,6518	0,5199
-21	72,1319	-
-30	67,9539	0,4055
-31	67,5484	-
-40	64,4693	0,2764
-41	64,1929	-
-51	62,1882	0,1228
-52	62,0654	0,1085
-53	61,9569	0,0945
-54	61,8624	0,0794
-55	61,783	0,0652
-56	61,7178	0,0508
-57	61,667	0,0363
-58	61,6307	0,0217
-59	61,609	0,0073
-60	61,6017	-0,0073
-61	61,609	-0,0217
-62	61,6307	-0,0363
-63	61,667	-0,0508
-64	61,7178	-0,0652
-65	61,783	-0,0797
-66	61,8627	-0,0942
-67	61,9569	-0,1085
-68	62,0654	-0,1228
-69	62,1882	-

Stroke calculations were performed for different angular positions using the stroke calculation formulas in the mathematical model section of Table 2, when pitch and roll values are 0°. Because pitch and roll values are 0°, the three stroke values are equal at different yaw angles. Calculations were performed assuming the zero position was 60° of yaw relative to the vertical position of the actuators. Therefore, singularity is observed around -60°. Note that the stroke difference between the positions 1°

before and after converges to 0mm around -60°. This indicates that the singularity is dependent on the yaw movement in this position.

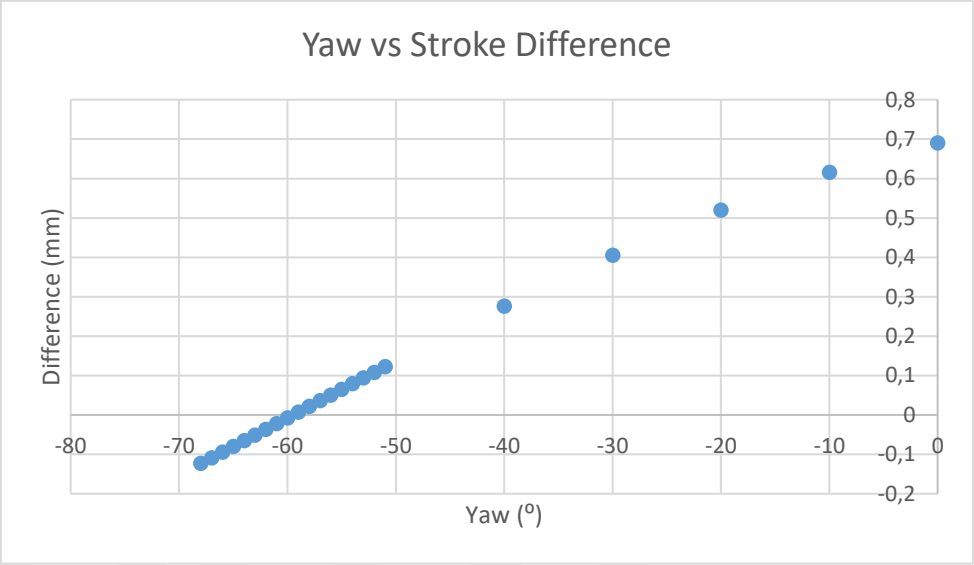

Figure 38: Stroke differences occurring at 1 degree yaw difference

The table in Figure 38 plots the stroke difference between 1º before and after. This difference becomes 0 at -60º, according to the graph. This graph also illustrates the stroke sensitivity required to achieve 1º accurate yaw control. For example, to achieve an accurate yaw value at -20º, stroke sensitivity must be less than 0.5 mm. Figure 39 shows the stroke distribution around the -60º yaw position. The fact that the bottom point is at -60º, the position where the actuators are perpendicular to the ground, indicates that this position is singular.

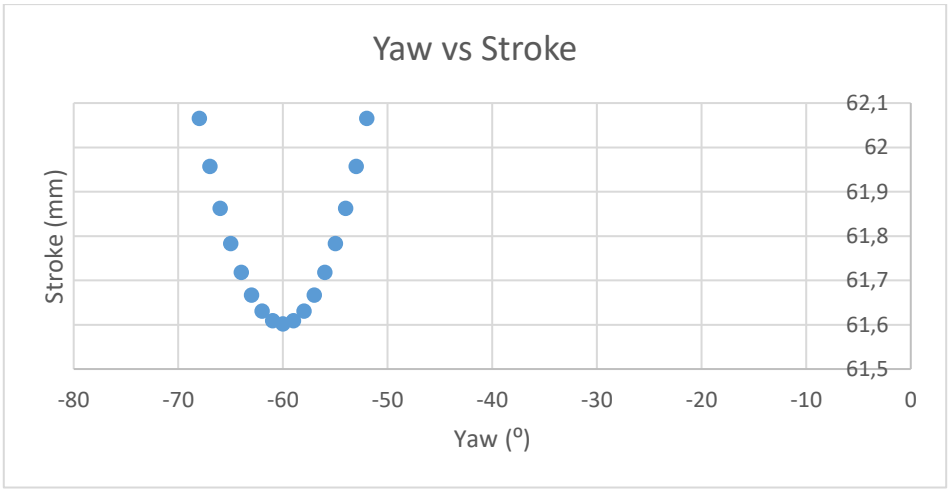

Figure 39: Yaw vs Stroke

2.2 ELECTRICAL DESIGN OF THE SYSTEM

2.2.1 Sensors

Sensor selection should be made carefully when considering the measurement sensitivity. When the range values planned for the sensors to work and the environmental conditions, they will be exposed to are evaluated together with the desired measurement sensitivity, sensor selection can be made within a reasonable framework.

2.2.1.1 Loadcell Selection and Usage

Two types of load cells are used in this system being worked on. One of the load cells is an S-Type single-axis load cell, while the other is a load cell with the capacity to read three axes. Both load cells can take positive and negative measurements according to the degree they are calibrated and tared in the axes they can measure. These measurements are provided by strain gauge measurements in the load cells. The basic logic is that the mechanical structure of the load cell under load is suitable for yielding within the permitted limits. As a result of this yielding, the H-Bridge circuit structure in the strain gauges starts to allow current flow in the mA range with the help of the resistance difference formed. By calibrating this current signal with the help of known weights as a result of loadings to be made to the load cell, the value read can be converted into an output value as mass or weight.

The two types of load cells used in the system were selected by paying attention to their geometric dimensions. As the load capacity increases, the load cell design is likely to grow in size. The reason for this is actually hidden in the basic operating logic. In general, a larger part will flex less under mechanical loading and will be used to measure higher capacity loads. Permanent deformation of these sensors will directly cause the sensor to fail, so attention should be paid to the working range based on the load.

A high load capacity does not mean that a load cell is better. The selection should be made according to the level of the measurement to be made as much as possible. The value at which the load cells are most optimally calibrated is half of their load capacity. In other words, the middle point of the span created by the load capacity is the most efficient value at which the load cell will operate because the most accurate measurements are taken around the calibrated value.

Although a load cell with optimum capacity should be preferred, the environmental conditions that the load cell will be exposed to throughout its life will also play an important role in the selection of these load cells. For example, there are heavy and rigid parts in the system being worked on due to the design. In addition, the powerful actuators can create unexpected loads on the parts on the mechanism during a malfunction. Considering the situations such as being placed under these heavy parts and the actuators being exposed to extra loads in cases where they operate outside of the desired range, these load cells have been selected with capacities that will prevent them from being damaged more than the range to be measured. Both S-Type single-axis and triaxial load cells have been selected with a capacity of 100 kgf, i.e., 1 kN, on each axis.

The estimated value range that the load cells will take measurements is 0-5kgf. Since this range is low compared to the capacity, it may create a margin of error, but if the calibration is performed in this range, the error will be greatly reduced. For this reason, the calibration processes were performed in the range of 2-2.5kg.

Another issue that should be considered in multi-axis load cells is the cross-talk sensitivity of these load cells [12]. This term describes the situation where the axis of a multi-axis load cell that is measured and the other axes of the load cell are affected even though there is no actual loading. For example, as a result of loading on the "x" axis in a three-axis load cell, value differences can be observed in the "y" and "z" axes. This creates a margin of error in axes other than the loaded axis. Although this value is not directly given in the given specifications of the selected load cell, the Total Error value is within ±0.2 %FS. This amount of error was found to be appropriate for the desired measurement sensitivity. In addition, as a result of the tests conducted, this interaction between the axes was not observed at levels that would increase the margin of error significantly.

When the Total Error values of the two types of load cells are examined, it is seen that this value of the S-Type load cell is ±0.022%FS. However, the Total Error value of the triaxial load cell is ±0.2%FS, which is relatively much larger than the S-Type, and there is a difference of approximately ten times. The reason for this is the extra error brought by the cross-talk sensitivity of the multiaxial load cell. Considering this error difference, in order not to increase the amount of error in the data collected from the triaxial load cell, more comprehensive digital indicators containing signal

amplifiers that reduce the margin of error were used. In this way, errors caused by electrical noise, etc. were reduced.

Another factor that increases the sensitivity in load cell selection is the excitation voltage and output signal voltage values of the load cells. Choosing these values relatively high ensures that the data received from the load cell fluctuates less and provides more accurate results. This situation is similar to the need to create a greater energy difference to change the temperature value of a material with a high heat capacity. Electrical noise that will occur due to environmental reasons will affect sensors operating at low potential energy more, because when the magnetic field strength created by electronic devices and current-carrying cables that create noise in the environment is compared to the magnetic field strength of the sensor itself, the noise has a chance to affect the sensor value according to the level of the value that will be obtained. Increasing the difference between these two types of magnetic field strength and ensuring that the sensor is more dominant in this regard will also reduce the errors caused by noise.

Whether the load cells will work properly can be tested with a short test before taking data from the load cells. Input and output resistance values are given in the specifications according to the production values of the load cells. By checking these values, it is possible to test at any time whether the load cell is manufactured correctly or whether it has been damaged during use. The measurement of these values can be done by measuring the input resistance value between the E+ and E- inputs and the output resistance value between the S+ and S- inputs with a multimeter. According to the datasheet, the resistance between the input pins should be 400 Ω ± 20 Ω, while the resistance between the output pins should be 350 Ω ± 3 Ω.

As a result, the correct selection of load cells followed by their use with the correct additional devices in the correct range and periodic necessary checks will ensure that the most accurate results are obtained. This study was attempted to be carried out within this scope.

2.2.1.2 Inertial Measurement Unit

The need to use an Inertial Measurement Unit (IMU) in this study arose for position tracking. However, in later stages of the study, the ability to precalculate the stroke values required for each position using the forward kinematics method, and then store the calculated values on an SD card to retrieve the required data, thus reducing

the need for input from the IMU for position control. Nevertheless, the use of this module is crucial for comparing position results within the system. This allows for the comparison of position data obtained from two different methods.

The MPU6050 is a 6-degree-of-freedom (6-DoF) inertial measurement unit (IMU) manufactured by InvenSense. It is used in many electronic devices. This sensor has a chip that includes a 3-axis accelerometer and a 3-axis gyroscope. This allows it to obtain information about both linear acceleration and angular velocity.

The sensor supports I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface) protocols for digital data output, although I2C is generally used in module versions. When communicating via I2C, the MPU6050's default address is 0x68, which can be changed to 0x69 by setting the AD0 pin to HIGH. The communication speed on the data bus can typically be 100kHz or 400kHz. The sensor can be powered by 3.3V or 5V, but if the chip is used directly, only 3.3V is suitable. In module versions, a 5V power supply is possible thanks to the integrated voltage regulator.

The MPU6050's built-in DMP (Digital Motion Processor) is one of its most important features. By combining data from the gyroscope and accelerometer, the DMP can figure out the orientation. This makes the microcontroller's job a lot easier. The DMP may give you orientation information in either Euler angles (yaw, pitch, roll) or quaternion format. This capability makes it possible to make very precise orientation computations in things like balance robots, gimbal systems, and virtual reality apps.

You can change the measurement ranges when you use the MPU6050 module. You can choose between ±250°/s, ±500°/s, ±1000°/s, and ±2000°/s for the gyroscope. The accelerometer can work in ranges of ±2g, ±4g, ±8g, and ±16g. You can change the resolution and sensitivity of this option to fit the application. The MPU6050 has a temperature sensor and can send this data over I2C, although the temperature readings aren't very accurate.

Versions of the module, often referred to as GY-521, come with a user-friendly pin interface. These modules typically include VCC, GND, SDA, SCL, and INT pins. The INT pin allows the MPU6050 to generate an interrupt signal under certain conditions and is particularly useful as a data-ready signal when used with a DMP. Integration with microcontrollers like Arduino is quite easy and can be quickly implemented thanks to widely available open-source libraries.

In conclusion, the MPU6050 IMU module, thanks to its low cost, small size, integrated DMP support, and precise sensors, is a powerful and economical sensor suitable for a wide range of applications, from amateur projects to advanced robotic systems. Its use is suitable for the specific application.

2.2.2 Actuators

The actuator used in the system is an actuator that provides linear motion. It provides this motion with the help of a brushed DC motor that can be driven with a 24VDC potential difference inside. This actuator, which can operate under 1000N force, can move at a speed of 12mm per second under 24V potential without any force effect. The stroke value, which is one of the most critical values for the design, is 150mm. In other words, there is a difference of approximately 150mm between the longest and shortest points of the actuator shaft. The longest and shortest values of the actuator are provided with the help of stopper switches placed inside the actuator. These switches, which are placed in a way that prevents the mechanical strain of the actuator, produce a signal as a result of the actuator movement and contact and are triggered. This signal affects the electronic circuit in a way that stops the operation of the actuator motor. Switch positions and the situations in which these switches are triggered are arranged according to the tolerance ranges of 291 ± 2mm and 150 ± 2mm given in the actuator technical drawing. Therefore, this tolerance range should be taken into consideration at the extreme points of the actuator in the design.

The actuator position control is provided by the potentiometer inside the actuator. There is a potentiometer inside the actuator that can show a resistance value between 0-10 kΩ. As the actuator stroke value approaches 0 mm, this value approaches 0 kΩ, while as the stroke value approaches 150 mm, this value approaches 10 kΩ. As a result of the tests performed, when the actuators triggered the home switch (the switch triggered by the actuator at 0 mm stroke value), the approximate resistance value read from the potentiometer outputs was 0.5 kΩ. In the case where the actuator opened the most, i.e., when the end switch was triggered, the approximate value read was 8 kΩ. These readings were obtained by multimeter measurements from the red and black cables coming out of the actuator wiring.

After these measurements, the actuators were initially controlled by deriving a linear equation, but after it was determined that the linear characteristics of the resistance-stroke graphs decreased at the extreme points and the error amount

increased, a fourth-degree polynomial was derived, and the error was greatly reduced and the sensitivity was increased. The last equation derived was derived as a result of measurements taken with the help of a reference resistor circuit designed for resistance measurement via the microcontroller, while all electronic elements of the entire system were connected. In this way, the actuator calibrations were performed while the effects of other electronic elements on the measurement method of the microcontroller were included. Thus, a calibration was performed by taking into account the noise generated in the environment and the deviation value generated in the measurement. In this way, the position controls of the actuators could be provided with a deviation of less than 1 mm and an uncertainty value of ±0.5 mm.

2.2.3 Microcontroller

The Arduino Mega 2560 was chosen for the system because it can do logic operations, hold calibration values, talk to digital indicators, execute commands, show outputs, and work reliably at a specified frequency. It supports shields and low- and high-level programming. Larger size may make it incompatible with some Uno shields. Based on the ATmega2560, the board features 256 KB flash memory, 8 KB SRAM, and 4 KB EEPROM. Programs and work fit in this memory. It can be powered by USB or 7–12V. It has many ground pins and 5V and 3.3V regulators.

Four hardware timers for PWM, interrupts, and counting plus a 16 MHz crystal oscillator keep time on the board. The ATmega16U2 is a USB-serial converter, hence programming requires a USB-B connection. This eliminates the requirement for FTDI. Perfect Arduino IDE compatibility. The Mega 2560 includes 54 digital I/O pins (15 PWM) and 16 10-bit analog inputs. Each pin has internal pull-up resistors and can output 40 mA. Six external interrupt pins enable event-driven operation.

The board works with four hardware UARTs, I2C, and SPI. You can use I2C (pins 20 and 21) to read data from the MPU6050 IMU and SPI (pins 50–53) to control SD cards. Loadcell indicators may talk to a lot of different devices using the different UARTs. There is a reset button, a pin 13 LED, and a reset pin on the board that anybody can use to restart it from the outside. The Mega 2560 is strong, adaptable, and good enough for the job.

2.2.4 Motor Drivers

A reliable motor controller is needed to control the linear actuators in the system in terms of speed and direction. Considering the voltage value at which the actuators operate and the electrical power they can draw, a motor driver that is resistant to high current and voltage and has good cooling properties should be selected. As confirmed by the tests conducted in this context, the BTS7960B motor driver card was found suitable.

BTS7960B is an integrated double-sided H-bridge driver used to provide direction and speed control of DC motors that require high current. It contains two half-bridge circuits, each of which can carry 43A continuous and 55A peak current, thus providing bidirectional control over both ends of the motor. This integrated circuit, which has an operating voltage between 5.5V and 27V, is used in many areas from industrial automation systems to robotic applications. Speed control can be done with PWM (Pulse Width Modulation) signal and can operate at PWM frequencies up to 25 kHz. BTS7960 is also equipped with integrated protection circuits against situations such as overcurrent, overtemperature and undervoltage.

BTS7960 integrated circuit has pins, each with specific functions. IN and INH pins determine the direction and active/passive status of the motor by receiving input signals. An analog signal from the IS pin can be used to measure motor current. Connect the VS pin to the motor's power supply and the OUT pins to its ends. The SR pins are usually the source reference points connected to GND and are pulled to ground with low impedance paths in order to suppress voltage oscillations during high-speed switching. The GND pins provide the reference ground connection of the system. Thanks to the MOSFET structure in the BTS7960, both high-current low-resistance switching is performed and energy losses are reduced.

In a typical application circuit, a full H-bridge structure is created using two BTS7960 integrated circuits. Both integrated circuits are connected to one end of the motor, allowing the motor to move forward and backward. The microcontroller rotates the motor in the desired direction and speed by sending PWM and logic signals to the IN and INH pins. Real-time motor current measurement can be made by connecting the IS pins to analog inputs. The circuit also includes a voltage regulator. This regulator (for example, TLE4278G) feeds the microcontroller by reducing the high voltage source to 5V. The reverse polarity protection circuit in the system protects the entire

system in case of incorrect connection of the motor source. For this purpose, a P-channel MOSFET and diode such as SPD50P03L are used.

In order to ensure the integrity of the system, resistors and filtering capacitors are used before the PWM signals and control pins. This reduces the effect of possible fluctuations in the signal outputs of the microcontroller on the BTS7960. Both driver integrated circuits turn the motor on and off, brake it, and release it at the same time. The technology works like a motor driver that can manage speed and direction completely. The BTS7960B integrated circuit is safe and powerful for heavy load applications since it has protection and power features.

2.2.5 Digital Loadcell Indicators & Signal Amplifiers

2.2.5.1 Digital Loadcell Indicator

To read data from load cells, the load cell outputs in the mV range must be amplified. Various products are available to achieve this. The selected product must be compatible with the load cell and be able to convert the signal from the load cell to the desired values. To avoid incompatibility issues, the M60 load cell indicator, manufactured by the same manufacturer as the purchased load cells, was selected. This indicator has high noise suppression properties and low EMC permeability. Therefore, it can reduce noise from surrounding circuit elements. It is generally recommended for use with the selected load cells.

The WEILO M60 digital load cell indicator is a high-tech electronic indication that is very accurate and dependable for weighing and measuring force in factories. This gadget has a 32-bit ARM CPU and a 24-bit resolution ADC that can sample data at a rate of 3200 times per second. With an internal count resolution of 1/1,000,000, this architecture makes it possible to get very accurate and consistent measurements. It uses its own digital filtering techniques to keep data stable even when things are shaking. The red LED display on the device has 6+6 digits and can show a large range of weights, from -9,999 to +99,999. You can change the refresh rate of the display and the number of decimal places.

The M60 can work with up to eight 350Ω load cells at a time. The signal input range is 0 to 12.5 mV, and the load cell output sensitivity should be between 1.0 and 2.5 mV/V. The supply voltage is 5V DC, and it can use up to 120mA. The drift in the signal is quite small: zero drift is +0.1 µV/°C and gain drift is +5 ppm/°C. The linearity error is only 0.005% Full Scale (FS), which makes the device's calibration and

measurements more reliable. It has optional analog outputs of 4–20mA and 0–10V. You can assign each of these outputs separately, and the linearity drift is 0.05% FS.

The digital input-output interface on the device has three generally open digital inputs and three outputs (transistor type, 24V 250mA). The user can assign several functions to these inputs and outputs. For instance, these digital inputs can be used to start operations like resetting, resetting the peak value, or starting weight control. It also supports features like user-defined alarm limits, peak value detection, and weight control. The device can work with PLCs, computers, or other industrial control systems because it has connectivity choices like RS232, RS485, and CANBUS. These connections make it possible to read and write data rapidly and safely.

The process of calibrating is easy for users. You can choose between zero calibration, data calibration, and load calibration. During load calibration, a known reference weight is used to check the system, and the device figures out the right calibration coefficient on its own. Advanced users can additionally employ 10-point load calibration. This makes it possible to take very precise and straight-line measurements, even in systems that aren't linear. Calibration also lets you enter things like sensor output (mV/V), total load cell capacity, and voltage ratio.

The interface is easy to use and understand. Four buttons make it easy to move through menus, reset the device, or set setpoints. Additionally, security features like locking the keypad and requiring a password keep people who shouldn't be able to access the system from doing so. LED lights on the gadget let you know in real time what the functioning status, alarm warnings, and process statuses are. The M60 indication is small enough to be mounted on a DIN rail. It can work at temperatures from -25°C to +45°C and can handle up to 85% humidity. This device weighs about 167 grams and is made to be used in industrial settings for a long time without any problems. Also, because it meets EMC standards, it is immune to electromagnetic interference. The M60 is a great choice for precision weighing systems that use load cells because of these qualities.

2.2.5.2 Signal Amplifier

To ensure cost-effectiveness in the system, the HX711 circuit was used as a signal amplifier, considering the low error values in S-Type load cells. While these circuits do not provide the same sensitivity as digital indicators and do not offer noise

cancellation or filtering features, they are reliable and common solutions for load cell use.

HX711 has a 24-bit ADC (analog-to-digital converter) that works very well and uses very little power. Very weak data from bridge-type load cells were used to make it. The block diagram shows that the HX711 has two separate channels that can handle differential inputs. These are called INA+ and INA−, and INB+ and INB−. When the bridge is set up, these inputs get data from the load cell. Up to 32, 64, or 128 times, a programmable gain (PGA) amplifier boosts the data that come in. After that, a 24-bit Sigma-Delta (∑Δ) ADC turns the data into digital ones. This structure lets you process analog signals with voltages as low as 1 mV very accurately.

The AVDD pin on the HX711 powers the load cell and is usually stabilized using a 10 µF bypass capacitor. This power source is also used for the reference voltage within. The integrated bandgap reference is connected to the VBG pin and filtered by a 0.1 µF capacitor. The inbuilt voltage regulator keeps all analog and digital signals stable, which cuts down on noise. There is a digital connection between the microcontroller and the integrated circuit. The PD_SCK and DOUT pins are used to sync data. Power management is another usage for PD_SCK.

The HX711 can be set to a sampling rate of 10 Hz or 80 Hz with a single digital pin (RATE). Thanks to its integrated internal oscillator, it eliminates the need for an external clock source, simplifying the system. Load cell connections are typically implemented with four wires (Excitation+, Excitation−, Signal+, Signal−), which are connected directly to the HX711's differential inputs. The HX711 receives these signals, amplifies them, converts them to digital form that can be processed by the microcontroller, and outputs them. Therefore, the HX711 is a very common solution in systems that collect load cell data, thanks to both its simplicity and sensitivity.

2.2.6 Communication Elements

The Weilo Loadcell Digital Indicator can precisely write load cell data to its screen. However, transferring these values to the microcontroller board is critical for data processing. There are two basic methods for achieving this. The first is to read the value proportionally from the device's analog output in amperes or voltage, based on the measured load cell value, and transfer this value to the microcontroller pins. In this case, a voltage output is preferable because the microcontroller's analog pins have voltage reading capabilities. However, two fundamental problems arise. The first is

that the analog pins of the Arduino Mega 2560 microcontroller board can read values between 0 and 5V. A potential difference greater than 5V means that the pin will not function properly or may even burn out. However, the Digital Indicator's analog potential output sends a signal between 0 and 10V. To prevent this problem, a voltage divider circuit must be created using a resistor. Even if this problem is resolved, this application reduces sensitivity by receiving data generated by the voltage difference at the indicator's analog output, which is proportional to the load cell load, rather than receiving the more accurate data filtered by the indicator. Therefore, transferring the values displayed directly from the indicator screen to the microcontroller using the communication ports will increase the sensitivity of the entire system by transferring more accurate data.

To achieve this, communication with the microcontroller is achieved using the TX and GND pins of the indicator, which has an RS232 communication line. The RX pin is not needed because one-way data transfer is sufficient for this system. Data flow is from the digital indicator to the microcontroller. However, to achieve this, the RS232 signal must be converted to a UART signal. This is because the microcontroller does not have the ability to directly read the RS232 communication protocol. The microcontroller is designed to interpret UART signals. To achieve this, an RS232 to TTL Converter module is used. This converter module is connected to the indicator via a female DB9 connector. As can be seen from the pinout of the DB9 connector, pin 3 is the module's RX pin, and pin 5 is the module's GND pin, located in the DB9 connector area. There are two GND pins on the module. One of these pins is located on the DB9 connector, while the other is output via the electronic board. These two GNDs are shared GNDs, which is critical because for the signal to be correctly converted and interpreted, the Arduino, the RS232 to TTL converter, and the indicator must be connected to the same ground. The module's connection to the Arduino is established by drawing 3.3V or 5V from the Arduino to the VCC pin. Additionally, the module's RX pin on the board must be connected to the Arduino RX pin. This may vary depending on the module's internal structure. For example, when using another module, the module's TX pin may need to be connected to the Arduino RX pin. However, following tests with this module, data was successfully received by connecting the RX pin to the Arduino RX pin.

This circuit is an RS-232 level converter that uses Sipex's SP3222E and SP3232E integrated circuits. These integrated circuits make it possible for devices

that work at the TTL/CMOS level, like microcontrollers, to talk to devices that work at the RS-232 level. The SP3222E is known for using very little power and may be turned off to save energy (*controlled by the SHDN pin). The SP3232E, on the other hand, can run all the time.

The TxIN and RxOUT pins on the left side of the circuit let you talk to logic-level devices like microcontrollers. The TxIN pin gets TTL-level data from the microcontroller and sends it to the TxOUT pin, which makes an RS-232 level output. The RxIN pin gets RS-232 level data from the outside world, changes it to TTL level, and transmits it to the RxOUT pin. Drivers and receivers inside the integrated circuit do these transformations. The circuit turns on when the EN pin is pulled low, and it turns off when the pin is pulled high. You may manage low power mode on the SP3222E model with the SHDN (Shutdown) pin. Pulling this pin to Vcc turns on the device.

The device's internal voltage is made by a charge pump circuit that is backed by external capacitors. Pins C1+, C1-, C2+, and C2- control the charge pump, which makes the V+ and V− voltages needed for the RS-232 output levels (around ±10V). This function has four 0.1 µF capacitors (C1–C4) linked to it. There is also a bypass capacitor (C5, 0.1 µF) close to the Vcc input. These capacitors keep the voltage stable and make sure that the charge pump circuit works correctly. Also, a 1000 pF capacitor is suggested to reduce noise on the Tx and Rx lines and make things more stable. This circuit is often used to make sure that TTL level devices, like microcontrollers, can talk to each other reliably with devices that use RS-232 communication, like a computer's serial port. The values and positions of the capacitors employed follow the IC manufacturer's instructions and give the charge pump the dynamic capacitor structure it needs to make positive and negative voltages. This configuration has the benefit of being able to create the needed voltages inside the IC without needing two separate external sources, such ±10V.

2.2.7 SD Card Module

The system includes an SD card module. This is because position control is provided based on the data contained within the SD card module. The complex calculations required for position and stroke control significantly increase microcontroller processing time. Therefore, embedding pre-calculated position data in the SD card as a “.bin” file and then extracting the desired position values from this

data using a specific algorithm speeds up processing time. Furthermore, the equations described in the mathematical model section allow forward kinematic calculations to calculate the final stroke values using the orientation data, pitch, roll, and yaw angles. If orientation values, i.e., pitch, roll, and yaw values, are to be calculated starting from stroke values, inverse kinematic calculations will be required. This requires numerical and iterative solutions, which both slows down the system and reduces accuracy. Instead, using advanced kinematic equations, all stroke and other necessary position data for all pitch, roll, and yaw values within the ±20º range at 1º resolution were calculated using code developed in MATLAB. A total of 68921 lines of data were obtained, and this data was loaded onto an SD card as a ".bin" file named "stroke1.bin". The algorithm was designed to find the "stroke1.bin" file and quickly access the desired line. Details are explained in the Software title.

The image of the SD card module used is shown in the image below, and the connection diagram is explained in the Electronic Wiring section. The most important point to note is that for SD card reading to be successful, the power voltage supplied to the module must be very stable. Small voltage fluctuations can cause SD card data to fail to be read. To avoid this problem, voltage-regulating capacitors can be used in the system. Alternatively, as in this study, the connection should be made directly between the SD card module and the Arduino, without using long cables or components such as breadboards.

2.2.8 Electronical Wiring

The designed electronic system includes 3 linear actuators, 3 motor drivers, 3 4.6kΩ resistors, 3 RS232 to TTL converters, 3 loadcell digital indicators, 3 HX711 amplifiers, 3 S-Type loadcells, 1 multi-axes (3-axes) loadcell, 1 MPU 6050 GY-521 IMU module, 1 SD Card module, 1 power supply, several jumper cables, and multiple breadboards. The connection diagrams for these components are shown in detail in the images below.

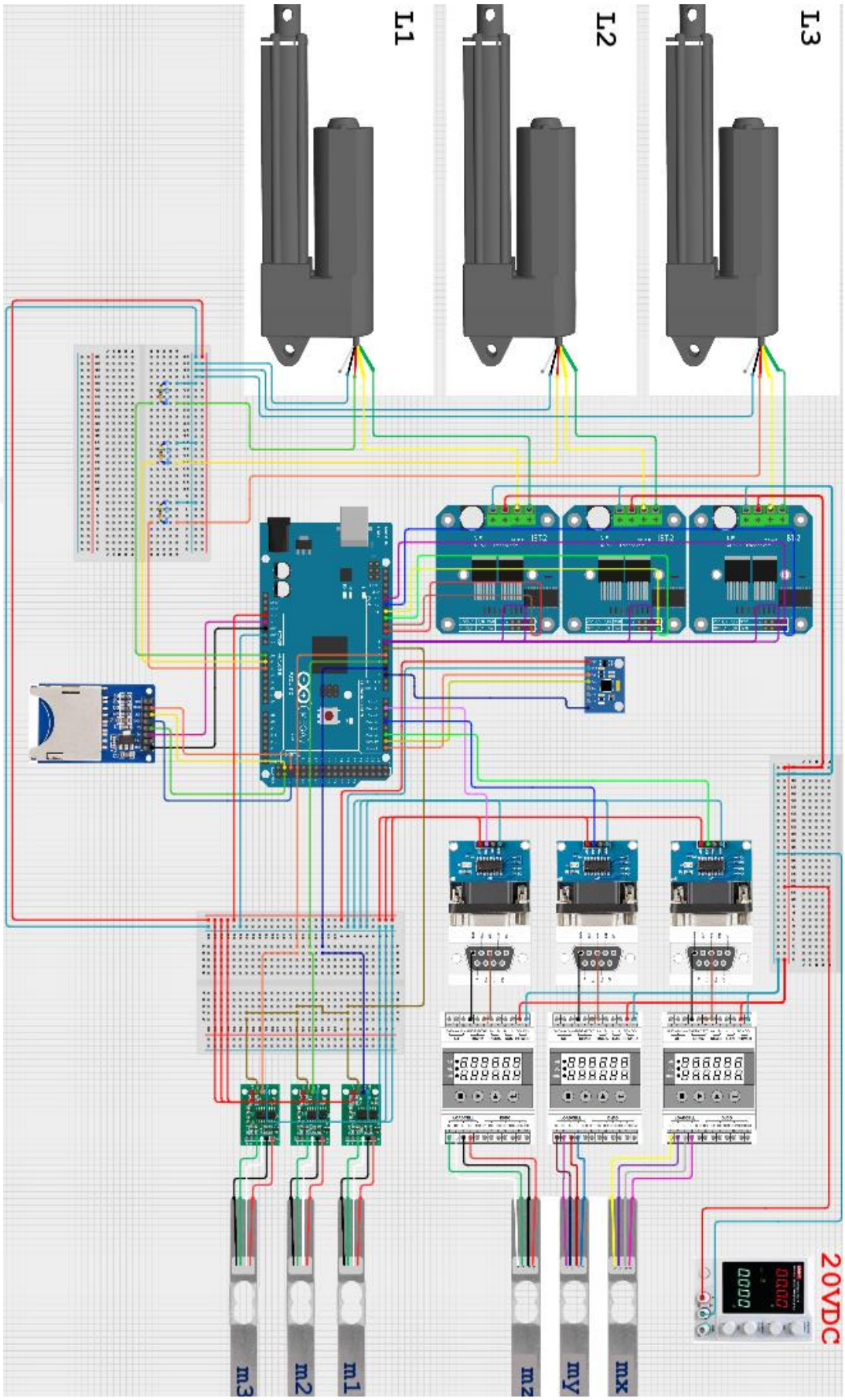

Figure 40: General Electronical Wiring Diagram

The connections for the S-Type load cells and HX711 amplifiers are as shown in the detailed image. The use of the HX711 was deemed sufficient as the circuit to amplify the S-Type load cell output signals. The connections between the three S-Type load cells and the HX711 are as follows:

Table 3: S-Type loadcell and HX711 connections

S-Type Loadcell	**HX711 Amplifier**
Excitation (+) (RED)	E (+)
Excitation (-) (BLACK)	E (-)
Signal (+) (GREEN)	S (+)
Signal (-) (WHITE)	S (-)

After the connection between the S-Type Loadcells and the HX711 amplifiers is complete, the HX711's power and microcontroller connections must be made. First, the power connections are made to the common channel brought from the microcontroller's 3.3VDC output to the breadboard. The ground of the three amplifiers is shared with the microcontroller's ground.

After the power connections are complete, the SCK and DT pins on the amplifiers must be connected. The SCK pins are common on the breadboard and connected to pin D6 of the microcontroller. The DT pins are connected separately to pins D3, D4, and D5. In this order, load cell m1 is connected to pin D3, and load cell m3 is connected to pin D5.

Table 4: HX711 - Microcontroller connection detail

HX711	**Microcontroller**
VCC	3.3VDC Output
SCK	D6
DT	D3 (m1), D4 (m2), D5 (m3)
GND	GND

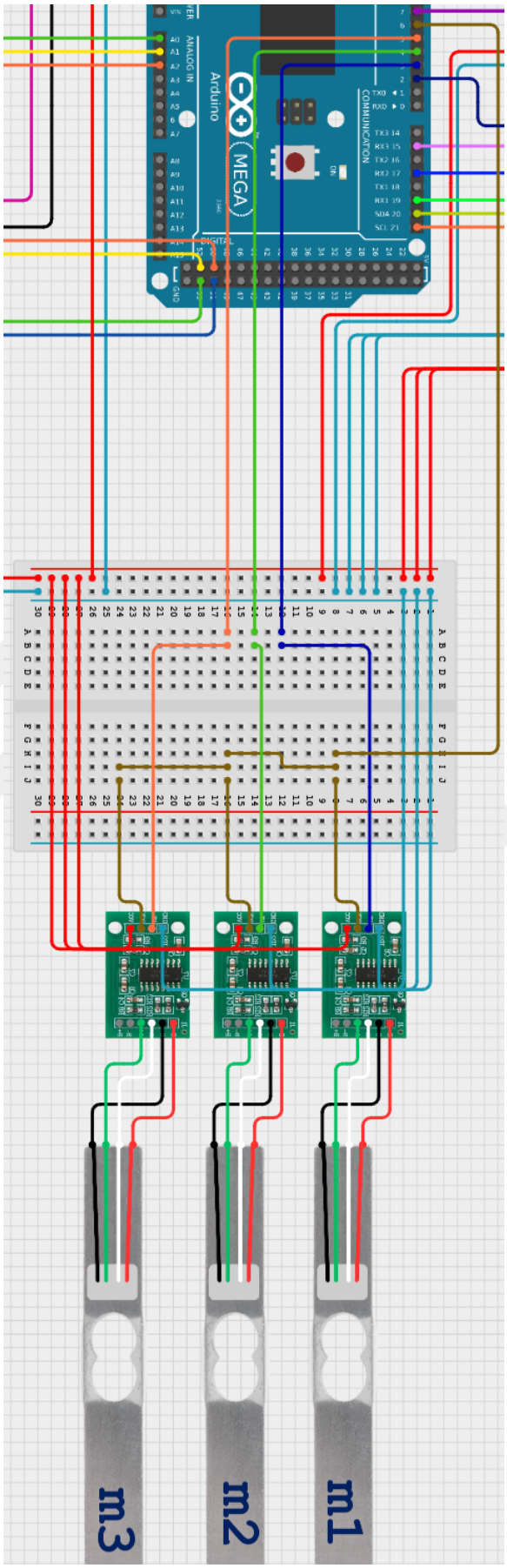

Figure 41: S-Type Loadcells and HX711 modules connection details

Due to the number of axes, categorizing multi-axis (3-axis) load cells based on cable colors is a bit more complex than single-axis load cells. The categorization of multi-axis load cells based on cable colors is as follows:

Table 5: Multi-axes loadcell cable colors

X-Axis	**Y-Axis**	**Z-Axis**
Pink (E+)	Blue (E+)	Red (E+)
Grey (E-)	Brown (E-)	Black (E-)
Purple (S+)	Blue & Red (S+)	Green (S+)
Yellow (S-)	Pink & Grey (S-)	White (S-)

When examining the resulting circuit diagram, it will be noticed that the load cell signals positive and signal negative pins are not always connected to match the indicator signal positive and signal negative pins. While the X-axis signal wires are connected to match the indicator, the reverse is true for the Y-axis and Z-axis. This is because the mechanical assembly of the system ensures consistent data delivery according to the axis set specified in the mathematical model. To ensure positive data readings from the indicators in the Y- and Z-axis directions are achieved according to the axis set specified in the mathematical model, the signal connections are reversed. The resulting connection diagram is as follows:

Table 6: Multi-axes loadcell & indicator connection detail

X-Axis → Indicator	**Y-Axis → Indicator**	**Z-Axis → Indicator**
Pink (E+) → (E+)	Blue (E+) → (E+)	Red (E+) → (E+)
Grey (E-) → (E-)	Brown (E-) → (E-)	Black (E-) → (E-)
Purple (S+) → (S+)	*Blue & Red (S+) → (S-)*	*Green (S+) → (S-)*
Yellow (S-) → (S-)	*Pink & Grey (S-) → (S+)*	*White (S-) → (S+)*

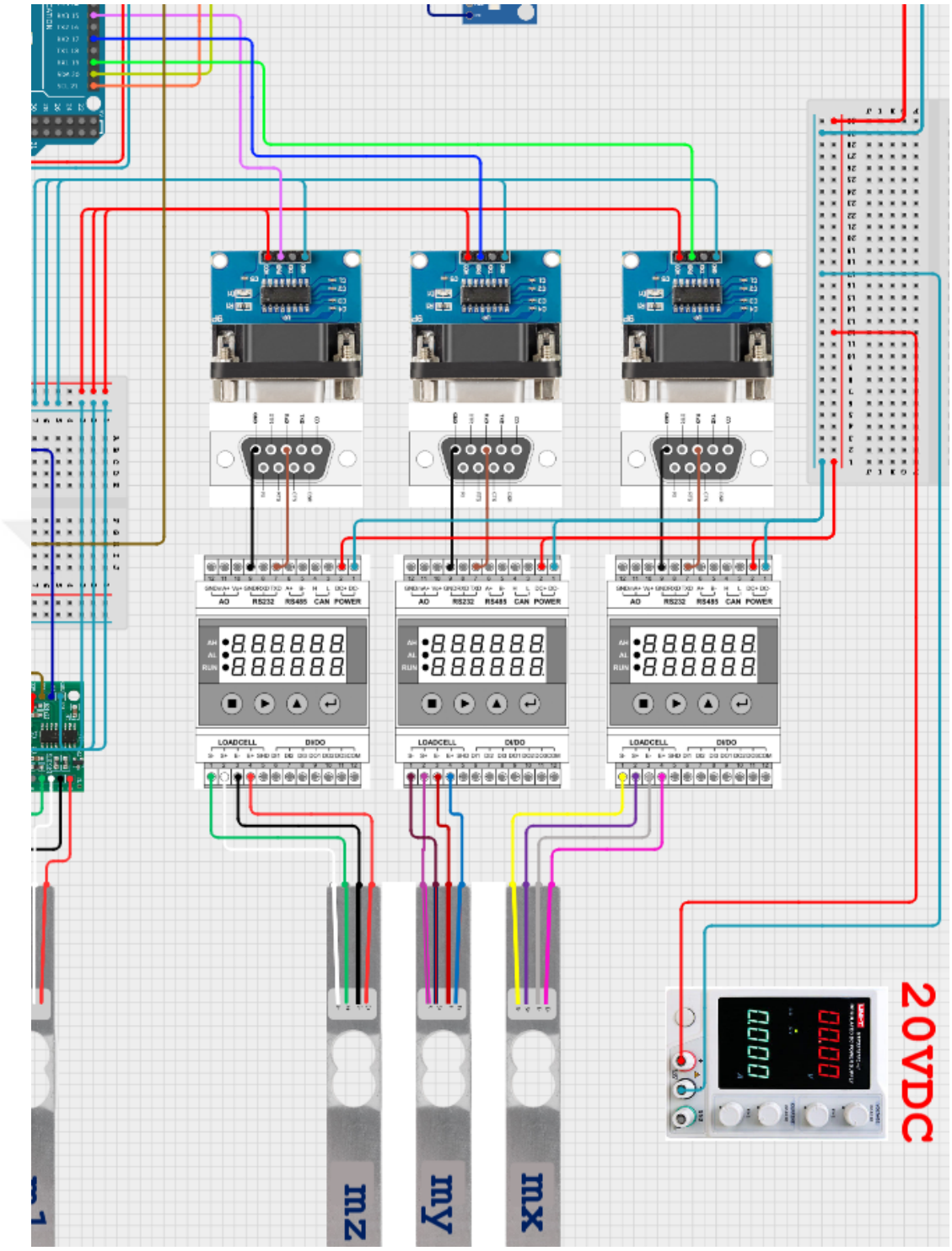

Figure 42: Multi-Axes Loadcell, Digital Indicators, and RS232 communication elements connection details

Indicators used are products that can operate with a 12-24VDC supply. They are powered by 20VDC by matching the potential difference they will use with the linear actuators in the system. The 20V potential difference received from the power supply is the element that operates both the linear actuators and the digital indicators.

After the connections between the indicators and load cells are completed, to transfer the data to be measured using the indicators to the microcontroller, the signals received from the RS232 communication line must be converted to TTL signals and transmitted to the Arduino digital communication pins. This is accomplished using RS232 to TTL converters. The TX pin on the indicator's RS232 port is connected to pin 3 (RX) of the converter's DB9 connector. The ground pin on the indicator port is connected to ground pin 5 on the DB9 connector. The VCC and GND power connections on the converters are then connected to the common channel drawn from the microcontroller's 3.3VDC output. After these connections are completed, the connections that will ensure the actual data flow to the microcontroller are made from the RX pins on the RS232 to TTL converters' electronic boards to the microcontroller's RX pins 1, 2, and 3. This entire connection diagram is summarized in the table below:

Table 7: Communication protocol convertor wiring

Mx Convertor	**My Convertor**	**Mz Convertor**
DB9 Pin 3 → Indicator (X) TX	DB9 Pin 3 → Indicator (Y) TX	DB9 Pin 3 → Indicator (Z) TX
DB9 Pin 5 Indicator (X) → RS232 GND	DB9 Pin 5 Indicator (Y) → RS232 GND	DB9 Pin 5 Indicator (Z) → RS232 GND
PCB VCC → Microcontroller 3.3VDC	PCB VCC → Microcontroller 3.3VDC	PCB VCC → Microcontroller 3.3VDC
PCB GND → Microcontroller GND	PCB GND → Microcontroller GND	PCB GND → Microcontroller GND
PCB RX → Microcontroller D19 (RX1)	PCB RX → Microcontroller D17 (RX2)	PCB RX → Microcontroller D15 (RX3)

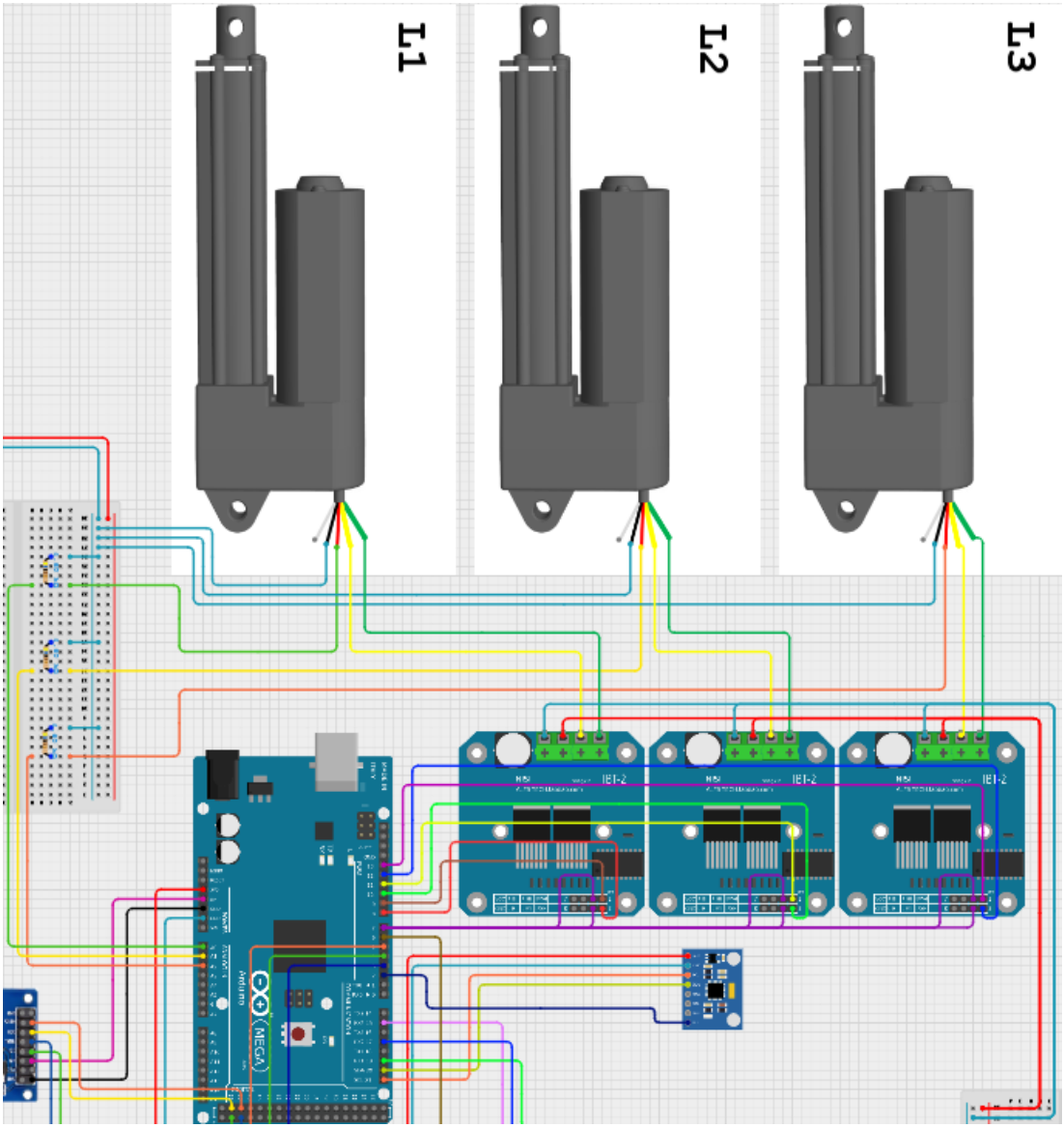

Figure 43: Linear Actuators, Motor Drivers, and Resistors' connection details

Motor drivers operate linear actuators in the system, while potentiometers inside the actuators measure their locations. You can use a circuit with reference resistors to measure the potentiometer resistance from the microcontroller's analog pins. This will tell you the current resistance values of the potentiometers.

The potentiometer resistance is the resistance between the black and red wires of the linear actuators. A 4.6kΩ reference resistor is utilized to do measurement, which takes into account the potentiometer's 0–10kΩ range.

The circuit for this measurement is designed as follows. First, a 3.3V potential from the Arduino is connected to one leg of the reference resistor. A line from the other leg of the resistor is drawn to both the red wire of the actuator and the analog pin on the microcontroller where the voltage measurement is to be taken. The black wire

of the actuators is connected to the same ground channel as the microcontroller. This completes a circuit between 3.3V and GND, with the reference resistor and the potentiometer inside the actuator remaining between them. A series current line is created through both the reference resistor and the potentiometer. The Arduino analog pin, connected by a jumper from the line after the reference resistor, measures the voltage drop across the reference resistor. This measurement is provided in values from 0 to 1023, a 10-bit range. When these values are proportioned to the potential difference, the reading becomes the voltage. Dividing the voltage reading by the current flowing through the potentiometer calculates the potentiometer resistance. However, to calculate this current value, another calculation is required. This can be achieved because the current flowing through the reference resistor and the potentiometer is known to be equal. When the 3.3V potential flows through the reference resistor, it drops to the voltage reading on analog pin A0. Therefore, the potential difference between the reference resistor's input and output becomes a known value. Since the reference resistor is also known, the current value can be calculated directly. This circuit is called a voltage divider.

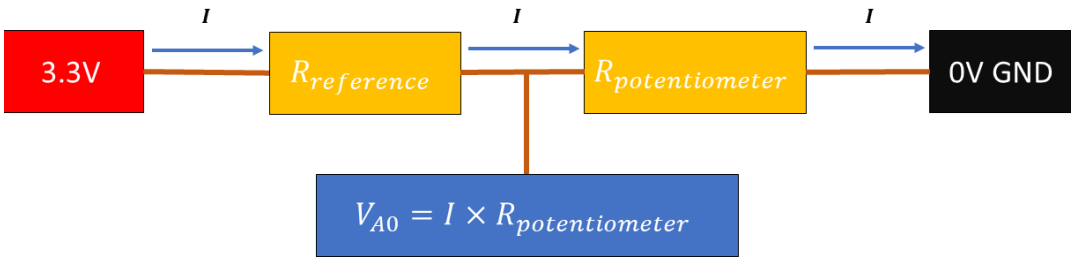

Figure 44: Voltage divider for resistance measurement

The two equations that should be used to calculate the potentiometer resistance in the voltage divider circuit are given below:

$$3.3V - V_{A0} = I \times R_{reference} : find\ current$$

$$V_{A0} = I \times R_{potentiometer} : find\ potentiometer\ resistance$$

While significant progress has been made by measuring the resistance of potentiometers, the actual connections that will move the actuators will be between the motor drivers, actuators, and microcontrollers. Motor drivers have positive and negative ports that can receive power from the power supply. Next to these ports are

the positive and negative terminals to which the actuator motor is connected. Motor drivers control the transfer between these two power connections, reflecting the logical algorithm they implement. These ports can preferably be reversed. This will only affect the direction of rotation of the motor. In this study, the connections were reversed, and this connection type was taken into account in the software environment. The connections between the motor drivers, power supply, and linear actuators are shown in the table below:

Table 8: Connections among actuators, motor drivers and power supply

Actuator 1 (L1) → Driver1	**Actuator 2 (L2) → Driver2**	**Actuator 3 (L3) → Driver3**
Green → Motor (-)	Green → Motor (-)	Green → Motor (-)
Yellow → Motor (+)	Yellow → Motor (+)	Yellow → Motor (+)
Power Supply (20V) → Driver1	**Power Supply (20V) → Driver2**	**Power Supply (20V) → Driver3**
20V → Vout	20V → Vout	20V → Vout
0V → Vin	0V → Vin	0V → Vin

The connection between motor drivers and microcontroller is shown in the table below:

Table 9: Connections between motor drivers and microcontroller

Driver1 → Microcontroller	**Driver2 → Microcontroller**	**Driver3 → Microcontroller**
L_EN → D7	L_EN → D7	L_EN → D7
R_EN → D7	R_EN → D7	R_EN → D7
LPWM → D8	LPWM → D10	LPWM → D12
RPWM → D9	RPWM → D11	RPWM → D13

The L_EN and R_EN pins are used to activate motor drivers to rotate left and right. Setting Digital 7 to a HIGH value activates all motors in both directions. The LPWM and RPWM pins control the direction and speed of the motor driver's connected motor. These pins must be connected to microcontroller pins capable of processing PWM signals, and since each motor's left and right rotation speed will be

controlled separately, they must be connected to separate pins. These conditions are met with pins D8-13.

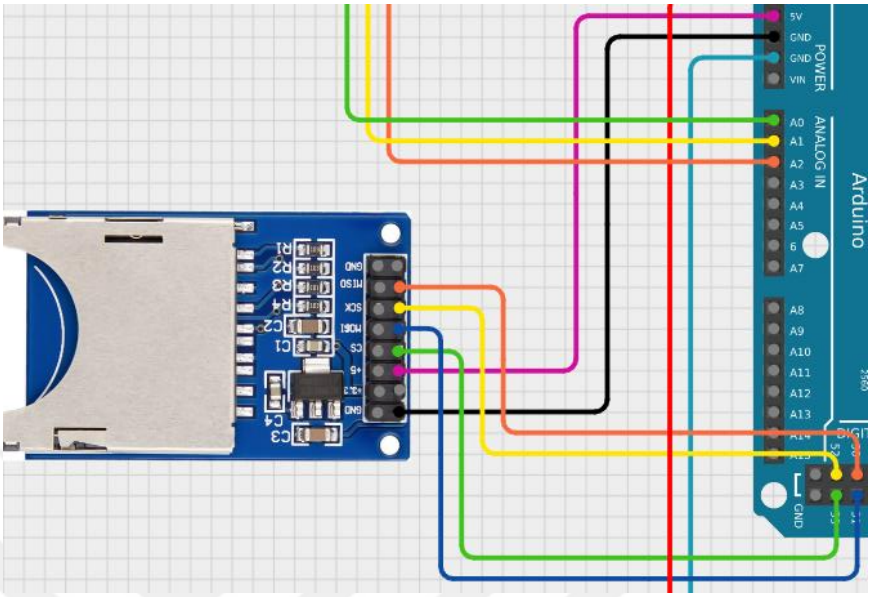

Figure 45: SD Card Module connection details

The SD Card module requires a stable voltage supply. To ensure this, it is connected directly to the Arduino 5V and GND pins. Breadboard interference or excessive cable lengths can cause the SD card to fail to read. Other connections are made as shown in the table below.

Table 10: SD Card module wiring

SD Card	Arduino
5V	5V
GND	GND
CS	D53
MOSI	D51
SCK	D52
MISO	D50

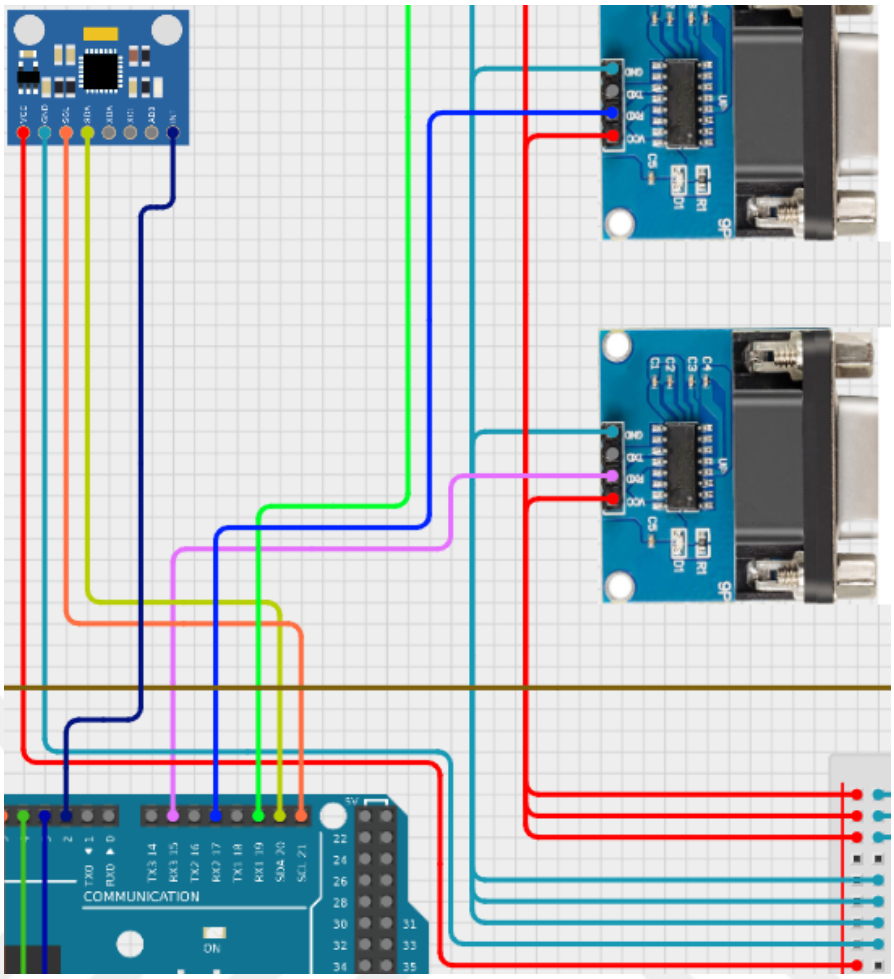

Figure 46: MPU6050 GY-521 IMU Module connection details

The IMU connections are implemented using I2C ports. Power is provided at 3.3V for compatibility with the MPU 6050 module. Additionally, the interrupt pin is used for more precise measurement. Connection details are provided in the table below.

Table 11: IMU wiring

MPU 6050 GY-521	**Arduino**
VCC	3.3V
GND	GND
SCL	SCL – D21
SDA	SDA – D20
INT	D2

2.3 SOFTWARE DESIGN OF THE SYSTEM

2.3.1 Algorithm

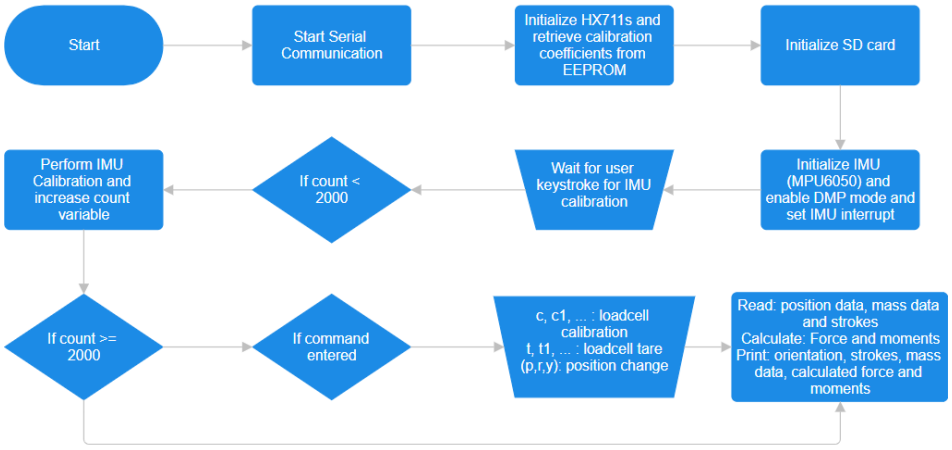

Figure 47: General algorithm

The resulting algorithm from the developed code is briefly summarized in Figure 47. When the system wakes up, it first activates the Serial Communication HX711 ADCs, SD card, and IMU. It reads the load cell calibration factors from the EEPROM and initializes the load cells according to these factors. The user is then prompted to enter a command by pressing any key to initiate the IMU calibration. When the user presses any key, the IMU calibration begins, and the calibration is completed when the count reaches 2000. The system now continuously reads position data, load cell data, and stroke values. Force and moment calculations are continuously performed, and all desired outputs are printed to the screen. If any of the embedded commands is entered within the code, the system executes the operations defined by that command and then continues to print the calculated data to the screen. Load cell calibration, tare operations, or position control can be performed at any time by issuing a command to the system.

2.3.2 Global Settings

The entire software is broken down into code blocks and shown in Appendix 1. The first part of global settings (Code Block 1) is where all the libraries needed for the system to work are listed and some global definitions are made. The BLA namespace gives access to the BasicLinearAlgebra.h library, which makes it easier to work with matrices and vectors. EEPROM.h allows data such as calibration to be preserved even after the device is rebooted. HX711.h controls the ADC module, which

reads data from load cells, while the SPI.h and SD.h libraries are required for reading files from the SD card. Motor drivers are cotroled by using BTS7960.h library. DMP (Digital Motion Processor) features of the MPU6050 IMU sensor can be used by using the I2Cdev.h and MPU6050_6Axis_MotionApps20.h files. It is necessary to add the Wire.h file to use the I2Cdev library with Arduino Wire. This is so that you can use functions like Wire.begin(). #define OUTPUT_READABLE_YAWPITCHROLL specifies that output is provided in Euler angles (yaw, pitch, roll) later in the code. The #define INTERRUPT_PIN 2 line specifies that the MPU6050's external interrupt pin is connected to pin 2 on the Arduino. The MPU6050 mpu; line creates an instance of the MPU6050 sensor. Finally, the float buffer[5]; array is defined to temporarily hold the stroke values read from the SD card. This structure forms the basic infrastructure for the system's communication with the hardware and control operations.

In Code Block 2, the basic point locations of the system's physical structure are defined as matrices. All locations are expressed in three-dimensional space (x, y, z) coordinates and defined using the BasicLinearAlgebra (BLA) library. Ag, Bg, and Cg are fixed ground points and represent the positions of the actuator's lower connection points in the system coordinate system. These points are usually fixed to the base. In contrast, the A, B, and C matrices are the actuator's upper connection points and are connected to the moving platform; these points allow the platform to move. The C_g vector indicates the location of the test model's center of mass. The test model to be used must be designed so that its center of mass coincides with this point. This data is important for both force and moment calculations because the locations of the center of mass and force application points are directly used in the mathematical model. After these vectors, the dot matrices are converted to vector matrices and rearranged to form r1, r2, r3, and rcg. This version makes it easier to perform linear algebra tasks such as rotating and multiplying power vectors. To facilitate system manipulation, these structures are copied and moved using transformation matrices.

The third code block defines the global constants and variables required for managing and processing data received from the MPU6050 IMU sensor. First, the lastPrintTime and printInterval variables are defined to print data to the serial monitor at regular intervals; this structure makes the data flow more readable and controllable. dmpReady is the flag used to indicate the successful initialization of the MPU6050's DMP (Digital Motion Processor) module. mpuIntStatus and devStatus are used to check the IMU's interrupt status and whether the initialization process was successful.

The packetSize variable holds the length, in bytes, of the data packet expected from the DMP; typically, 42 bytes. fifoCount and fifoBuffer[] are defined to store the current state and data in the IMU's FIFO (First In, First Out) data buffer. Variables such as drift_comp, last_update, and drift_offset are used to compensate for sensor drifts that may occur during extended use. The basic structures used in orientation calculations are the Quaternion q, the gravity vector gravity, the accelerometer measurements aa, the gravity-free acceleration aaReal, the acceleration aaWorld with respect to the world coordinate system, and the euler[] and ypr[] arrays holding the heading angles. ypr[] specifically contains the Euler angles; yaw, pitch, and roll of the system and defines the system's position. The last_yaw, last_pitch, and last_roll variables are used to store the reference positions during calibration. The count variable counts the number of samples taken during this calibration process. Finally, the teapotPacket[] array is a special data packet used in InvenSense's demo applications and can be used for visual connection or testing with the system. All of these structures form the predefined infrastructure for the IMU to generate and process data correctly.

This section defines the global structures required to read the three single-axis S-type loadcells used in the system via the HX711 ADC module. The `scales[3]` array defines an `HX711` object for each loadcell, allowing for separate input management. The `calibration_factors[]` array holds predetermined calibration coefficients for each loadcell, allowing digital signals to be directly converted to mass in kilograms. The `tare_values[]` array is the zero point (empty value) of the loadcell and is used to calculate the net value in weight measurement. The `m_values[]` array contains the current measured net mass values. The `mx`, `my`, and `mz` variables hold the magnitudes of the forces calculated from the triaxial loadcell data received via RS232; these data are the components of the total force passing through the origin. "zeroCountX," "zeroCountY," and "zeroCountZ" count how many zeros the RS232 has sent in a row. This is used to check if the loads are "really zero" when the signal or link is lost for a brief time. The "dtPins[]" array shows how to link the data (DT) pins of the Arduino to the HX711 modules. The `SCK_PIN` pin sends the identical clock signal to all of the HX711s. The `ZERO_THRESHOLD_COUNT` parameter tells the RS232 how many zeros it has to see in a row before it can figure out how much force there is. This fixes problems that come up out of nowhere or only last a short period.

This section contains the settings for the resistance measurement system (Code Block 5), which is implemented to provide position feedback via potentiometers within three linear actuators. The constants sensorpin1, sensorpin2, and sensorpin3 indicate that the actuator potentiometer outputs are connected to the analog input pins (A0, A1, and A2) on the Arduino. Analog voltage values obtained from these pins are calculated through the variable resistor within the actuator and converted into position information. The Rref value is the value of the fixed reference resistor connected in series with the potentiometer and is defined as 4.6 kΩ. Using this resistance, the voltage divider method is used to find the potentiometer's resistance value. Vin is defined as 3.3 V which will be used to find the unknown resistance value. A linear or polynomial function is then used to turn the resistance number into the length of the actuator stroke. This helps you always know where the motor is. This set-up is the simplest way to get input for precise control of the positions of actuators.

Code Block 6 discusses the necessary hardware and configurations to enable BTS7960 dual-channel motor drivers to control three linear actuators. "Stroke1," "Stroke2," and "Stroke3" are the parameters to be controlled. "targetStroke1," "targetStroke2," and "targetStroke3" inform the actuators of what they should be able to do as the stroke numbers vary. "StartStroke" will tell where to travel and how far to go. There are starting points in it. The Arduino's digital pins control the driver. Speed and direction of each motor can be controled with the "L_PWMx" and "R_PWMx" pins. To turn on the motors "EN" pin is set to HIGH. `L_PWM1` and `R_PWM1` regulate the left and right PWM inputs of motor1, respectively. The BTS7960 class designates each motor driver as an object named "motor1," "motor2," or "motor3." These components allow the software to transmit commands directly to the actuators. The amplitude of the PWM signal regulates the motor's speed, while the "maxPWM" and "minPWM" constants set limits for the motor. This allows the system to accurately control position by varying the actuator's speed and direction according to the target stroke. Setting the values to -1 means there is no initial target. This configuration creates a closed-loop motion control system that collects information about the motor's location.

Code Block 7 defines the parameters related to the SD card, receiving input from the serial monitor, and RS232. The chipSelect variable determines which pin to receive SD card data. The file to be read is in ".bin" format. The inputString variable is initially defined to receive input from the serial monitor. The inputReady variable is

created to control the input status. The "bufferX," "bufferY," and "bufferZ" strings are defined here for later parsing.

2.3.3 Setup

The setup section of the code begins with Code Block 8. First, the necessary serial communications are configured to transfer data at the specified frequency. The Serial.begin function is defined as a baud rate of 500,000 because the rapid communication between the Arduino and the computer via the USB port prevents the Arduino's temporary memory from draining quickly and thus preventing FIFO overflow caused by the rapid accumulation of IMU data. Other serial communication frequencies are matched to the baud rate set for the load cell indicators in RS232 communication. To ensure a more frequent data flow, the highest baud rate for the indicators, 115,200, was selected, and the indicators' data transfer frequency was set to 1000 Hz. Tests have shown that these values are suitable.

The setup phase was performed in Code Block 9 to read the data from three S-Type Loadcells. This code, written within a for loop where the variable "i" ranges from 1 to 3, automatically reads the load cell's previous calibration data from the EEPROM. The validity of the read value is checked, and if it is not valid, the default value is used. The default calibration value is not one that will ensure accurate load cell measurements, but this step is designed to ensure the sequence continues. Later, the error in the read data can be determined from the printed data, and calibration can be performed. Finally, to ensure the load cells start with a value of "0," 30 samples are taken to determine the average "0" point. The system begins printing the readings, dated at this value.

An attempt is made to read the SD card with Code Block 10. If the reading is successful, the following strings are printed on the screen: "SD Card is ready"; if not, "SD Card could not be found."

In Code Block 11, the steps described in the setup section of the Interrupt Detection routine are performed. The I2C clock is initially set to 400 kHz. With serial communication enabled, informational text is printed to the screen, the interrupt pin is switched to input mode, and the connection is checked with the mpu.testConnection() function. Data reading during calibration is performed using the while loops in this block. Additionally, the MPU's initial offset coefficients are set here.

Code Block 12 makes sure that the MPU6050 sensor's DMP (Digital Motion Processor) module is set up correctly. If devStatus is zero, it signifies that the DMP has started up correctly and the sensor is ready to start collecting orientation data in real time. The mpu.setRate(39) command then sets the frequency at which the sensor generates data to 25 Hz. To turn on the DMP, mpu.setDMPEnabled(true) function is used. After that, the Arduino's interrupt system is turned on, and the attachInterrupt() function makes sure that the dmpDataReady function is executed when the MPU6050's interrupt pin (INTERRUPT_PIN) sends a rising edge signal. This ensures that the main loop is notified of the arrival of a new data packet. The mpu.getIntStatus() function reads the DMP's interrupt status, and the main loop is notified of the system's readiness with the dmpReady flag. Finally, the size of the data packet generated by the DMP is retrieved using mpu.dmpGetFIFOPacketSize() and stored in the packetSize variable. This information is used later to retrieve the correct amount of data from the FIFO buffer. If the DMP fails to initialize, an error code is printed over the serial port; this error is typically caused by a failed initial download or failure to transmit configuration data to the sensor. This structure is the final checkpoint that allows the system to safely begin generating IMU data.

2.3.4 Loop

A loop contains operations that the system will perform continuously in a loop. These operations include continuously receiving data from all sensors, triggering different conditions based on commands entered via the serial monitor, initiating sensor calibration, providing motor control, and performing calculations based on the content of the mathematical model.

The beginning of this loop() function includes the operations related to receiving data from the IMU sensor and processing it. Firstly, in Code Block 13, the dmpReady flag checks whether the system's DMP (Digital Motion Processor) module has been successfully initialized; if the system is not ready, the function exits without performing any action. After this process, the mpuInterrupt flag is set until data is received from the sensor. Once sufficient data is accumulated, the flag is reset, and the current IMU data is retrieved with the mpu.getIntStatus() function. The current amount of data in the FIFO buffer is then updated with the mpu.getFIFOCount() function. If the FIFO overflows (1024 bytes full) or the interrupt status contains an overflow flag, the mpu.resetFIFO() command resets the FIFO, preventing data loss. If there is no

overflow and the DMP has provided new data (interrupt 0x02), the data packet in the FIFO is read until it reaches the expected packet size. Then, a data packet is read from the FIFO using mpu.getFIFOBytes(). This packet is written to the fifoBuffer[] array, which is stored in memory for later analysis. If there is more than one packet in the FIFO, only one is read, and fifoCount is decremented accordingly. This allows data to be retrieved in the next loop without any additional waiting. Finally, the section starting with the #ifdef OUTPUT_READABLE_YAWPITCHROLL line specifies that the received data will be converted to Euler angles (yaw, pitch, roll) and used.

Code Block 14, in short, receives various types of IMU data, but only uses yaw, pitch, and roll values. During calibration, the count variable provides calibration for 1999 readings. During calibration, a YPR array is printed on the screen. As the number of readings increases, the values become more stable and converge to a single value. Once this convergence is achieved, position data can be accurately captured.

With Code Block 15, the formulas in the mathematical model section begin to be applied. First, the read MPU values are assigned as integers to the p, r, and y values, respectively, to represent the pitch, roll, and yaw angles in accordance with the system orientation and IMU placement. The distance between the designated points on the fixed and mobile platforms is calculated in strokes. Then, the data from the S-Type load cells are multiplied by the gravitational acceleration to convert them from mass to force, resulting in the magnitude values of the reaction forces. The transformation matrix is then calculated using the p, r, and y variables obtained from the IMU. These calculations are performed in accordance with the format of the BasicLinearAlgebra.h library. By multiplying the calculated position vectors with the transformation matrix, the coordinates of the points on the mobile platform after the orientation change are found using operations performed with respect to the ground coordinate system. The transposed helicals of the position vectors define those points. The next step is to calculate the unit vectors of the reaction forces using the newly found point positions and find their orientations. This process is also performed in Code Block 16. The coordinates of the points on the fixed platform were also used in calculating the unit vectors of the reaction forces. These points are the points initially defined in the global settings section. The reaction force at the origin is calculated separately from the other reaction forces. To calculate the vector form of the R0 force, defining the values from the separate axes from the triaxial load cell as the components of the R0 reaction force is sufficient to define the vector form of this force. By summing the reaction forces,

Pforce, whose x, y, and z components are side, drag, and lift forces, respectively, can be easily calculated. Then, to perform the cross-product operations required for moment calculations, the position vectors, reaction forces, and Pforce force are defined as an array in accordance with the crossProduct function defined in the final sections of the code. Then, cross product operations are performed and the results are defined in the cross_rcg_Pforce, cross_r1_R1, cross_r2_R2, cross_r3_R3 arrays.

After the calculations, the moment vector Mg is calculated by summing the cross-product results in Code Block 17. The desired result, the "M" vector, is calculated by multiplying this vector by the transformation matrix.

Code Block 18 calls the defined updateStrokes() and controlactuator() functions within a loop. The updateStrokes() function keeps the stroke values updated with the resistance value read, while the controlactuator() function provides actuator control when a command for position change is entered into the serial monitor.

The pitch, roll, and yaw values defined in this code block are different from the IMU data. These values are the variables used to enter the desired orientation values in position control. After the initial definitions of these pitch, roll, and yaw values, the while loop that receives input is also shown in Code Block 18.

In Code Block 19, if loops are designed to perform different operations for different command types written to the serial monitor. Initially, the inputReady flag being set to true indicates that the command has been entered. Resetting this flag prevents repeating the same operation. Resetting the flag is achieved by setting the inputReady value to false. When the user enters a command, any blank characters at the beginning or end of the command are cleared with the inputString.trim function. The commands that can be entered, specifically in Code Block 18, are c, t, c1, c2, c3, t1, t2, and t3. The "c" symbol indicates load cell calibration, while the "t" symbol indicates load cell taring. If no number is added to the end of the letter used, that function is performed for all load cells. For example, entering "c" will initiate the calibration process for all load cells. Entering "t" will tare all load cells. Adding a number to the end of these commands will ensure that only the load cell with that number performs the desired function.

The input format for position control is defined in Code Block 20. The desired command format here is "(p,r,y)." The pitch, roll, and yaw values entered are expected to be integers. The angle restriction of ±20° in all directions, determined by the mechanism's movement limits, also applies here. Angles within this range are required

in all directions; otherwise, position control will not be achieved. Instead, a warning is printed on the screen stating, "Values must be between -20° and 20°."

Code Block 21 plays a critical role in position control, enabling the relevant line of the ".bin" file located on the SD card to be read. The data is arranged in the .bin file as shown in Table 12. Each line represents a different orientation condition. Although only stroke values are displayed in the table, three components of points Ag, Bg, Cg, A, B, C, and C_g have also been embedded in the SD card for use if needed. Therefore, there are 27 columns in a row. Access to these other lines can be achieved by defining "float buffer[6]" as "float buffer[27]" in the global settings.

Table 12: SD Card data

Pitch	Roll	Yaw	Stroke1	Stroke2	Stroke3
-20	-20	-20	88,20152	113,3151	-5,7718
-20	-20	-19	89,65324	112,934	-5,17561
-20	-20	-18	91,10331	112,5522	-4,54853
…	…	…	…	…	…
-20	-19	-20	89,6722	111,1665	-4,06219
-20	-19	-19	91,09013	110,7913	-3,44144
-20	-19	-18	92,50638	110,416	-2,79037
…	…	…	….	…	…
-19	-20	-20	86,27621	114,0006	-3,53393
-19	-20	-19	87,72146	113,6562	-2,9722
-19	-20	-18	89,16574	113,3109	-2,38001
…	…	….	…	….	….

Regardless of the buffer size we specify, the calculations must be based on the data on the SD card. When reading from a ".bin" file, the file will be searched based on the most recent bytePos value. To perform this calculation, we need to know that each column in each row, when the data is float, takes up 4 bytes. Since there are 27 columns in total, the size of each row is 27 x 4 = 108 bytes. So, if we can find the row in which the desired data is located using an index, we can search within the ".bin" file using the byte size obtained by multiplying this index by 108.

The index calculation is performed as follows. Since the data is sorted as in Table 12, the yaw value changes in each row. The roll value changes every 41 rows. The pitch value changes every 41 x 41 = 1681 rows. Furthermore, there are 41 x 41 x 41 = 68921 rows in total. For this reason, the row number containing the desired data can be calculated with the following formula: “`long index = (pitch + 20) * 1681L + (roll + 20) * 41L + (yaw + 20);`”. In this formula, all pitch, roll, and yaw

values obtained are first shifted by adding the value 20, because the row number cannot be less than 0. The value obtained by multiplying the entered pitch value by 20 plus 1681 represents the row in which the entered pitch value first appears in Table 12. Similar operations are then performed for the roll and yaw values and the sum of all the values found to determine the row number. By multiplying the row number by the byte size of each row, the byte at which the data begins is determined, and the seek function locates the desired data.

After Code Block 21 completes its main function of reading the SD card, it simply determines the target stroke values, ensures movement to these values, and prints the necessary information to the screen. At the end of the loop, the inputString value is reset to allow for further commands.

Code Block 22 simply determines the number of times three load cell data read on the HX711s will be read and averaged to be printed on the screen. Based on the tests conducted, and considering the system's operating speed, two readings were deemed sufficient. Furthermore, the net value is calculated by subtracting the value recorded during the tare phase from the read value. This net value is printed on the screen and similarly included in the calculations. The m_values array is created for this purpose. After the HX711 readings are completed, the readRS232 function is executed to read another load cell type. This function allows the data from the three-axis load cell to be read via the RS232 communication protocol. Details of this function are detailed under the "Functions" section.

Code Block 23 is a code block designed primarily for printing data to the serial monitor. Controlling the print frequency is critical for preventing FIFO overflow problems, as printing to the serial monitor can significantly slow down the loop if implemented incorrectly. The print interval is controlled by the currentMillis value, which is controlled by the lastPrintTime and printInterval variables defined in the global settings. The structure seen in the code allows printing when the specified printInterval value is exceeded, thus controlling the print frequency.

Furthermore, using the Serial.println command repeatedly slows down the loop significantly. A single string named data is defined for this purpose, and all data is embedded within this string. Using the Serial.println function only once in each loop increases the code's efficiency and speed. Code Block 23 displays the data planned to be printed.

CHAPTER III
VALIDATION OF 6 COMPONENT WIND TUNNEL FORCE BALANCE

3.1 CALIBRATION

3.1.1 Calibration of Linear Actuators

Calibration of linear actuators is critical to ensuring position control accuracy. The primary objective is to accurately measure stroke using potentiometers and ensure the actuators reach the desired stroke value. To ensure this, the experimental environment was designed with all electronic systems active. This was to include any potential noise-induced deviations from the electronics into the calibration phase. The system shown in Figure 48 was designed to achieve this. Stroke values were verified as read from the serial monitor, and actual stroke values were measured using a 1mm resolution tape measure, and the two measurements were compared.

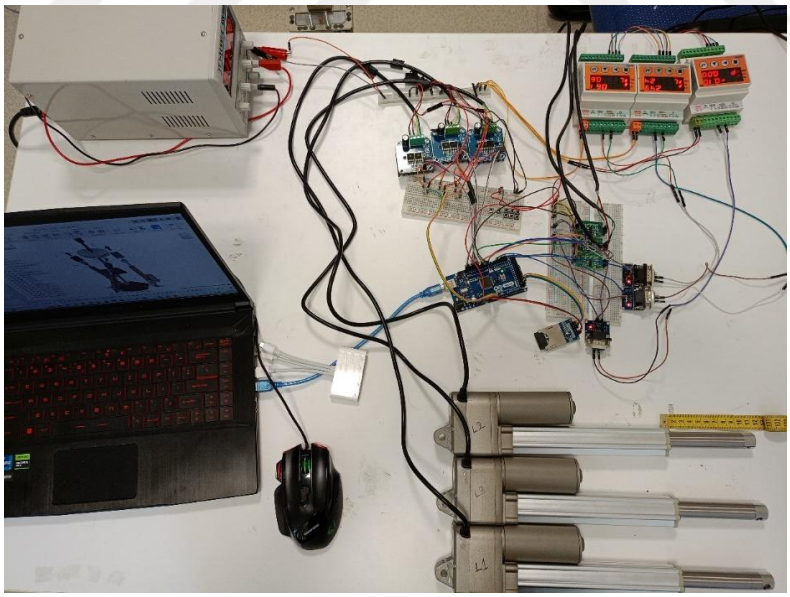

Figure 48: Linear actuator calibration setup

First, a linear equation was derived from manual measurements, which provides the stroke output corresponding to the actuator's resistance input. This equation was found as " $Linear\ Calibration\ Equation: Stroke = 20.238 \times resistance - 8.905$" However, when using this equation, the error size can reach

the order of 5 mm, especially as the stroke approaches its minimum and maximum points. Tables 13, 14 and 15 show the data and errors obtained as a result of the Linear Calibration Equation.

Table 13: Linear actuator 1 measurements for linear calibration equation

Command mm	L1 mm	L1 Real mm	Resistance Ohm	Error mm
5	4,68	1	0,67	3,68
10	10,21	6	0,94	4,21
15	15,01	10	1,18	5,01
20	20,01	17	1,43	3,01
25	24,94	22	1,67	2,94
30	30	28	1,92	2
35	35,19	33	2,18	2,19
40	40,15	38	2,42	2,15
45	44,8	42	2,65	2,8
50	49,76	48	2,9	1,76
55	54,68	53	3,14	1,68
60	59,5	58	3,38	1,5
65	65	64	3,66	1
70	70,12	69	3,9	1,12
75	75,47	75	4,17	0,47
80	80,1	80	4,4	0,1
85	84,97	85	4,64	-0,03
90	90,11	90	4,89	0,11
95	95,55	96	5,16	-0,45
100	100	101	5,38	-1
105	104,64	106	5,61	-1,36
110	109,51	111	5,89	-1,49
115	115,36	117	6,14	-1,64
120	119,96	122	6,37	-2,04
125	125,77	127	6,61	-1,23
130	129,79	133	6,85	-3,21
135	135,03	139	7,11	-3,97
140	139,59	144	7,34	-4,41
145	144,32	150	7,57	-5,68

Table 14: Linear actuator 2 measurements for linear calibration equation

Command mm	L2 mm	L2 Real mm	Resistance Ohm	Error mm
5	4,68	1	0,67	3,68
10	10,01	6	0,93	4,01
15	15,01	12	1,18	3,01
20	20,01	17	1,43	3,01
25	25,19	22	1,68	3,19
30	29,72	27	1,91	2,72
35	35,19	32	2,18	3,19
40	39,83	37	2,41	2,83
45	45,14	42	2,67	3,14
50	49,76	48	2,9	1,76
55	55,46	53	3,18	2,46
60	59,92	58	3,4	1,92
65	65,09	63	3,66	2,09
70	70,12	68	3,9	2,12
75	74,97	73	4,14	1,97
80	79,57	78	4,37	1,57
85	84,97	84	4,64	0,97
90	90,11	88	4,89	2,11
95	94,93	94	5,13	0,93
100	100	99	5,38	1
105	104,64	104	5,61	0,64
110	109,51	110	5,85	-0,49
115	114,61	115	6,1	-0,39
120	119,18	120	6,33	-0,82
125	124,77	126	6,61	-1,23
130	129,79	131	6,85	-1,21
135	135,03	136	7,11	-0,97
140	139,59	142	7,34	-2,41
145	144,32	147	7,52	-2,68

The maximum stroke length is 150 mm due to the selected actuator. Therefore, measurements were taken at 5 mm intervals between 5 mm and 145 mm during calibration. The reason for not taking measurements at the minimum and maximum stroke values is that limit switches are activated at these points.

The "Command" value in Tables 13, 14, and 15 represents the command given to the actuator, while the "L" and "Resistance" values include the values read from the serial monitor. The "L Real" values include the results of measurements taken with a tape measure. Errors are the difference between the "L Real" and "L" values.

Table 15: Linear actuator 3 measurements for linear calibration equation

Command mm	L3 mm	L3 Real mm	Resistance Ohm	Error mm
5	5,05	2	0,69	3,05
10	10,21	8	0,94	2,21
15	14,79	12	1,17	2,79
20	19,54	17	1,41	2,54
25	25,19	23	1,68	2,19
30	30	27	1,92	3
35	35,19	33	2,18	2,19
40	39,83	38	2,41	1,83
45	44,8	42	2,65	2,8
50	50,13	48	2,92	2,13
55	55,07	53	3,16	2,07
60	59,92	58	3,4	1,92
65	64,21	63	3,61	1,21
70	69,65	69	3,88	0,65
75	74,97	75	4,14	-0,03
80	80,1	80	4,4	0,1
85	85,53	85	4,67	0,53
90	90,11	90	4,89	0,11
95	94,93	95	5,13	-0,07
100	99,99	100	5,38	-0,01
105	104,64	105	5,61	-0,36
110	109,51	110	5,85	-0,49
115	114,61	116	6,1	-1,39
120	119,96	121	6,37	-1,04
125	124,77	126	6,61	-1,23
130	130,64	132	6,9	-1,36
135	135,03	137	7,11	-1,97
140	139,59	142	7,34	-2,41
145	145,29	149	7,62	-3,71

When a higher-order equation was used instead of a first-order equation to reduce the error rate, particularly at the minimum and maximum stroke values, it was observed that fourth-order equations reduced the error rate to the desired levels. Polynomials that best approximated the data points in the graphs in Figures 49, 50, and 51 were plotted, and the polynomial equations were saved for insertion into the software.

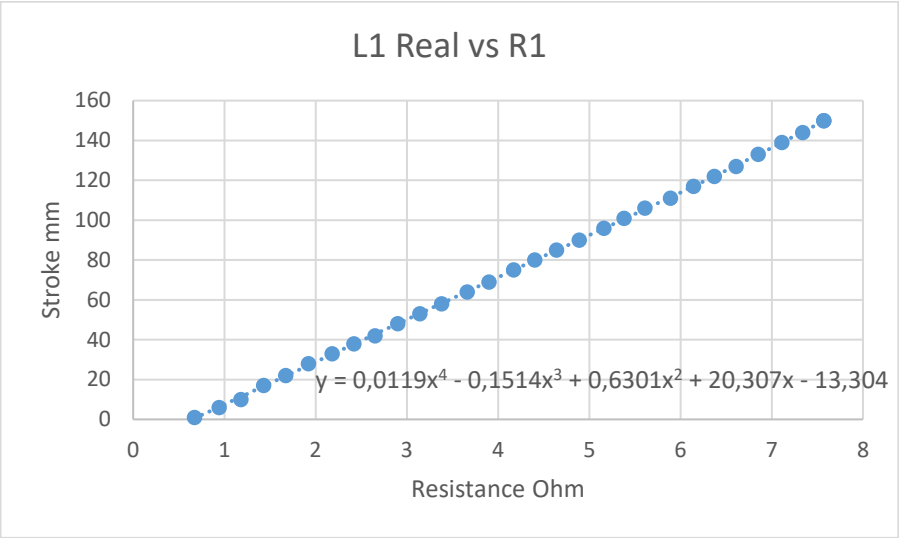

Figure 49: Linear actuator 1, 4th degree calibration equation generation

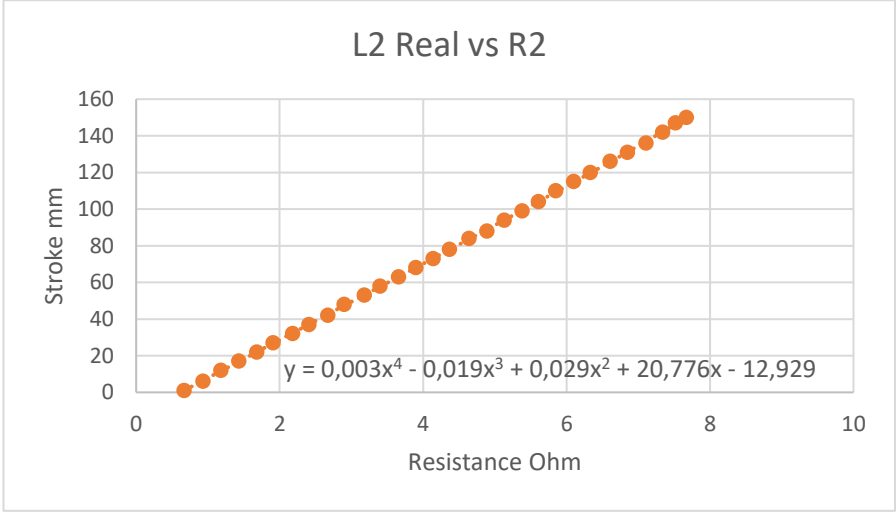

Figure 50: Linear actuator 2, 4th degree calibration equation generation

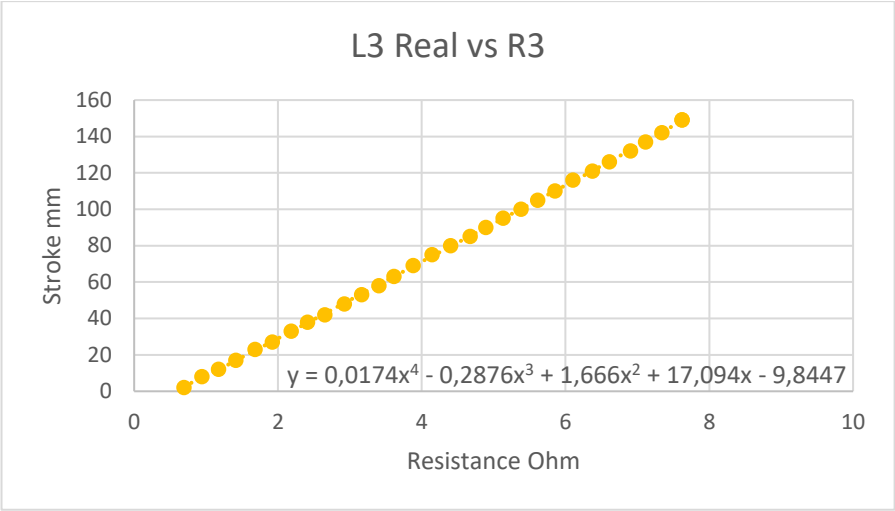

Figure 51: Linear actuator 3, 4th degree calibration equation generation

The graph in Figure 52 was obtained by superimposing the data points in Figures 49, 50, and 51. This graph shows that the stroke behavior of the three different actuators is similar depending on the resistance. This means that the same calibration equation can be used for different actuators. However, in this study, to maximize precision, specific equations were used for each actuator.

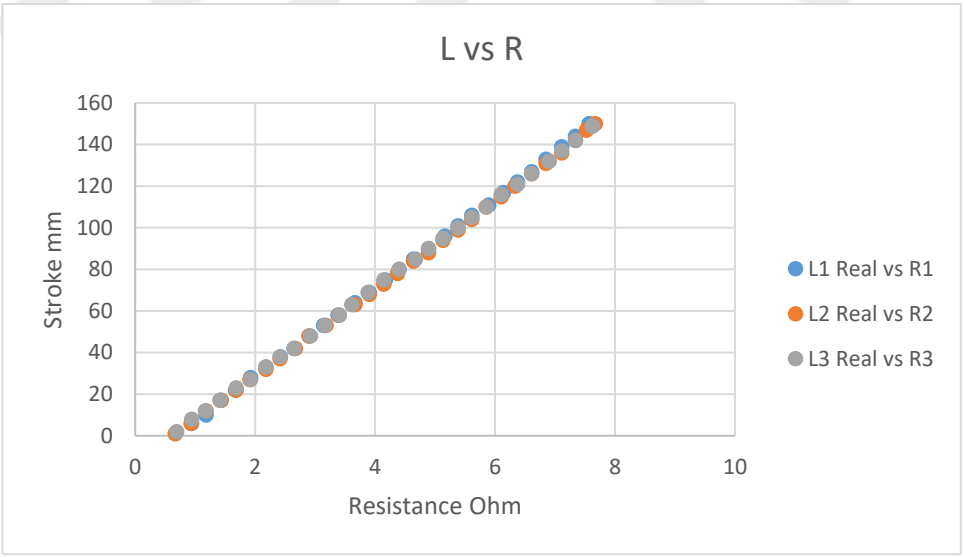

Figure 52: All actuators in one graph

As a result, the tests were repeated using the 4th-degree calibration equations generated at the end of the study. It was observed that the error in the measurements was mostly reduced to less than 1 mm.

Table 16: Linear actuator 1 calibrated data

Command mm	L1 mm	L1 Real mm	Resistance Ohm	Error mm
5	5,69	6	0,91	-0,31
10	9,99	10	1,12	-0,01
15	14,88	15	1,35	-0,12
20	20,2	20	1,6	0,2
70	69,84	70	3,93	-0,16
75	75,98	76	4,22	-0,02
80	80,31	80	4,42	0,31
135	134,89	135	6,94	-0,11
140	139,97	140	7,16	-0,03
145	145,35	145	7,38	0,35

Table 17: Linear actuator 2 calibrated data

Command mm	L2 mm	L2 Real mm	Resistance Ohm	Error mm
5	5,28	5	0,88	0,28
10	9,66	10	1,09	-0,34
15	14,85	15	1,34	-0,15
20	20,27	20	1,6	0,27
70	70,17	70	4	0,17
75	74,76	74	4,22	0,76
80	80,15	80	4,48	0,15
135	135,11	134	7,02	1,11
140	140,12	139	7,25	1,12
145	147,57	146	7,57	1,57

Table 18: Linear actuator 3 calibrated data

Command mm	L3 mm	L3 Real mm	Resistance Ohm	Error mm
5	4,93	5	0,81	-0,07
10	9,74	10	1,06	-0,26
15	15,12	15	1,33	0,12
20	19,89	20	1,56	-0,11
70	69,74	69	3,93	0,74
75	74,29	74	4,14	0,29
80	79,62	79	4,4	0,62
135	135,24	135	7,02	0,24
140	140,18	140	7,25	0,18
145	146,5	146	7,52	0,5

3.1.2 Calibration of Loadcells

Loadcell calibration can be performed in three different ways. The first, and most preferred, is achieved by establishing a linear relationship between the weight and the loadcell output by using the output value of the loadcell loaded with a known weight and the value output when there is no weight on the loadcell. The second method can be achieved by creating an exponential relationship using multiple

weights. Another method is by directly entering a predetermined calibration coefficient into the loadcell data acquisition system.

In the current system, calibrations were performed using the first method mentioned. Calibrating with a weight close to the likely measured values will increase accuracy. Therefore, calibration was performed at levels close to the estimated loads that would occur on the loadcells. To facilitate calibration with different loads, loadings were performed using a scale. However, to verify the scale's accuracy beforehand, a known weight of 0.4 kg was used to compare the scale's readings, as shown in Figure 53. Although the scale's resolution is 0.05 kg, calibration accuracy should be evaluated accordingly.

Figure 53: Scale verification

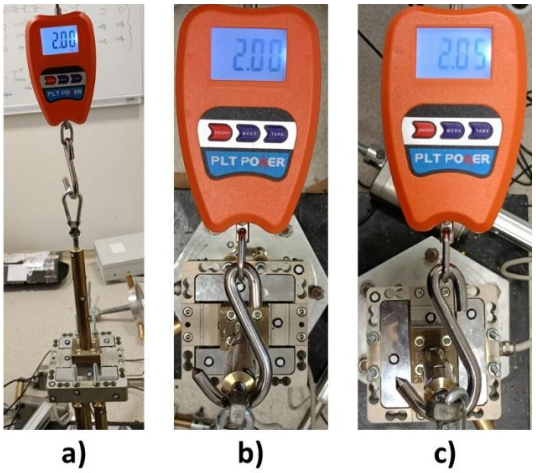

Figure 54: Three axes loadcell calibration: a) Z axis, b) Y axis, c) X axis

All load cell calibrations were performed under a load of approximately 2 kg. First, the triaxial load cell was calibrated separately for all three axes. To ensure angular accuracy, data changes were observed in real-time while loading was being applied to one axis, and the calibration coefficient was recorded when no readings were available in the other axes. A reading on only one axis indicates that loading was occurring only in that axis direction. Figure 54 shows the calibration moments of the triaxial load cell.

S-Type loadcells are calibrated when subjected to the weight of the linear actuator and connecting elements to which they are attached. Since weight readings can be taken using a scale, there's no need for calibration using a standard weight. Calibration can be performed simply under any load. The key is to input the scale value into the software in calibration mode via the serial monitor.

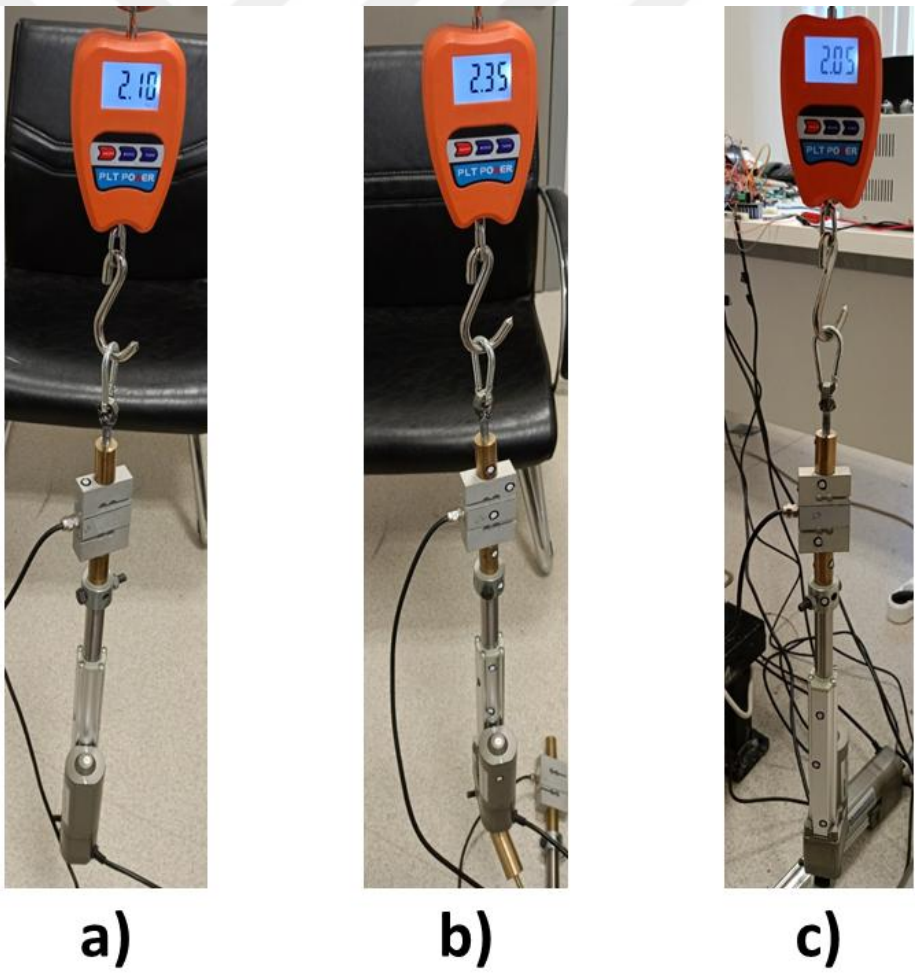

Figure 55: S-Type loadcells calibration: a) Loadcell1, b) Loadcell 2, c) Loadcell 3

As a result of the calibration, the triaxial load cell calibration coefficients were recorded in the digital indicator. The calibration factors for the S-Type load cells were recorded in the Arduino EEPROM as 42583.8046 for Loadcell 1, 43092.3437 for Loadcell 2, and 43852.0976 for Loadcell 3 for direct use later.

3.1.3 Calibration of Inertial Measurement Unit

The MPU6050 inertial measurement unit is an electronic circuit capable of performing calibration and filtering operations internally. Within the circuit's proprietary closed-source firmware, the DMP (Digital Motion Processor) algorithm performs filtering and sensor fusion operations. This algorithm utilizes complementary filtering, low-pass filtering, high-pass filtering, and bias estimation algorithms, providing a consistent output with these data correction methods implemented internally by the MPU6050.

The sensor fusion algorithm combines accelerometer and gyroscope data on a frequency basis. Accelerometer data is reliable for low-frequency, long-term data because it is sensitive to the gravity vector. However, the accelerometer is more prone to short-term drift due to vibration and impact. Gyroscope data, on the other hand, can detect short-term, high-frequency movements much more accurately. However, gyroscope data is also prone to drift due to time. Combining the strengths of these two units yields more stable data. When the current system performs these filtering operations, the initial data is unstable. However, after a certain number of data points are received, the readings become stable, and the stationary IMU reads constant values. To test the MPU6050's measurement accuracy, the system was first woken up and the count variable was counted up to 2000, allowing the data to stabilize, as shown in Figure 56.

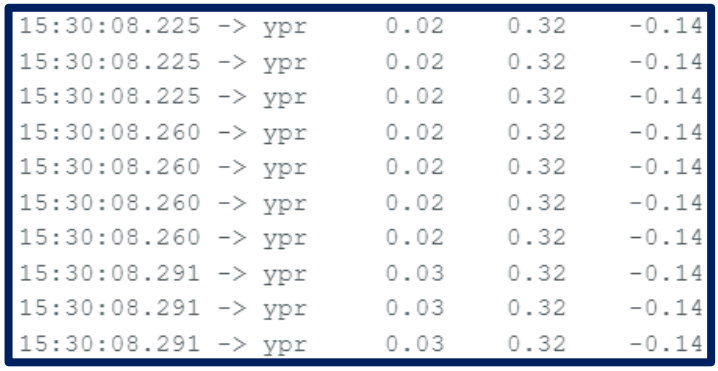

Figure 56: Stabilized IMU data.

During the verification process, the IMU was positioned at 0°, 10°, and 20° on all axes using a protractor, and data from the serial monitor was recorded. The measurements confirmed that the IMU could provide results better than 1°. Figures 57-62 show how the measurements were performed and the results. Measurements were made while the MPU6050 was attached to the fixture used to mount it to the force balance system. This allowed the IMU's orientation to be verified by aligning the fixture's reference surfaces with the corresponding surfaces of the positioned protractor.

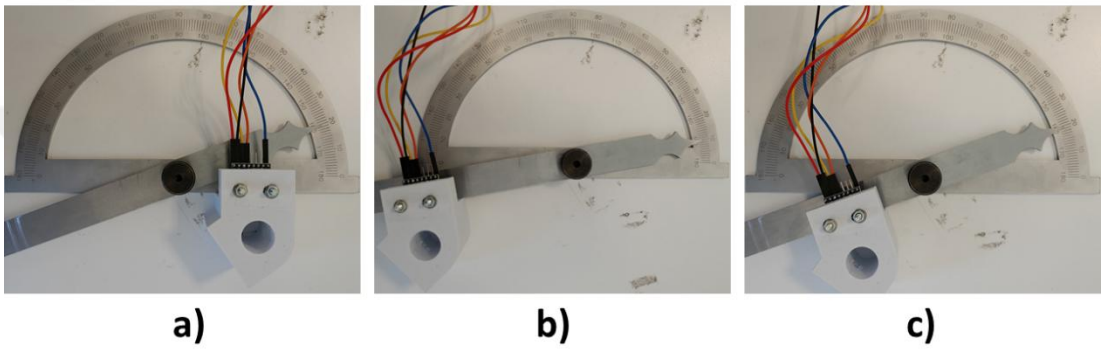

Figure 57: Yaw verification: a) Yaw = 0, b) Yaw = 10º, c) Yaw = 20º

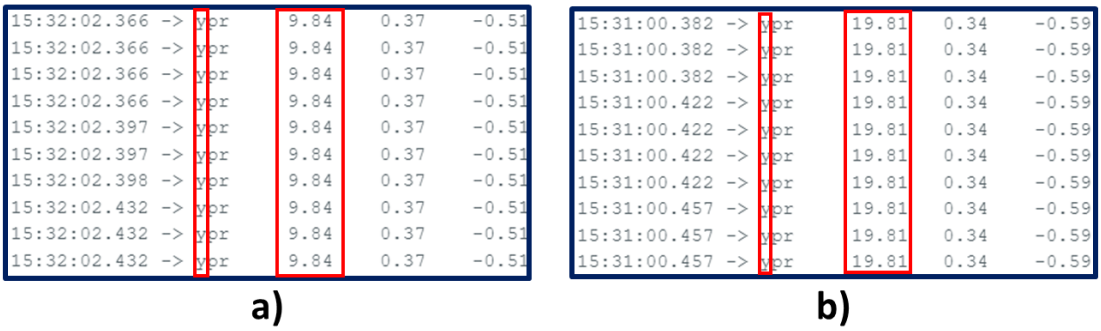

Figure 58: Yaw measurement: a) y = 9.84º, b) y = 19.81º

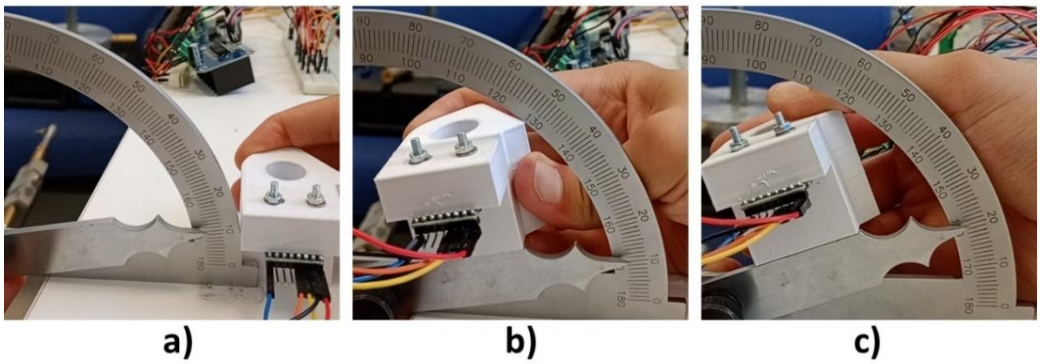

Figure 59: Pitch verification: a) Pitch = 0, b) Pitch = 10º, c) Pitch = 20º

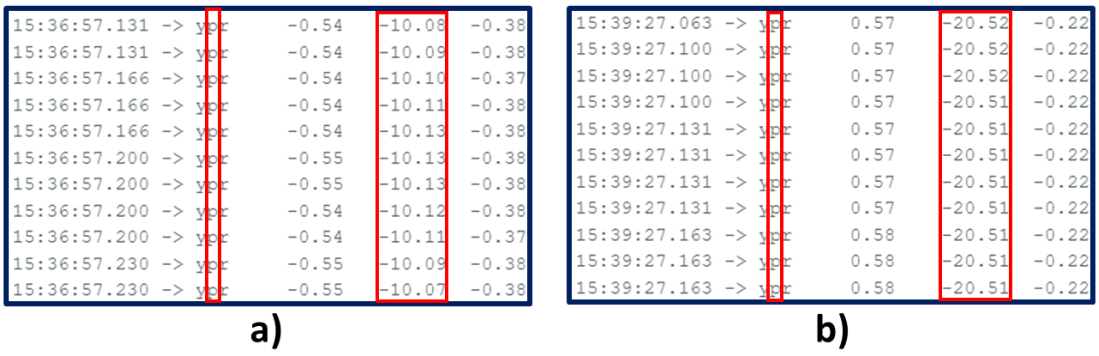

Figure 60: Pitch measurement: a) p = -10.1º, b) p = -20.5º

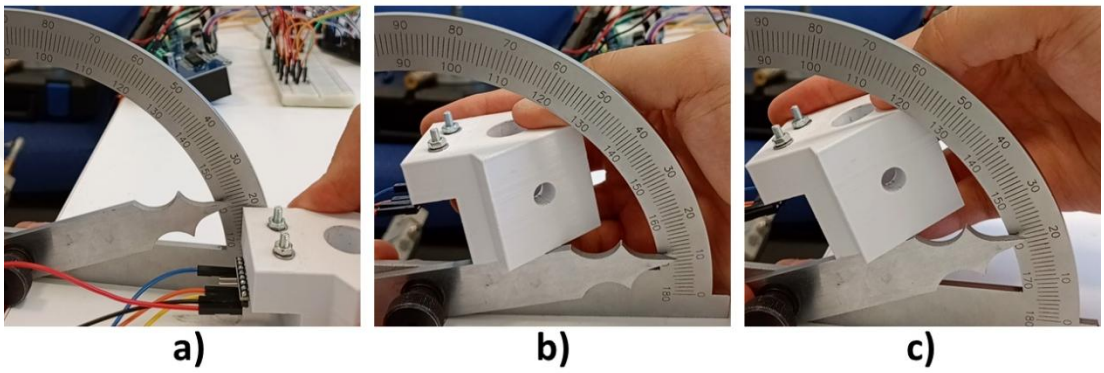

Figure 61: Roll verification: a) Roll = 0, b) Roll = 10º, c) Roll = 20º

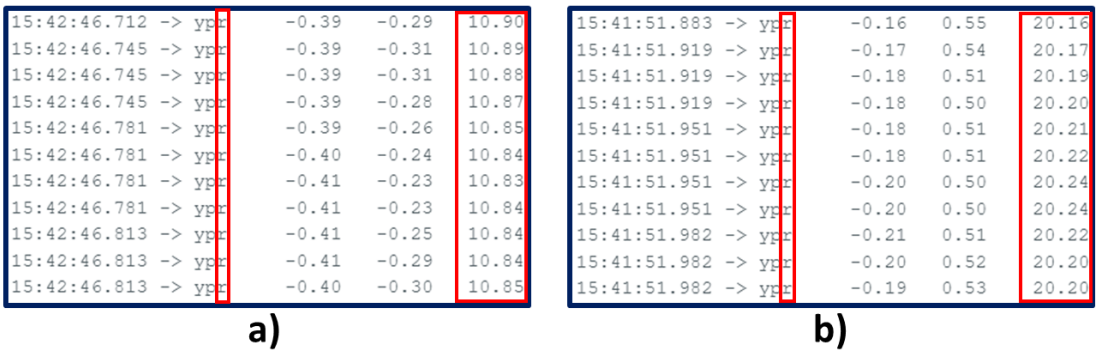

Figure 62: Roll measurement: a) r = 10.8º, b) r = 20.2º

As a result, yaw measurements were made with an error of 0.2º at 10º and an error of 0.2º at 20º. Similarly, for pitch measurements, the error was 0.1º at 10º, and for 20º, the error was 0.5º. In roll measurements, the measurement errors at 10º and 20º were 0.8º and 0.2º, respectively.

3.2 MEASUREMENTS

To verify the force balance mechanism, a setup was designed to take measurements under specific loads. The first element of this setup is a fixture attached to the end of the sting of the force balance mechanism, which generates a moment around the point determined as the center of gravity in the mathematical method section when a weight is hung or a tensile load is applied. This fixture can be seen in Figures 63 and 65. This fixture has two arms extending the x and y axes. The intersection of these two arms was selected as point Cg, and the code was manipulated accordingly during the experiment. The starting coordinate of the Cg point was determined as (0, 180, 324) after the fixture was installed. Therefore, during the verification, the 167.5 mm position of Cg on the y axis was updated with the fixture's installation.

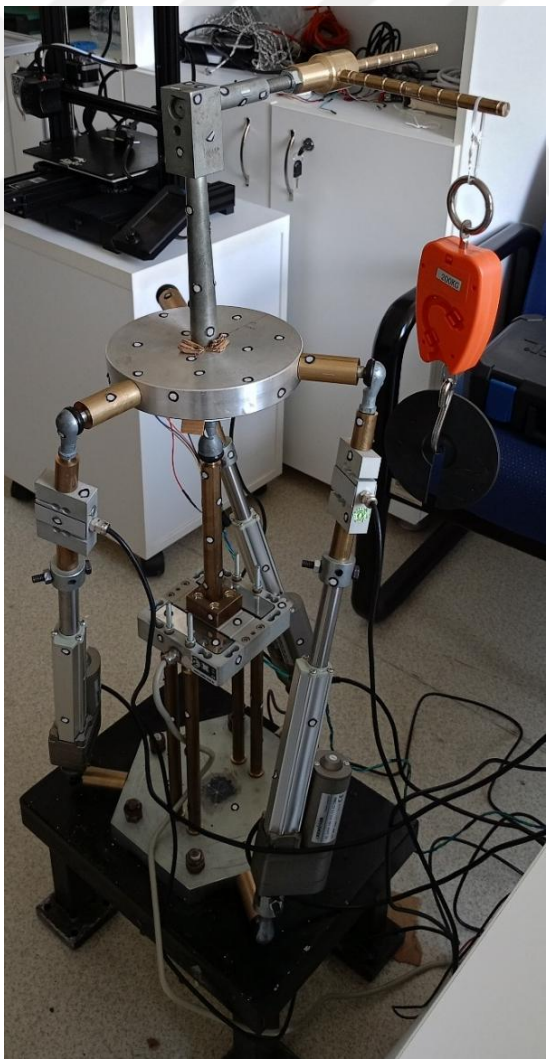

Figure 63: Measurement at (0,0,0). Force is applied on “z” axis and moment created on “x” axis.

Figures 63 and 65 summarize the experimental methods. Generally, the system was loaded in different orientations, and readings were taken from the serial monitor in six axes. The primary outputs required for readings are force, moment, and orientation angles. However, other data was also printed repeatedly to the serial monitor for control purposes against any potential errors during measurement. These printed control data include the individual outputs of each load cell, the current stroke positions of each actuator, and the vector representations of each reaction force. These readings were printed as shown in Figures 64 and 66, and all readings were recorded in Tables 19-22. The tables contain both measured and calculated values. The calculated values represent the force and moment values expected to occur if the system were positioned precisely and the load cell received completely accurate data.

```
17:28:08.348 -> Command for position: (pitch,roll,yaw) Ex:(0,0,0)
17:28:08.348 -> p = 0   r = 0   y = 0
17:28:08.348 -> S1 = 84.26      S2 = 84.26      S3 = 84.26
17:28:08.348 -> m1 = 0.33       m2 = -0.37      m3 = -0.04      mx = -0.08      my = 0.11       mz = -0.29
17:28:08.348 -> R0 = [0.78,-1.08,2.94]
17:28:08.348 -> R1 = [-0.83,0.34,-3.17]
17:28:08.348 -> R2 = [-0.14,1.00,3.54]
17:28:08.348 -> R3 = [-0.08,-0.06,0.36]
17:28:08.348 -> Pforce = [0.27,-0.19,-3.68]
17:28:08.348 -> Mg = [-0.08,0.65,0.02]
17:28:08.348 -> M = [-0.08,0.65,0.02]
```

Figure 64: Measurement on serial monitor at (0,0,0). Force is applied on “z” axis and moment created on “x” axis.

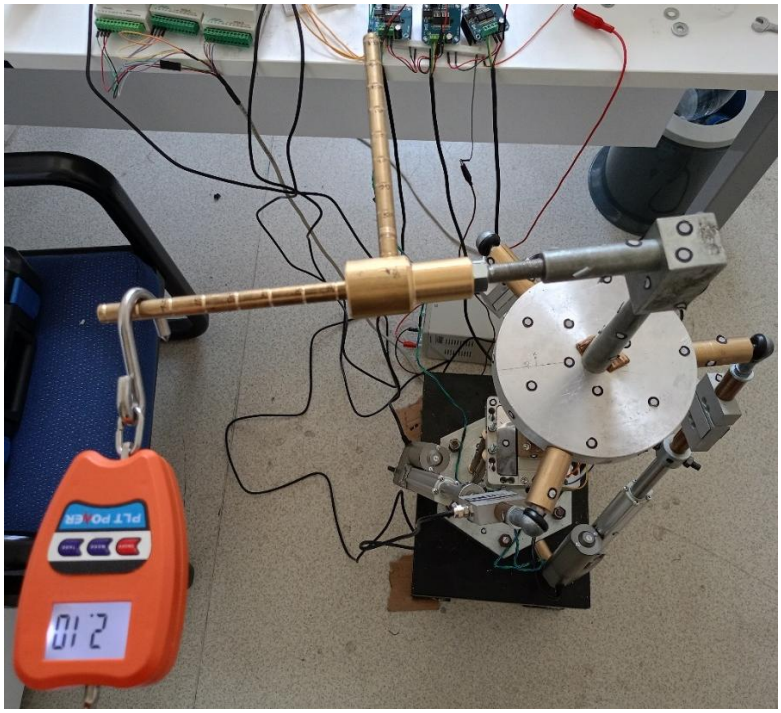

Figure 65: Measurement at (0,0,10). Force is applied on “x” axis and moment created on “z” axis.

```
16:55:28.345 -> Command for position: (pitch,roll,yaw) Ex:(0,0,0)
16:55:28.345 -> p = 0    r = 0    y = 10
16:55:28.345 -> S1 = 92.70       S2 = 92.65       S3 = 92.45
16:55:28.345 -> m1 = 5.30        m2 = 7.95        m3 = 2.30        mx = -2.44       my = -1.43       mz = -14.56
16:55:28.345 -> R0 = [24.82,14.52,152.55]
16:55:28.345 -> R1 = [-15.53,5.21,-50.37]
16:55:28.345 -> R2 = [5.06,-25.00,-78.48]
16:55:28.345 -> R3 = [5.95,5.26,-24.43]
16:55:28.345 -> Pforce = [-20.31,0.02,0.74]
16:55:28.345 -> Mg = [-0.06,-0.22,3.21]
16:55:28.345 -> M = [-0.02,-0.23,3.21]
```

Figure 66: Measurement on serial monitor at (0,0,10). Force is applied on "x" axis and moment created on "z" axis.

Tables 19, 20, 21, and 22 show the measurements taken at positions (0, 0, 0), (0, 0, 10), (0, 10, 0), and (10, 0, 0), respectively. At each of these positions, loading was performed twice in the -z direction and once each in the -x and -y directions. These loadings were always performed according to the ground coordination system, and an attempt was made to generate forces along only one axis.

Table 19: Data collection and error calculation for (p = 0, r = 0, y = 0)

x-moment/z-force						
(p,r,y) = (0,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,12	-0,13	-3,71	-0,66	-0,16	0,02
Calculated	0,00	0,00	-3,92	-0,61	0,00	0,00
Error %	-	-	5%	9%	-	-
y-moment/z-force						
(p,r,y) = (0,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,27	-0,19	-3,68	-0,08	0,65	0,02
Calculated	0,00	0,00	-3,92	0,00	0,67	0,00
Error %	-	-	6%	-	3%	-
z-moment/x-force						
(p,r,y) = (0,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	-19,27	0,47	-0,30	-0,10	-0,12	3,13
Calculated	-19,62	0,00	0,00	0,00	0,00	3,04
Error %	2%	-	-	-	-	3%
z-moment/y-force						
(p,r,y) = (0,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	-0,33	-19,73	-0,48	0,60	0,33	-3,39
Calculated	0,00	-19,62	0,00	0,00	0,00	-3,34
Error %	-	1%	-	-	-	2%

In Tables 19 and 20, only the moment generation in a single axis was calculated for all loading conditions. This is because the orientation change of the coordinate system on the top plate relative to the ground coordinate system at positions (0, 0, 0) and (0, 0, 10) is around the z axis in Table 20. Since the system is already in position of zero in Table 19, loading on individual axes in the ground coordinate system will not generate moments in more than one axis. For the zero position, loading in the z direction generates moments around the x axis when the moment arm is relative to the x axis, while it generates moments in the y axis when it is relative to the y axis. Because the force vector generated is on the xy plane for loadings applied to the x and y axes, respectively, it generates moments only in the z axis. The difference in the moment arm causes the expected moment values for the z-moment/x-force and z-moment/y-force loadings in Table 20 to differ in magnitude. The moment arms are 0.155m and 0.170m.

Table 20: Data collection and error calculation for (p = 0, r = 0, y = 10)

x-moment/z-force						
(p,r,y) = (0,0,10) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,13	-0,15	-3,81	-0,64	-0,08	0,03
Calculated	0,00	0,00	-3,92	-0,61	0,00	0,00
Error %	-	-	3%	5%	-	-
y-moment/z-force						
(p,r,y) = (0,0,10) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,16	-0,09	-3,66	-0,05	0,59	0,00
Calculated	0,00	0,00	-3,92	0,00	0,67	0,00
Error %	-	-	7%	-	12%	-
z-moment/x-force						
(p,r,y) = (0,0,10) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	-20,31	0,02	0,74	-0,02	-0,23	3,21
Calculated	-19,62	0,00	0,00	0,00	0,00	2,99
Error %	4%	-	-	-	-	7%
z-moment/y-force						
(p,r,y) = (0,0,10) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,43	-19,92	-0,82	0,35	0,14	-3,36
Calculated	0,00	-19,62	0,00	0,00	0,00	-3,29
Error %	-	2%	-	-	-	2%

Tables 20 and 21 also show two-component moment calculations. This is because, when the system rolls and pitches, an angular difference occurs between the

xy plane of the coordinate system located on the top plate and the xy plane of the ground coordinate system. No such angle is created in Tables 19 and 20, so moments are always calculated as a single-component. In Table 21, the first dual-component moment measurement is taken along with the "x&z-moment/z-force" measurement, as the system angles along the roll axis. As shown in Figure 67, the force applied to the x moment arm in the z direction of the ground coordinate system has a magnitude not only in the z direction but also in the x direction due to position. This creates a moment, albeit small compared to the moment about x, around the z axis of the shifted coordinate system.

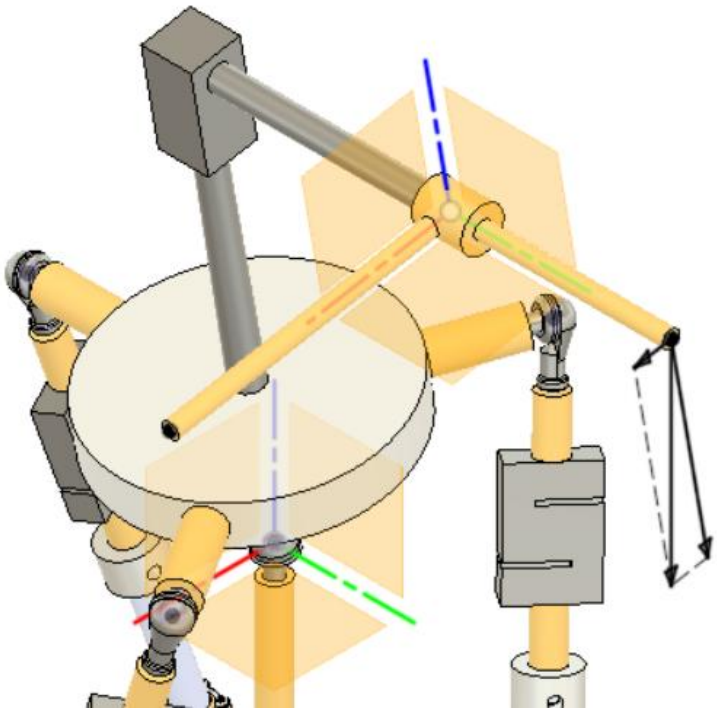

Figure 67: Force components for (0,10,0), x&z-moment/z-force condition.

Similarly, for Table 21, another two-component moment occurs for the x&z-moment/x-force condition. This time, the loading applied along the x-axis according to the ground coordinate system does not only have a magnitude along the x-axis according to the top plate coordinate system. This force also has a magnitude in the z-direction. Therefore, as seen in Figure 68, the presence of the moment arm creates a small moment around x in addition to the moment about z. A simple calculation shows that the 3.92N force in the -z direction in Figure 67 has a projection on the top plate's z-axis of 3.86N, and its projection on the top plate's x-axis is 0.68N. While the moment arm is 0.155m for both forces, when the forces and moment arms are multiplied,

moments of magnitude 0.60Nm and 0.11Nm are expected to occur on the Mx and Mz axes, respectively.

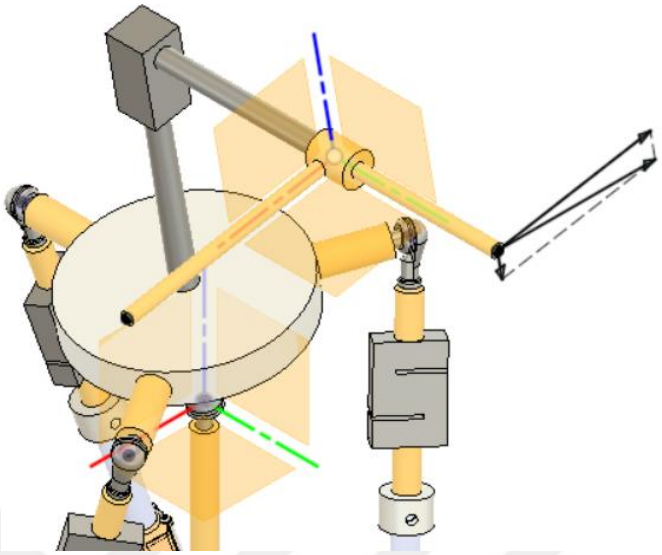

Figure 68: Force components for (0,10,0), x&z-moment/x-force condition.

Table 21: Data collection and error calculation for (p = 0, r = 10, y = 0)

x&z-moment/z-force						
(p,r,y) = (0,10,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,23	-0,10	-3,74	-0,64	-0,05	-0,12
Calculated	0,00	0,00	-3,92	-0,60	0,00	-0,11
Error %	-	-	5%	7%	-	13%
y-moment/z-force						
(p,r,y) = (0,10,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,13	-0,23	-3,84	-0,07	0,58	0,00
Calculated	0,00	0,00	-3,92	0,00	0,66	0,00
Error %	-	-	2%	-	12%	-
x&z-moment/x-force						
(p,r,y) = (0,10,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	-19,52	-0,15	-0,88	-0,57	-0,08	3,12
Calculated	-19,62	0,00	0,00	-0,53	0,00	2,99
Error %	1%	-	-	8%	-	4%
z-moment/y-force						
(p,r,y) = (0,10,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	-0,74	-19,46	-0,51	-0,38	0,82	-3,58
Calculated	0,00	-19,62	0,00	0,00	0,00	-3,29
Error %	-	1%	-	-	-	9%

If similar calculations are performed for Table 22, when a weight is hung from the end of the moment arm, which is an extension of the x-axis, this force, which is only in the z direction according to the ground coordinate system due to the pitch angle of the top plate, also has a component in the y-axis in addition to the z-axis according to the top plate coordinate system. If there were no orientation change, this moment, which would have occurred entirely on the My, generates the moment Mz, albeit in a small amount, due to the force component located in the y-axis. The same situation also occurs for Table 22 when a force is applied from the end of the moment arm, which is positioned in the x-axis in the -y-axis according to the ground coordinate system. In this case, a portion of the expected moment about the z-axis manifests itself, albeit slightly, around the y-axis. While these calculations are the correct values, actual measurements show some variation.

Table 22: Data collection and error calculation for (p = 10, r = 0, y = 0)

x-moment/z-force						
(p,r,y) = (10,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,35	-0,38	-3,73	-0,64	-0,11	0,00
Calculated	0,00	0,00	-3,92	-0,60	0,00	0,00
Error %	-	-	5%	7%	-	-
y&z-moment/z-force						
(p,r,y) = (10,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	0,32	-0,20	-4,01	-0,14	0,61	-0,14
Calculated	0,00	0,00	-3,92	0,00	0,66	-0,12
Error %	-	-	2%	-	7%	21%
z-moment/x-force						
(p,r,y) = (10,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	-20,38	-0,51	0,66	0,25	-0,68	3,09
Calculated	-19,62	0,00	0,00	0,00	0,00	3,04
Error %	4%	-	-	-	-	2%
y&z-moment/y-force						
(p,r,y) = (10,0,0) (º)	Px (N)	Py (N)	Pz (N)	Mx (Nm)	My (Nm)	Mz (Nm)
Measured	-0,91	-20,61	-0,76	0,21	-0,57	-3,42
Calculated	0,00	-19,62	0,00	0,00	-0,58	-3,29
Error %	-	5%	-	-	2%	4%

Error calculations were performed using the differences between the calculated and measured values as a result of the measurements. Tables 19-22 list these error values. Error calculations were not performed when the calculated value was zero, as dividing any number by zero renders the result undefined. Calculated errors are generally in the range of 0% to 5%. However, in some cases, even a very small difference between the calculated value and the measured value due to the small size of the calculated value can still result in a high percentage error. For example, in Table 22, in the y&z-moment/z-force condition, the calculated Mz value is -0.12, while the measured value is -0.14. Although this difference is small, it represents a percentage error of around 21%. Considering this, if the system is used for experimental purposes, data should be ignored until a certain magnitude is reached. When the data in the tables are examined, the force reading becomes significant when it exceeds 1N on any component. For moment readings, values greater than 0.5 Nm are significant. The minimum measurement capacity of the force balance system was determined by using both 0.4 kg and 2 kg weights on the mechanism. Under higher forces, the system will provide more accurate results due to the load cell capacity. It is recommended to use the system under aerodynamic forces of approximately 2 kgf and above.

3.3 CONCLUSION

This study completed the design, fabrication, and validation of a six-component Force Balance system capable of measuring force and moment, and changing model orientation with actuator control. The system is now ready for use in wind tunnels.

A comprehensive literature review was conducted at the outset of the study. As a result of this research, a unique platform-type Force Balance system design was developed. Sensors, actuators, microcontrollers, and more were optimally selected and procured based on literature research and preliminary designs. All electronic connections within the system are explained in detail, and all connection diagrams are presented. The software developed for the system's control and user interface, along with its algorithm, is comprehensively explained. Finally, verification and calibration of the actuators, load cells, and inertial measurement unit within the system were completed, making the system usable as a whole and suitable for testing. To assess measurement accuracy, various loadings were applied to the system in various orientations, and the results were compared with ideal loading values. Comparisons

have shown that the system generally has a margin of error between 0% and 5%, and that it cannot operate with the desired stability at very low loads due to its inherent nature. It has been determined that the system can deliver stable and highly accurate results under aerodynamic resultant forces of approximately 2 kgf and above.

As a future work, the system software can be accelerated, measurement sensitivities can be increased, correlations can be created that will enable the system to take measurements during movement.

REFERENCES

[1] CHAIKIN Andrew (2009), "How the Spaceship Got Its Shape", *Air & Space Magazine*, *Smithsonian Magazine*, https://www.smithsonianmag.com/air-space-magazine/how-the-spaceship-got-its-shape-137293282/, DoA. 9.2.2025.

[2] NILSSON Leif-Uno and BERNDTSSON Anders (1987), "The New Volvo Multipurpose Automotive Wind Tunnel", *SAE Technical Paper 870249*, DOI: 10.4271/870249.

[3] CERMAK Jack E (2003), "Wind-tunnel development and trends in applications to civil engineering", *Journal of Wind Engineering and Industrial Aerodynamics*, Vol. 91, No 3, pp. 355–370, DOI: 10.1016/S0167-6105(02)00396-3.

[4] BÄCKSTRÖM Mikael, CARLSSON Peter, DANVIND Jonas, KOPTYUG Andrey, SUNDSTRÖM David, and TINNSTEN Mats (2016), "A New Wind Tunnel Facility Dedicated to Sports Technology Research and Development", *Procedia Engineering*, Vol. 147, pp. 62–67, DOI: 10.1016/J.PROENG.2016.06.190.

[5] HOLSTEN J., OSTERMANN T., and MOORMANN D. (2011), "Design and wind tunnel tests of a tiltwing UAV", *CEAS Aeronautical Journal*, Vol. 2, No. 1–4, pp. 69–79, DOI: 10.1007/S13272-011-0026-4.

[6] MIKLOSOVIC David S., SNYDER Hyung S. and MURRAY R. Snyder (2011), "Ship Air Wake Wind Tunnel Test Results", *29th AIAA Applied Aerodynamics Conference*, pp. 27-30, Honolulu, DOI: 10.2514/6.2011-3155.

[7] KIOCK R., LEHTHAUS F., BAINES N. C., and SIEVERDING C. H. (1986), "The Transonic Flow Through a Plane Turbine Cascade as Measured in Four European Wind Tunnels", *Journal of engineering for gas turbines and power*, Vol. 108, No. 2, pp. 277–284, DOI: 10.1115/1.3239900.

[8] HUFNAGEL K. (2022), *Wind Tunnel Balances*, Springer Nature, Darmstadt.

[9] HOFFMANN Karl (1974), *Applying the wheatstone bridge circuit*, HBM, Darmstadt.

[10] GEYLING F. T. and FORST J. J. (1960), “Semiconductor Strain Transducers”, *Bell System Technical Journal*, Vol. 39, No. 3, pp. 705–731, DOI: 10.1002/J.1538-7305.1960.TB03939.X.

[11] AL-DAHIREE Omar Sabah, TOKHI Mohammad Osman, HADI Nabil Hassan, HMOAD Nassar Rasheid, GHAZILLA Raja Ariffin Raja, YAP Hwa Jen and ALBAADANI Emad Abdullah (2022), “Design and Shape Optimization of Strain Gauge Load Cell for Axial Force Measurement for Test Benches”, *Sensors*, Vol. 22, No. 19, p. 7508, DOI: 10.3390/S22197508.

[12] WICAKSONO Satrio, ARIANA G. P., FERRYANTO F., WIBOWO A. and ISRA Mahyuddin Andi (2019), “Design optimization, manufacturing, and testing of affordable 3-axis load cell for reliable force plate component”, *In, INNOVATIVE SCIENCE AND TECHNOLOGY IN MECHANICAL ENGINEERING FOR INDUSTRY 4.0: Proceedings of the 4th International Conference on Mechanical Engineering*, Vol. 2187, Issue 1, pp. 28-29, AIP Publishing, NY, DOI: 10.1063/1.5138338.

[13] EWALD B., and KRENZ G. (1986), “The accuracy problem of airplane development force testing in cryogenic wind tunnels”, *14th Aerodynamic Testing Conference*, pp. 405-415, Florida, DOI: 10.2514/6.1986-776.

[14] DASGUPTA Bhaskar and T. S. MRUTHYUNJAYA (2000), “The Stewart platform manipulator: a review”, *Mechanism and Machine Theory*, Vol. 35, No. 1, pp. 15–40, DOI: 10.1016/S0094-114X(99)00006-3.

[15] FURQAN Mohd, SUHAIB Mohd, and AHMAD Nazeer (2017), “Studies on Stewart platform manipulator: A review”, *Journal of Mechanical Science and Technology*, Vol. 31, No. 9, pp. 4459–4470, DOI: 10.1007/s12206-017-0846-1.

[16] THOMAS M. and TESAR D. (1982), “Dynamic Modeling of Serial Manipulator Arms”, *Journal of Dynamic Systems, Measurement, and Control*, Vol. 104, No. 3, pp. 218–228, DOI: 10.1115/1.3139701.

[17] ULAŞ Burak (2009), *Stewart platformu tasarımı* (Master’s Thesis), İstanbul Technical University of Natural and Applied Sciences, İstanbul.

[18] UĞUROĞLU Enes (2015), *Serbestlik dereceli rotasyonel stewart platformu tasarımı ve yüzey pürüzlülük ölçümlerinde eğim sensörü ile konum kontrolü* (Master’s Thesis), İstanbul Technical University of Natural and Applied Sciences, İstanbul.

[19] PRADIPTA Justin, KLÜNDER Mario, WEICKGENANNT Martin and SAWODNY Oliver (2013), "Development of a pneumatically driven flight simulator Stewart platform using motion and force control", *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 158-163, Wollongong, DOI: 10.1109/AIM.2013.6584085.

[20] ARCONADA Verónica Santos, GARCÍA-BARRUETABEÑA Jon and HAAS Rainer (2022), "Validation of a ride comfort simulation strategy on an electric Stewart Platform for real road driving applications", *Journal of Low Frequency Noise, Vibration and Active Control*, Vol. 42, No. 1, pp. 368-391, DOI: 10.1177/14613484221122109.

[21] TYTUŁA Jakub, ZAREMBA A., and NITKIEWICZ S. (2023), "Evaluation of forces in a racing simulator based on a Stewart platform", *Diagnostyka*, Vol. 24, No. 4, pp. 1–8, DOI: 10.29354/diag/171729.

[22] Alkhedher MOHAMMAD, Younes TARIQ, Mohamad OMAR and Ali UZAIR (2020), "Adaptive 6 DOF Self-Balancing Platform for Autonomous Vehicles", *International Journal of Computing and Digital Systems*, Vol. 9, No. 1, pp. 69–75, DOI: 10.12785/IJCDS/090107.

[23] TING Yung, CHEN Yu-Shin, and JAR Ho-Chin (2004), "Modeling and control for a Gough-Stewart platform CNC machine", *Journal of Robotic Systems*, Vol. 21, No. 11, pp. 609–623, DOI: 10.1002/ROB.20039.

[24] HARIB K. and SRINIVASAN K. (2003), "Kinematic and dynamic analysis of Stewart platform-based machine tool structures", *Robotica*, Vol. 21, No. 5, pp. 541–554, DOI: 10.1017/S0263574703005046.

[25] WAPLER Matthias, URBAN Volker, WEISENER Thomas, STALLKAMP Jan, DÜRR Mark and HILLER Andrea (2003), "A Stewart platform for precision surgery", *Transactions of the Institute of Measurement and Control*, Vol. 25, No. 4, pp. 329–334, DOI: 10.1191/0142331203TM092OA.

[26] GIRONE M., BURDEA G., BOUZIT M., POPESCU V. and DEUTSCH J. E. (2001), "A Stewart Platform-Based System for Ankle Telerehabilitation", *Autonomous Robots*, Vol. 10, No. 2, pp. 203–212, DOI: 10.1023/A:1008938121020.

[27] MU Zongliang and KAZEROUNIAN Kazem (2002), “A Real Parameter Continuation Method for Complete Solution of Forward Position Analysis of the General Stewart”, *Journal of Mechanical Design*, Vol. 124, No. 2, pp. 236–244, DOI: 10.1115/1.1446476.

[28] DASGUPTA Bhaskar and MRUTHYUNJAYA T. S. (1998), “A Newton-Euler formulation for the inverse dynamics of the Stewart platform manipulator”, *Mechanism and Machine Theory*, Vol. 33, No. 8, pp. 1135–1152, DOI: 10.1016/S0094-114X(97)00118-3.

[29] KHALIL Wisama and IBRAHIM Ouarda (2007), “General Solution for the Dynamic Modeling of Parallel Robots”, *Journal of Intelligent & Robotic Systems*, Vol. 49, No. 1, pp. 19–37, DOI: 10.1007/S10846-007-9137-X.

[30] RIEBE S. and ULBRICH H. (2003), “Modelling and online computation of the dynamics of a parallel kinematic with six degrees-of-freedom”, *Archive of Applied Mechanics*, Vol. 72, No. 11, pp. 817–829, DOI: 10.1007/S00419-002-0262-5.

[31] SEFRIOUI Jaouad and GOSSELIN Clément M. (1992), “Singularity analysis and representation of planar parallel manipulators”, *Robotics and Autonomous Systems*, Vol. 10, No. 4, pp. 209–224, DOI: 10.1016/0921-8890(92)90001-F.

[32] SEFRIOUI Jaouad and GOSSELIN Clément M. (1995), “On the quadratic nature of the singularity curves of planar three-degree-of-freedom parallel manipulators”, *Mechanism and Machine Theory*, Vol. 30, No. 4, pp. 533–551, DOI: 10.1016/0094-114X(94)00052-M.

[33] ZHANG Tianli, JIANG Chengbao, ZHANG Hu and XU Huibin (2004), “Giant magnetostrictive actuators for active vibration control”, *Smart Materials and Structures*, Vol. 13, No. 3, pp. 473–477, DOI: 10.1088/0964-1726/13/3/004.

[34] HAUGE G. S. and. CAMPBELL Mark E. (2003)., “Sensors and control of a space-based six-axis vibration isolation system”, *Journal of Sound and Vibration*, Vol. 269, No. 3–5, pp. 913–931, DOI: 10.1016/S0022-460X(03)00206-2.

[35] WANG Shao-Chi, HIKITA Hiromitsu, KUBO Hiroshi, ZHAO Yongsheng, HUANG Zheng and IFUKUBE Tohru (2003), “Kinematics and dynamics of a 6 degree-of-freedom fully parallel manipulator with elastic joints”, *Mechanism and Machine Theory*, Vol. 38, No. 5, pp. 439–461, DOI: 10.1016/S0094-114X(02)00132-5.

APPENDICES

APPENDIX 1: COMPLETE CODE

Global Settings

Code Block 1

```
// ======                    GLOBAL SETTINGS                    ======
#include <BasicLinearAlgebra.h>
#include <EEPROM.h>
#include "HX711.h"
#include <SPI.h>
#include <SD.h>
#include "BTS7960.h"
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
#define OUTPUT_READABLE_YAWPITCHROLL
#define INTERRUPT_PIN 2
MPU6050 mpu;
using namespace BLA;
float buffer[6];
```

Code Block 2

```
// ================================================================
// ===                  INITIAL MATRIX CONSTANTS                ===
// ================================================================

  BLA::Matrix<1, 3> Ag = {-159.349, -92.00, -511.00};
  BLA::Matrix<1, 3> Bg = {159.349, -92.00, -511.00};
  BLA::Matrix<1, 3> Cg = {0.00, 184.00, -511.00};

  BLA::Matrix<1, 3> A = {0.00, -157.00, 94.5};
  BLA::Matrix<1, 3> B = {135.966, 78.50, 94.5};
  BLA::Matrix<1, 3> C = {-135.966, 78.50, 94.5};
  BLA::Matrix<1, 3> C_g = {0.00, 167.5, 324};

  BLA::Matrix<3, 1> r1 = ~A;
  BLA::Matrix<3, 1> r2 = ~B;
  BLA::Matrix<3, 1> r3 = ~C;
  BLA::Matrix<3, 1> rcg = ~C_g;

// ================================================================
```

Code Block 3

```
// ================================================================
// ===                      IMU GLOBAL CONSTANTS                  ===
// ================================================================

// Data print speed parameters
unsigned long lastPrintTime = 0;
const unsigned long printInterval = 150;

//Other IMU necessery parameters
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from
MPU
uint8_t devStatus;      // return status after each device
operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default 42
bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
uint16_t drift_comp = 0;
uint16_t last_update = 0;
uint16_t drift_offset = 0;

// Orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion
container
VectorInt16 aa;         // [x, y, z]            accel sensor
measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi] Euler angle container
float ypr[3];           // [yaw, pitch, roll] container
float last_yaw = 0;
float last_pitch = 0;
float     last_roll = 0;
int count = 0;
// Packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0,
0x00, 0x00, '\r', '\n' };
// ================================================================
```

Code Block 4

```
// ================================================================
// ===             LOADCELL READING GLOBAL SETTINGS             ===
// ================================================================
HX711 scales[3];
float calibration_factors[3];
long tare_values[3];
float m_values[3];
float mx = 0.0, my = 0.0, mz = 0.0;
int zeroCountX = 0, zeroCountY = 0, zeroCountZ = 0;
const int dtPins[3] = {3, 4, 5};
#define SCK_PIN 6
#define ZERO_THRESHOLD_COUNT 5
// ================================================================
```

Code Block 5

```
// ================================================================
// ===         ACTUATOR RESISTENCE MEASUREMENT SETTINGS         ===
// ================================================================

const int sensorpin1 = A0;
const int sensorpin2 = A1;
const int sensorpin3 = A2;
const float Rref = 4.6;
const float Vin = 3.3;

// ================================================================
```

Code Block 6

```
// ================================================================
// ===                    MOTOR CONTROLLER SETTINGS                ===
// ================================================================

float Stroke1 = 0, Stroke2 = 0, Stroke3 = 0;
float targetStroke1 = -1, targetStroke2 = -1, targetStroke3 = -1;
float startStroke1 = 0, startStroke2 = 0, startStroke3 = 0;

const uint8_t EN = 7;
const uint8_t L_PWM1 = 8, R_PWM1 = 9;
const uint8_t L_PWM2 = 10, R_PWM2 = 11;
const uint8_t L_PWM3 = 12, R_PWM3 = 13;

BTS7960 motor1(EN, L_PWM1, R_PWM1);
BTS7960 motor2(EN, L_PWM2, R_PWM2);
BTS7960 motor3(EN, L_PWM3, R_PWM3);

const int maxPWM = 150;
const int minPWM = 30;

// ================================================================
```

Code Block 7

```
// ================================================================
// ===                    SD CARD GLOBAL SETTINGS                  ===
// ================================================================

const int chipSelect = 53;
File binFile;

// ================================================================

// Serial input initial definition
String inputString = "";
bool inputReady = false;

// RS232 input parsing definition
String bufferX = "", bufferY = "", bufferZ = "";

// ================================================================
// ================================================================
```

Setup

Code Block 8

```
// ================================================================
// ================================================================
// ======                          SETUP                         ======
// ================================================================
// ================================================================

void setup() {

  Serial.begin(500000);  // USB communication baudrate
  Serial1.begin(115200); // X axis indicator baudrate
  Serial2.begin(115200); // Y axis indicator baudrate
  Serial3.begin(115200); // Z axis indicator baudrate
```

Code Block 9

```
  // ================================================================
  // ===    S-TYPE LOADCELLS' CALIBRATION FACTOR FROM EEPROM    ===
  // ================================================================

  for (int i = 0; i < 3; i++) {
    scales[i].begin(dtPins[i], SCK_PIN);
    EEPROM.get(i * sizeof(float), calibration_factors[i]);
    if (isnan(calibration_factors[i]) || calibration_factors[i] ==
0.0) calibration_factors[i] = 1.0;
    tare_values[i] = scales[i].read_average(30);
  }
  // ================================================================
```

Code Block 10

```
  // ================================================================
  // ===                  SD CARD INFORMATION                   ===
  // ================================================================

  Serial.println("SD Card is being searched...");
  if (!SD.begin(chipSelect)) Serial.println("❌ SD Card could not
be found!");
  else Serial.println("☑ SD Card is ready.");

  // ================================================================
```

Code Block 11

```
  // ================================================================
  // ===               INTERRUPT DETECTION ROUTINE2                 ===
  // ================================================================

  // join I2C bus (I2Cdev library doesn't do this automatically)
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000); // 400kHz I2C clock. Comment this line
if having compilation difficulties
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif
  while (!Serial);
  // initialize device
  Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
  pinMode(INTERRUPT_PIN, INPUT);
  // verify connection
  Serial.println(F("Testing device connections..."));
  Serial.println(mpu.testConnection() ? F("MPU6050 connection
successful") : F("MPU6050 connection failed"));
  // wait for ready
  Serial.println(F("\nSend any character to begin DMP programming
and demo: "));
  while (Serial.available() && Serial.read()); // empty buffer
  while (!Serial.available());                 // wait for data
  while (Serial.available() && Serial.read()); // empty buffer
again
  // load and configure the DMP
  Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();
  // supply your own gyro offsets here, scaled for min sensitivity
  mpu.setXGyroOffset(220);
  mpu.setYGyroOffset(76);
  mpu.setZGyroOffset(0);
  mpu.setZAccelOffset(1688); // 1688 factory default for my test
chip
```

Code Block 12

```
  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setRate(39);  // 19 -- 50 Hz yapmak için, 39 -- 25 Hz
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0)..."));
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN),
dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows
it's okay to use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }

  // ===============================================================

}
```

Loop

Code Block 13

```
void loop() {

  // ================================================================
  // ===                      IMU LOOP                              ===
  // ================================================================

  // if programming failed, don't try to do anything
  if (!dmpReady) return;

  // wait for MPU interrupt or extra packet(s) available
  while (!mpuInterrupt && fifoCount < packetSize) {}

  // reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

  // get current FIFO count
  fifoCount = mpu.getFIFOCount();

  // check for overflow
  if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    //Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should
happen frequently)
  } else if (mpuIntStatus & 0x02) {
    // wait for correct available data length, should be a VERY
short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an
interrupt)
    fifoCount -= packetSize;
    #ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
```

Code Block 14

```
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    if (count < 2000) {

      last_yaw = ypr[0];
      last_pitch = ypr[1];
      last_roll = ypr[2];
      count ++;

      String data1 = "Wait! IMU Calibration\t" +
                     String(count) + "\t" +
                     String((ypr[0]) * 180 / M_PI) + "\t" +
                     String((ypr[1]) * 180 / M_PI) + "\t" +
                     String((ypr[2]) * 180 / M_PI);

      Serial.println(data1);

    }

    #endif

  }

  //
=================================================================
```

Code Block 15

```
  // ============================================================
  // ===                        MAIN LOOP                     ===
  // ============================================================

  if (count >= 2000) {

    int p = (ypr[2] - last_pitch) * 180 / M_PI;
    int r = -(ypr[1] - last_roll) * 180 / M_PI;
    int y = -(ypr[0] - last_yaw) * 180 / M_PI;

    float L1_full = Stroke1 + 544.5;
    float L2_full = Stroke2 + 544.5;
    float L3_full = Stroke3 + 544.5;

    float R1mag = m_values[0] * 9.81;
    float R2mag = m_values[1] * 9.81;
    float R3mag = m_values[2] * 9.81;

    BLA::Matrix<3, 3> T1 = {1, 0, 0, 0, cosd(p), sind(p), 0, -
sind(p), cosd(p)};
    BLA::Matrix<3, 3> T2 = {cosd(r), 0, -sind(r), 0, 1, 0, sind(r),
0, cosd(r)};
    BLA::Matrix<3, 3> T3 = {cosd(y), sind(y), 0, -sind(y), cosd(y),
0, 0, 0, 1};

    BLA::Matrix<3, 3> T = T3*T2*T1;
    BLA::Matrix<3, 3> T_T = ~T;

    BLA::Matrix<3, 1> r1prime = T_T*r1;
    BLA::Matrix<3, 1> r2prime = T_T*r2;
    BLA::Matrix<3, 1> r3prime = T_T*r3;
    BLA::Matrix<3, 1> rcgprime = T_T*rcg;

    BLA::Matrix<1, 3> Aprime = ~r1prime;
    BLA::Matrix<1, 3> Bprime = ~r2prime;
    BLA::Matrix<1, 3> Cprime = ~r3prime;
    BLA::Matrix<1, 3> C_gprime = ~rcgprime;
```

Code Block 16

```
    BLA::Matrix<3, 1> R0 = {-mx * 9.81, -my * 9.81, -mz * 9.81};
    BLA::Matrix<3, 1> R1 = ~(-(Aprime-Ag)/L1_full*R1mag);
    BLA::Matrix<3, 1> R2 = ~(-(Bprime-Bg)/L2_full*R2mag);
    BLA::Matrix<3, 1> R3 = ~(-(Cprime-Cg)/L3_full*R3mag);

    BLA::Matrix<3, 1> Pforce = -(R0 + R1 + R2 + R3);

    float Pforcecross[3]; Pforcecross[0] = Pforce(0,0);
Pforcecross[1] = Pforce(1,0); Pforcecross[2] = Pforce(2,0);
    float R0cross[3]; R0cross[0] = R0(0,0); R0cross[1] = R0(1,0);
R0cross[2] = R0(2,0);
    float R1cross[3]; R1cross[0] = R1(0,0); R1cross[1] = R1(1,0);
R1cross[2] = R1(2,0);
    float R2cross[3]; R2cross[0] = R2(0,0); R2cross[1] = R2(1,0);
R2cross[2] = R2(2,0);
    float R3cross[3]; R3cross[0] = R3(0,0); R3cross[1] = R3(1,0);
R3cross[2] = R3(2,0);

    float rcgprimecross[3]; rcgprimecross[0] = rcgprime(0,0);
rcgprimecross[1] = rcgprime(1,0); rcgprimecross[2] = rcgprime(2,0);
    float r1primecross[3]; r1primecross[0] = r1prime(0,0);
r1primecross[1] = r1prime(1,0); r1primecross[2] = r1prime(2,0);
    float r2primecross[3]; r2primecross[0] = r2prime(0,0);
r2primecross[1] = r2prime(1,0); r2primecross[2] = r2prime(2,0);
    float r3primecross[3]; r3primecross[0] = r3prime(0,0);
r3primecross[1] = r3prime(1,0); r3primecross[2] = r3prime(2,0);

    float cross_rcg_Pforce[3];
    float cross_r1_R1[3];
    float cross_r2_R2[3];
    float cross_r3_R3[3];

    crossProduct(rcgprimecross, Pforcecross, cross_rcg_Pforce);
    crossProduct(r1primecross, R1cross, cross_r1_R1);
    crossProduct(r2primecross, R2cross, cross_r2_R2);
    crossProduct(r3primecross, R3cross, cross_r3_R3);
```

Code Block 17

```
    float Mg[3];
    Mg[0] = -(cross_rcg_Pforce[0] + cross_r1_R1[0] + cross_r2_R2[0]
+ cross_r3_R3[0])/1000;
    Mg[1] = -(cross_rcg_Pforce[1] + cross_r1_R1[1] + cross_r2_R2[1]
+ cross_r3_R3[1])/1000;
    Mg[2] = -(cross_rcg_Pforce[2] + cross_r1_R1[2] + cross_r2_R2[2]
+ cross_r3_R3[2])/1000;

    BLA::Matrix<3, 1> MgBLA = {Mg[0], Mg[1], Mg[2]};
    BLA::Matrix<3, 1> M = T_T * MgBLA;
```

Code Block 18

```
    updateStrokes();
    controlactuator(motor1, Stroke1, startStroke1, targetStroke1,
"L1");
    controlactuator(motor2, Stroke2, startStroke2, targetStroke2,
"L2");
    controlactuator(motor3, Stroke3, startStroke3, targetStroke3,
"L3");

    static int pitch = 0, roll = 0, yaw = 0;

    while (Serial.available()) {
      char c = Serial.read();
      if (c == '\n' || c == '\r') inputReady = true;
      else inputString += c;
    }
```

Code Block 19

```
    if (inputReady) {
      inputReady = false;
      inputString.trim();

      // Checking for calibration and tare commands
      if (inputString == "c") {
        for (int i = 0; i < 3; i++) handleCalibration(i);
      }

      if (inputString == "t") {
        for (int i = 0; i < 3; i++) handleTare(i);
      }

      if (inputString == "c1") {
        handleCalibration(0);
      }

      if (inputString == "c2") {
        handleCalibration(1);
      }

      if (inputString == "c3") {
        handleCalibration(2);
      }

      if (inputString == "t1") {
        handleTare(0);
      }

      if (inputString == "t2") {
        handleTare(1);
      }

      if (inputString == "t3") {
        handleTare(2);
      }
```

Code Block 20

```
      else {

        int i1 = inputString.indexOf(',');
        int i2 = inputString.indexOf(',', i1 + 1);
        if (i1 > 0 && i2 > i1) {
          pitch = inputString.substring(0, i1).toInt();
          roll  = inputString.substring(i1 + 1, i2).toInt();
          yaw   = inputString.substring(i2 + 1).toInt();

          if (pitch < -20 || pitch > 20 || roll < -20 || roll > 20
|| yaw < -20 || yaw > 20) {
            Serial.println("⚠ Values must be between -20 and
20.");
          }
```

Code Block 21

```
          else {
            long index = (pitch + 20) * 1681L + (roll + 20) * 41L +
(yaw + 20);
            long bytePos = index * 108L;

            binFile = SD.open("stroke1.bin");
            if (binFile) {
              binFile.seek(bytePos);
              binFile.read((uint8_t*)buffer, sizeof(buffer));
              binFile.close();

              targetStroke1 = max(buffer[3], 0.0f);
              targetStroke2 = max(buffer[4], 0.0f);
              targetStroke3 = max(buffer[5], 0.0f);

              updateStrokes();
              startStroke1 = Stroke1; startStroke2 = Stroke2;
startStroke3 = Stroke3;

              Serial.println("🎯 Target Strokes:");
              Serial.print("L1: "); Serial.println(targetStroke1);
              Serial.print("L2: "); Serial.println(targetStroke2);
              Serial.print("L3: "); Serial.println(targetStroke3);

            } else {
              Serial.println("❌ stroke1.bin could not open.");
            }
          }
        }
      }

      inputString = ""; //Reset old command

    }
```

Code Block 22

```
    for (int i = 0; i < 3; i++) {
      long reading = scales[i].read_average(2);
      float net = reading - tare_values[i];
      m_values[i] = net / calibration_factors[i];
    }

    readRS232(); // RS232 data reading
```

Code Block 23

```
    unsigned long currentMillis = millis();

    if (currentMillis - lastPrintTime >= printInterval) {

      lastPrintTime = currentMillis; // Update last print time

      String data = "Command for position: (pitch,roll,yaw)
Ex:(0,0,0)\n"
                    "p = " + String(p) + "\t" +
                    "r = " + String(r) + "\t" +
                    "y = " + String(y) + "\n" +
                    "S1 = " + String(Stroke1) + "\t" +
                    "S2 = " + String(Stroke1) + "\t" +
                    "S3 = " + String(Stroke1) + "\n" +
                    "m1 = " + String(m_values[0]) + "\t" +
                    "m2 = " + String(m_values[1]) + "\t" +
                    "m3 = " + String(m_values[2]) + "\t" +
                    "mx = " + String(mx) + "\t" +
                    "my = " + String(my) + "\t" +
                    "mz = " + String(mz) + "\n" +
                    "R0 = [" + String(R0cross[0]) + "," +
String(R0cross[1]) + "," + String(R0cross[2]) + "]\n" +
                    "R1 = [" + String(R1cross[0]) + "," +
String(R1cross[1]) + "," + String(R1cross[2]) + "]\n" +
                    "R2 = [" + String(R2cross[0]) + "," +
String(R2cross[1]) + "," + String(R2cross[2]) + "]\n" +
                    "R3 = [" + String(R3cross[0]) + "," +
String(R3cross[1]) + "," + String(R3cross[2]) + "]\n" +
                    "Pforce = [" + String(Pforcecross[0]) + "," +
String(Pforcecross[1]) + "," + String(Pforcecross[2]) + "]\n" +
                    "Mg = [" + String(Mg[0]) + "," + String(Mg[1])
+ "," + String(Mg[2]) + "]\n" +
                    "M = [" + String(M(0,0)) + "," + String(M(1,0))
+ "," + String(M(2,0)) + "]\n"
                    ;
      Serial.println(data); // Main print

    }
  }
  // ============================================================
}
```

Software Functions

Code Block 24

```
void readRS232() {
  while (Serial1.available()) {
    char c = Serial1.read();
    if (c == '\n') {
      float parsed = parseWeight(bufferX);
      if (!isnan(parsed)) {
        if (parsed == 0.0) zeroCountX++; else { mx = parsed;
zeroCountX = 0; }
        if (zeroCountX >= ZERO_THRESHOLD_COUNT) mx = 0.0;
      }
      bufferX = "";
    } else {
      bufferX += c;
    }
  }

  while (Serial2.available()) {
    char c = Serial2.read();
    if (c == '\n') {
      float parsed = parseWeight(bufferY);
      if (!isnan(parsed)) {
        if (parsed == 0.0) zeroCountY++; else { my = parsed;
zeroCountY = 0; }
        if (zeroCountY >= ZERO_THRESHOLD_COUNT) my = 0.0;
      }
      bufferY = "";
    } else {
      bufferY += c;
    }
  }
```

Code Blocks 24 and 25 show how the readRS232 function works. This function reads data from three separate serial ports. For these reads, the while loops must be active, which is achieved when the relevant serial communication is active. Three serial buffers have been defined. These buffers are called buffer, buffer, and bufferZ, and are used to store data received via Serial 1, Serial 2, and Serial 3, respectively. The while loops will continue to run as long as there is data in these buffers. Each character coming from the communication line is read one by one. If the incoming character is "\n", the data to be allocated to the relevant buffer is complete, and the relevant buffer is defined. Following this, the parseWeight function converts the string data to

numerical data. False "0" values may appear in the data stream. To eliminate these, zeroCount counters are implemented. This counter checks the number of consecutive "0" values, and only after a certain number of "0" values are received is the actual value considered "0." The number of times a "0" is returned can be determined using the "ZERO_THRESHOLD_COUNT" variable, whose value is set in the global settings. In the current case, a "0" return five times is considered a 0 value.

Code Block 25

```
  while (Serial3.available()) {
    char c = Serial3.read();
    if (c == '\n') {
      float parsed = parseWeight(bufferZ);
      if (!isnan(parsed)) {
        if (parsed == 0.0) zeroCountZ++; else { mz = parsed;
zeroCountZ = 0; }
        if (zeroCountZ >= ZERO_THRESHOLD_COUNT) mz = 0.0;
      }
      bufferZ = "";
    } else {
      bufferZ += c;
    }
  }
}

float parseWeight(String line) {
  int idxStart = line.indexOf('+');
  if (idxStart == -1) idxStart = line.indexOf('-');
  int idxEnd = line.indexOf('k');
  if (idxStart != -1 && idxEnd != -1 && idxEnd > idxStart) {
    String numStr = line.substring(idxStart, idxEnd);
    return numStr.toFloat();
  }
  return NAN;
}
```

The updateStrokes function in Code Block 26 is the function that allows stroke values to be read. The voltage values read from the analog pins are then read as resistance using a reference resistor after the operations are performed. These resistance values are used as inputs to the polynomials generated as a result of calibration for use in stroke calculations, and the stroke value is output as output. This

allows continuous stroke values to be read and incorporated into necessary calculations.

Code Block 26

```
void updateStrokes() {
  int sv1 = analogRead(sensorpin1);
  int sv2 = analogRead(sensorpin2);
  int sv3 = analogRead(sensorpin3);

  float corrected1 = sv1 * (5.0 / 3.3);
  float corrected2 = sv2 * (5.0 / 3.3);
  float corrected3 = sv3 * (5.0 / 3.3);

  float r1 = Rref * corrected1 / (1023.0 - corrected1);
  float r2 = Rref * corrected2 / (1023.0 - corrected2);
  float r3 = Rref * corrected3 / (1023.0 - corrected3);

  Stroke1 = 0.0118810809 * r1 * r1 * r1 * r1 - 0.151403074 * r1 *
r1 * r1 + 0.630064595 * r1 * r1 + 20.3074838 * r1 - 13.3042067;
  Stroke2 = 0.0029754 * r2 * r2 * r2 * r2 - 0.0189902 * r2 * r2 *
r2 + 0.0290367 * r2 * r2 + 20.7764 * r2 - 12.9290;
  Stroke3 = 0.01744375 * r3 * r3 * r3 * r3 - 0.2875559 * r3 * r3 *
r3 + 1.66602424 * r3 * r3 + 17.09392262 * r3 - 9.84465467;
  //Stroke1 = 20.238 * r1 - 8.905;
  //Stroke2 = 20.238 * r2 - 8.905;
  //Stroke3 = 20.238 * r3 - 8.905;
}
```

Code Block 27 defines the function that allows actuator motors to be moved to the taglet position using the BTS7960 motor driver cards. This function aims to simultaneously initiate and complete the movements of three separate actuators. While this PWM control process cannot be achieved precisely in the real world, it does provide a similar effect. Using stroke readings, the distance between the initial and target stroke values is calculated, and each actuator's speed is controlled proportionally to this distance. A feedback-controlled algorithm is created by continuously updating the present variable with the current stroke length. As the movement progresses, the stroke value is continuously checked, and the speed values are updated accordingly.

Code Block 27

```
void controlactuator(BTS7960& motor, float present, float initial,
float target, String name) {
  if (target < 0) return;
  float tolerance = (target > 140) ? 0.5 : 0.5;
  float initialdistance = abs(initial - target);
  float leftdistance = abs(present - target);

  motor.Enable();
  if (leftdistance > tolerance) {
    int pwm = map(leftdistance * 1000, 0, initialdistance * 1000,
minPWM, maxPWM);
    pwm = constrain(pwm, minPWM, maxPWM);
    if (present < target) motor.TurnLeft(pwm);
    else motor.TurnRight(pwm);
  } else {
    motor.Stop();
    Serial.print(name); Serial.println(" target arrived.");
    if (name == "L1") targetStroke1 = -1;
    if (name == "L2") targetStroke2 = -1;
    if (name == "L3") targetStroke3 = -1;
  }
}
```

As its name suggests, Code Block 28 is a function that performs the cross-product linear algebra operation on two three-component arrays. Since such a function doesn't exist directly in C++, it needs to be defined simply. It embeds the result of the operation into a three-component result array.

Code Block 28

```
void crossProduct(float a[3], float b[3], float result[3]) {
  result[0] = a[1] * b[2] - a[2] * b[1];
  result[1] = a[2] * b[0] - a[0] * b[2];
  result[2] = a[0] * b[1] - a[1] * b[0];
}
```

Code Block 29

```
// Loadcell Calibration Function
void handleCalibration(int index) {
  Serial.print("\n📏 m"); Serial.print(index + 1); Serial.println("
is calibrating.");
  Serial.println("1) Press ENTER without load:");
  while (Serial.available() == 0);
  Serial.readStringUntil('\n');
  tare_values[index] = scales[index].read_average(30);
  Serial.print("Tare value: "); Serial.println(tare_values[index]);

  Serial.println("2) Place known weight and write its value as
kg:");
  while (Serial.available() == 0);
  float knownWeight = Serial.readStringUntil('\n').toFloat();

  long loaded = scales[index].read_average(30);
  long net = loaded - tare_values[index];
  if (knownWeight <= 0 || net == 0) {
    Serial.println("❗ Incorrect calibration data.");
    return;
  }

  calibration_factors[index] = net / knownWeight;
  EEPROM.put(index * sizeof(float), calibration_factors[index]);
  Serial.print("☑ New calibration factor (m");
  Serial.print(index + 1); Serial.print("): ");
  Serial.println(calibration_factors[index], 4);
}
```

Code Block 29 defines the function that allows calibration of S-Type load cells. This function lets the user calibrate a specific load cell step by step. The index parameter tells the function which load cell (0, 1, or 2) to calibrate. The serial monitor first shows the user an information message, and then the first step of the calibration begins. The user is asked to press ENTER if the sensor is empty. If there isn't any data coming in right now, the line while (Serial.available() == 0); will wait. The read_average(30) command takes 30 samples from the load cell, finds the average, and stores it in the tare_values[index] variable when the user sends data. The system will use this value as the starting point. The user is then asked to put a known weight on the load cell and type in its weight in kilograms. The toFloat() function turns the value that the user entered into a number. At this point, the load cell is read again to find out

what the new load state is and to figure out the net value (loaded - empty). If this net value is zero or the user enters an invalid mass, the process stops and an error occurs. The net/knownWeight formula is used to figure out the calibration coefficient if there is valid data. Writing to EEPROM makes this coefficient permanent, and it can be used to directly convert the ADC reading to mass in kilograms. Finally, the user gets a message through the serial port that the calibration worked and the new calibration coefficient. This structure makes it easy and reliable to recalibrate the system after replacing sensors or making physical changes in the field.

Code Block 30

```
// Tare Function
void handleTare(int index) {
  tare_values[index] = scales[index].read_average(30);
  Serial.print("🔄 m"); Serial.print(index + 1); Serial.println("
is tared.");
}
```

Lastly, the handleTare function in Code Block 30 does the tare operation on a specific load cell. It needs an index value as an argument to know which sensor to zero (for example, 0 → m1, 1 → m2, and 2 → m3). The read_average(30) command tells the function to take 30 samples from the HX711 module that was chosen. It then puts the average of these samples into the tare_values[index] array. This number tells you what the ADC output is when the load cell isn't full. We get the net weight by taking this number away from the next load measurements. The serial port sends a message to the user saying that the sensor has been set to zero. This step makes sure that the system sees the sensor's weight as "zero." This lets it correctly measure loads even when they are changing. It comes in handy for experiments or when you can't actually set something to zero.

APPENDIX 2: ELECTRONICAL COMPONENTS USED

WEILO 3-Axes Loadcell

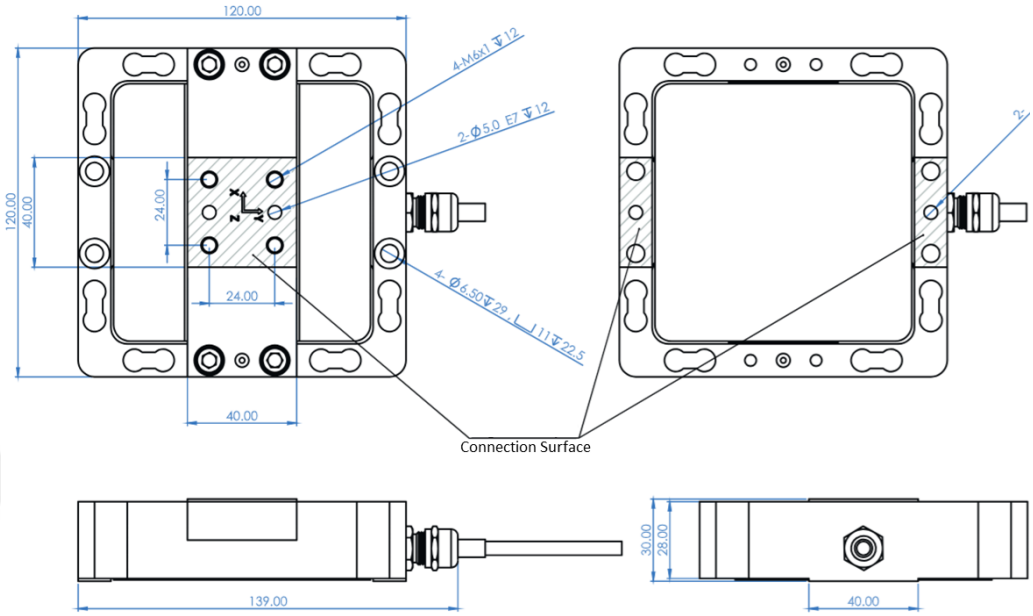

Figure 69: Weilo 3-axis loadcell

Table 23: 3 axis loadcell specifications

Weilo 3 Axis Loadcell	
Specification	**Value**
Capacity	1kN
Accuracy Class	0.5%
Minimum Load	0 kg
Total Error	±0.2 %FS
Output Signal (FS)	1.00 ± 0.01 mV/V
Zero Balance	±1 %FS
Input Resistance	740 Ω ± 10 Ω
Output Resistance	700 Ω ± 5 Ω
Insulation Resistance	≤ 5000 MΩ (100VDC)
Compensated Temperature Range	-10 to +40 °C
Operating Temperature Range	-30 to +70 °C
Recommended Excitation Voltage	10 VDC
Maximum Excitation Voltage (Umax)	15 VDC
Safe Overload	120 %FS
Ultimate Overload	200 %FS
Protection Class (EN60529)	IP 65
Material	Stainless Steel / Aluminum
Cable	12x0.22 mm²

WEILO S-Type Loadcell

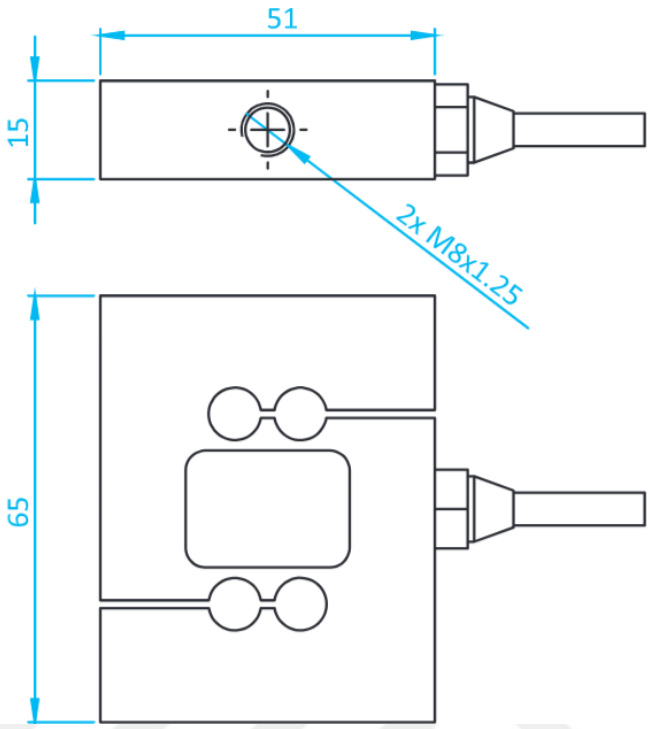

Figure 70: Weilo S-Type Loadcell

Table 24: S-Type loadcell specifications

Weilo S Type Loadcell	
Specification	**Value**
Capacity	100Kg
Accuracy Class	C3
Minimum Load	0 kg
Maximum Number of Load Cell Intervals (nLC)	3000
Minimum Verified Load Cell Interval	12000
Total Error	±0.022 %FS
Output Signal (FS)	2.00 ± 0.005 mV/V
Zero Balance	±1 %FS
Input Resistance	400 Ω ± 20 Ω
Output Resistance	350 Ω ± 3 Ω
Insulation Resistance	≤ 5000 MΩ (100VDC)
Compensated Temperature Range	-10 to +40 °C
Operating Temperature Range	-30 to +70 °C
Recommended Excitation Voltage	10 VDC
Maximum Excitation Voltage (Umax)	15 VDC
Safe Overload	150 %FS
Breaking Load	300 %FS
Protection Class (EN60529)	IP 66
Load Cell Material	Aluminum
Cable	4x0.22 mm

MPU6050 GY-521 (IMU)

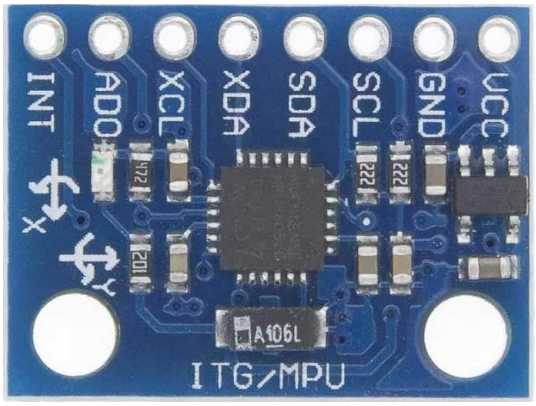

Figure 71: IMU MPU 6050 GY-521 Module

SLA019 Linear Actuator

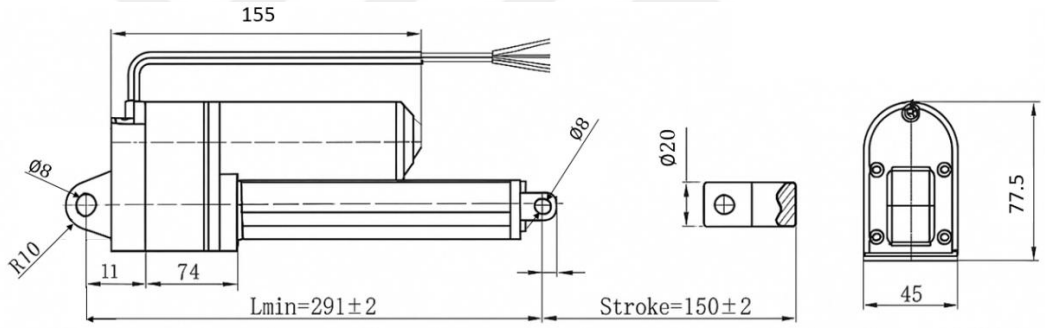

Figure 72: SLA019 Linear Actuator drawing

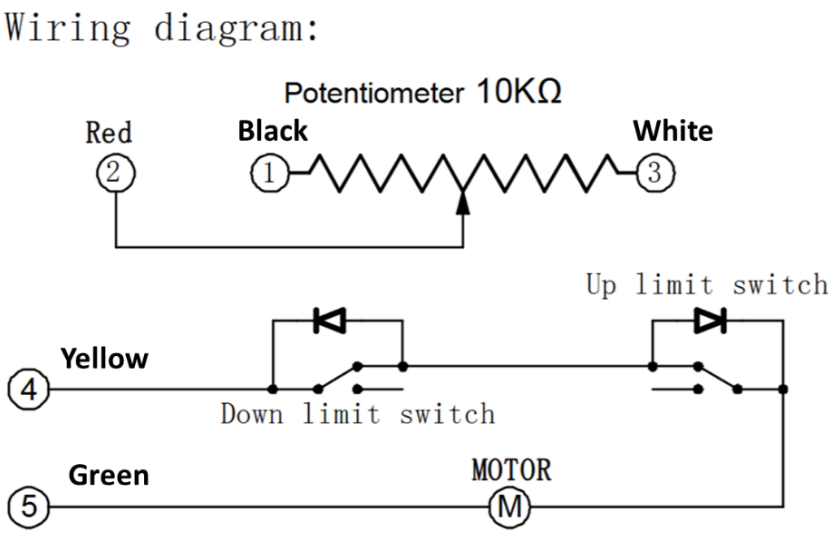

Figure 73: Actuator potentiometer circuit scheme

Table 25: SLA019 150mm linear actuator specifications

Specifications	
Parameter	**Value**
Input Voltage	24V DC
Load Capacity	1000N
Speed	12 mm/s
Stroke	150 mm
Type of duty	10% 2 min. on / 18 min. off
Limit switches	Inner
Operation Temperature	-15°C ~ 45°C
Protection Class	IP65

Arduino Mega 2560 (Microcontroller)

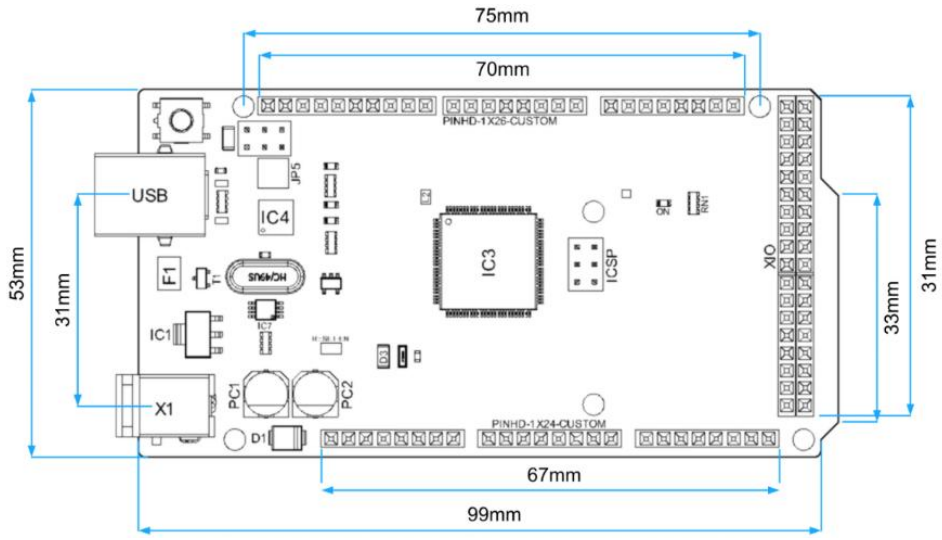

Figure 74: Arduino Mega 2560 general dimensions

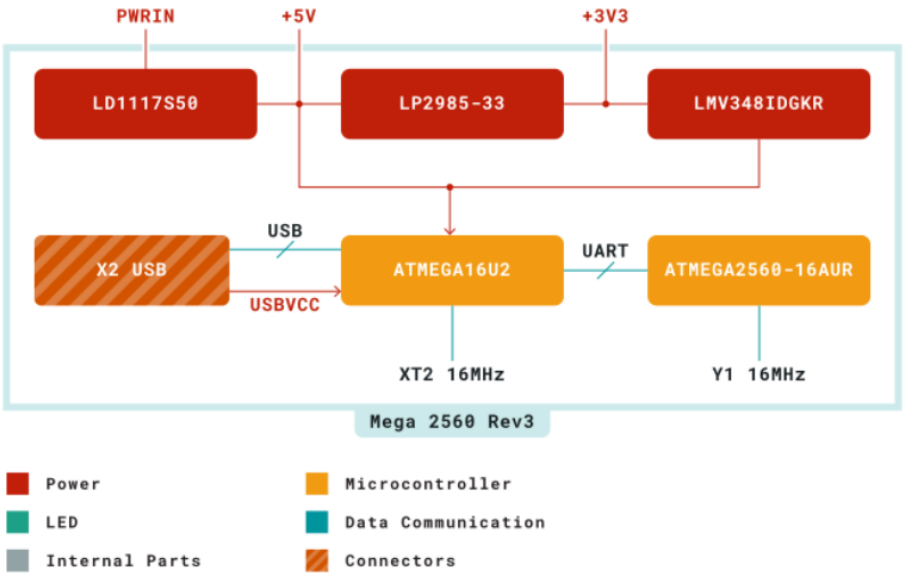

Figure 75: Block Diagram of Arduino Mega 2560

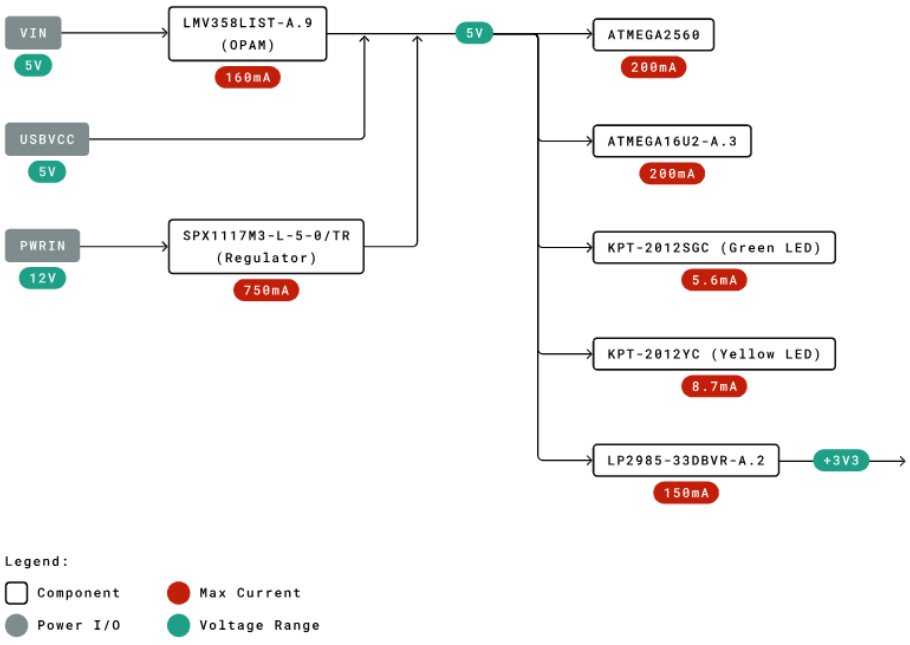

Figure 76: Power Distribution

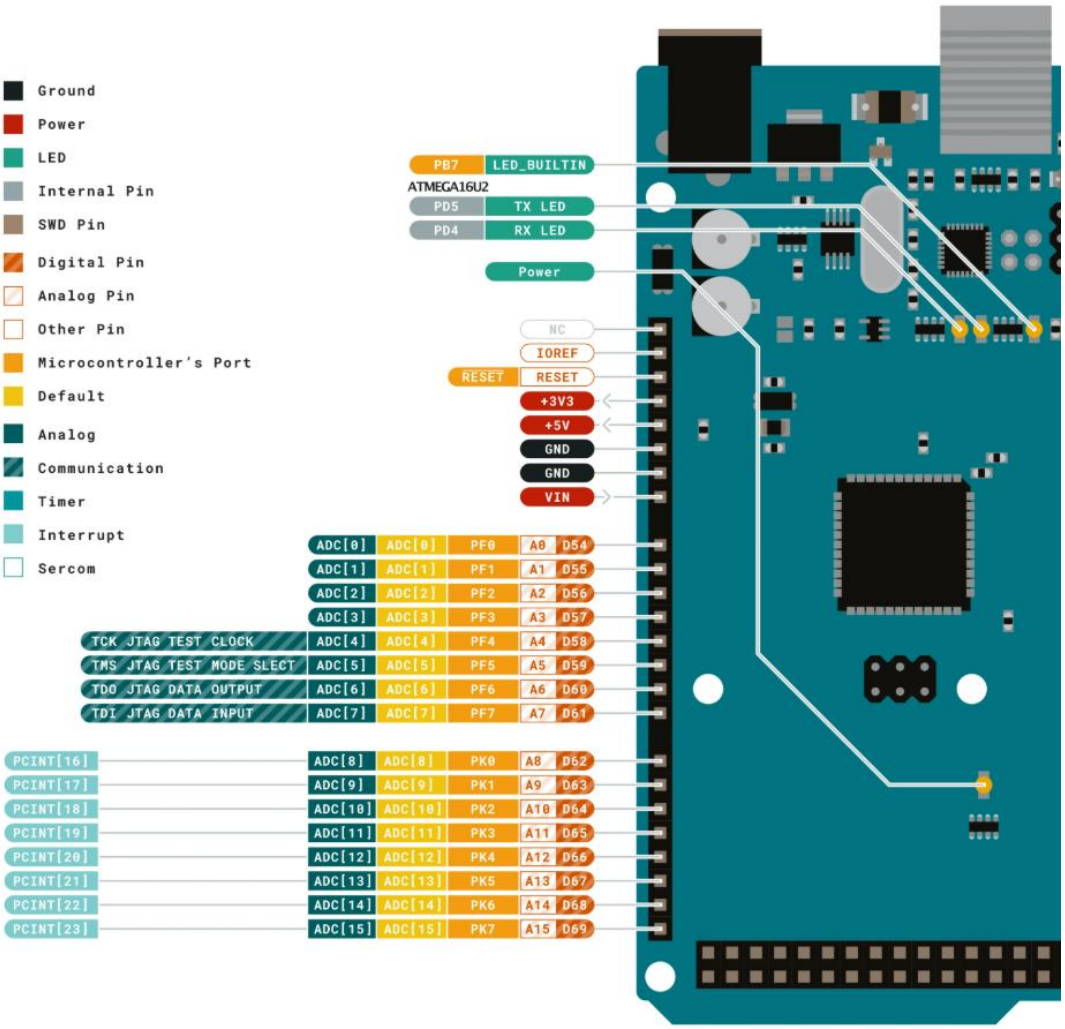

Figure 77: Arduino Mega 2560 pinout (1)

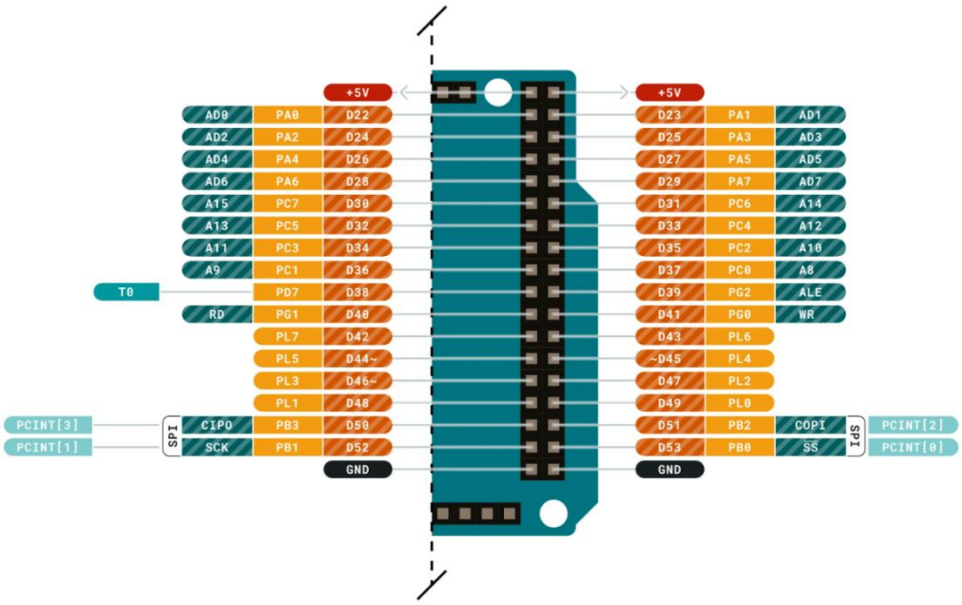

Figure 78: Arduino Mega 2560 pinout (2)

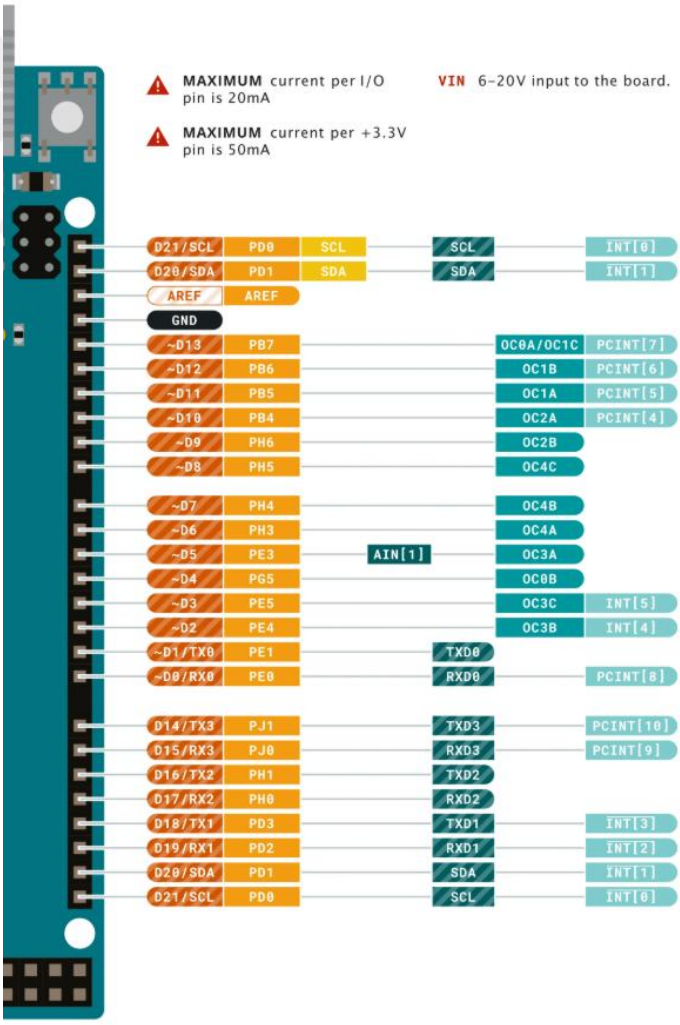

Figure 79: Arduino Mega 2560 pinout (3)

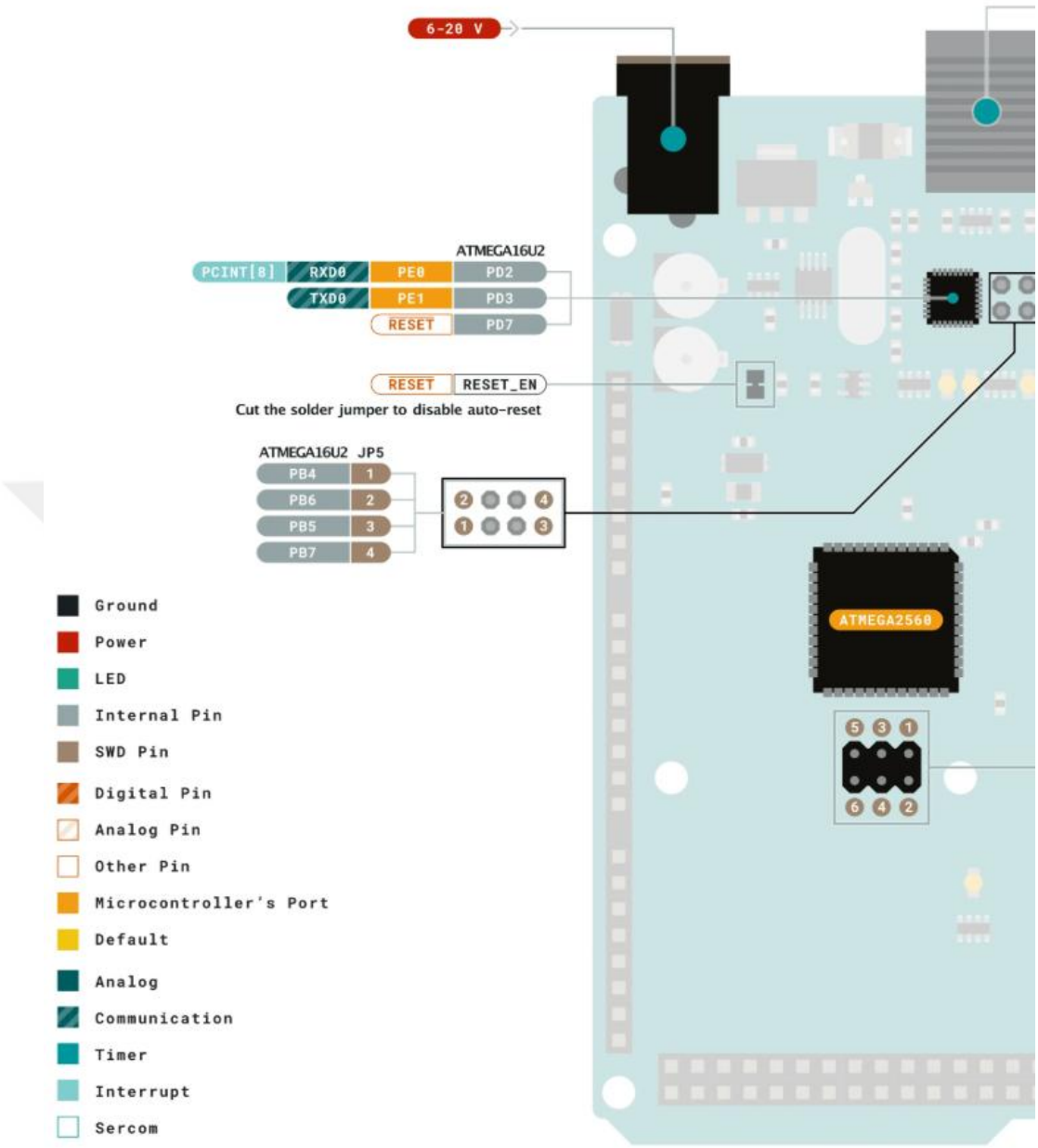

Figure 80: Arduino Mega 2560 pinout (4)

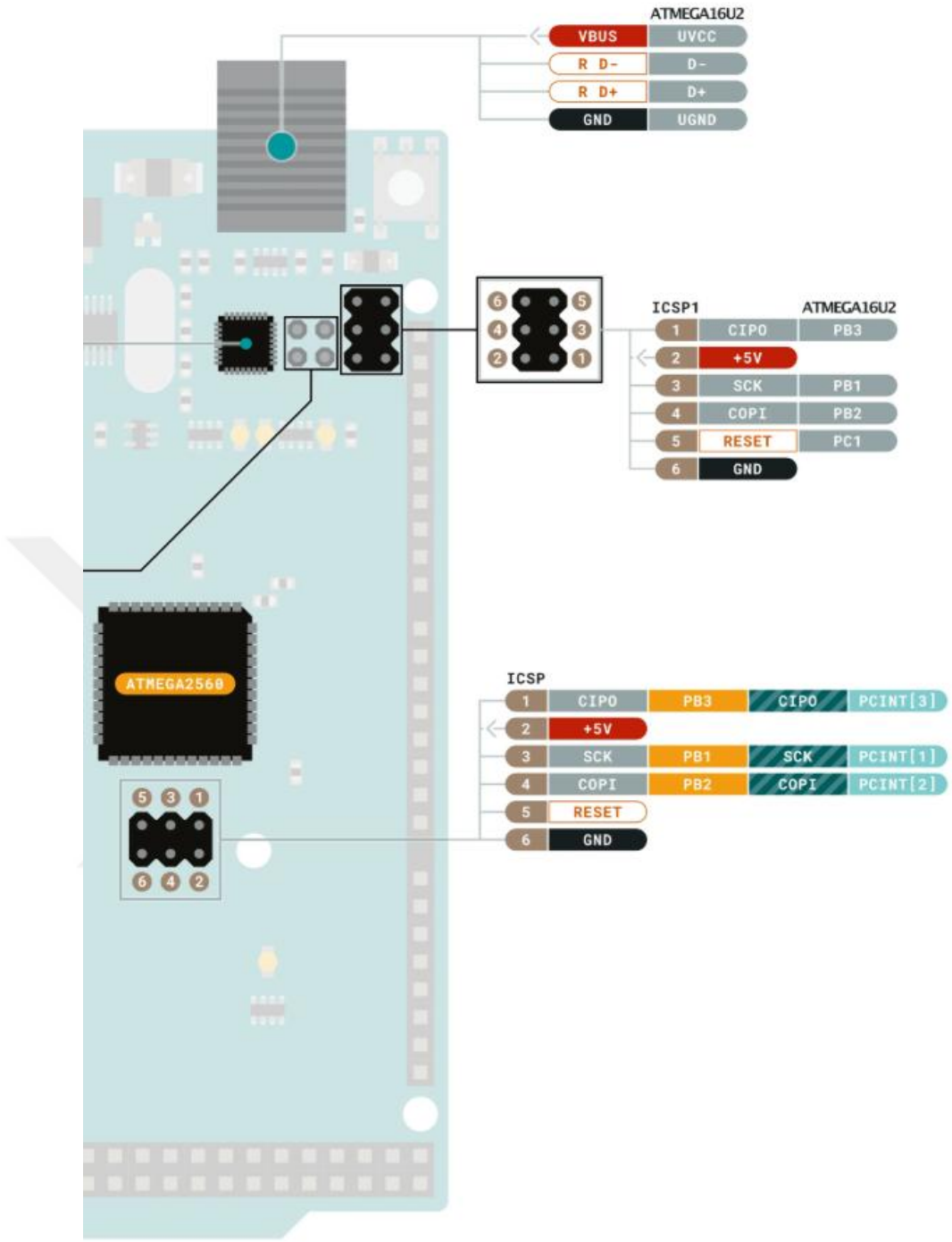

Figure 81: Arduino Mega 2560 pinout (5)

BTS7960B Motor Driver

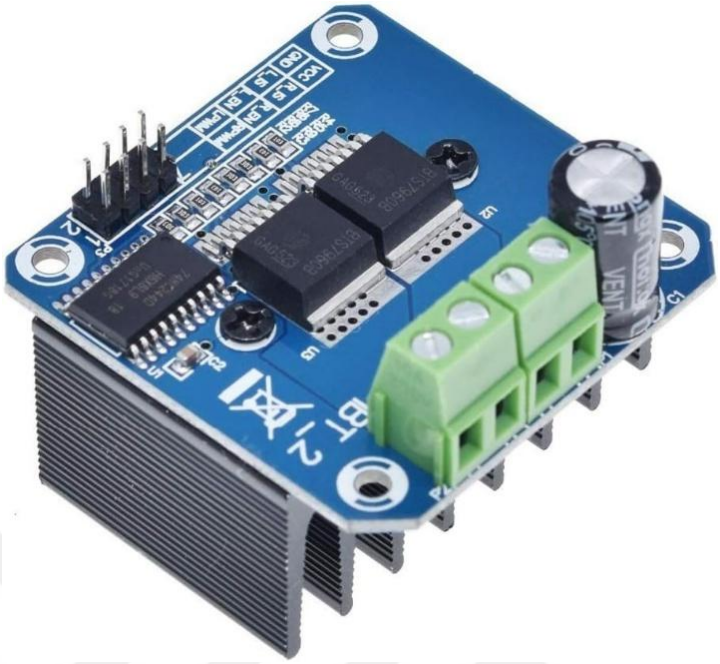

Figure 82: BTS7960B Motor Driver

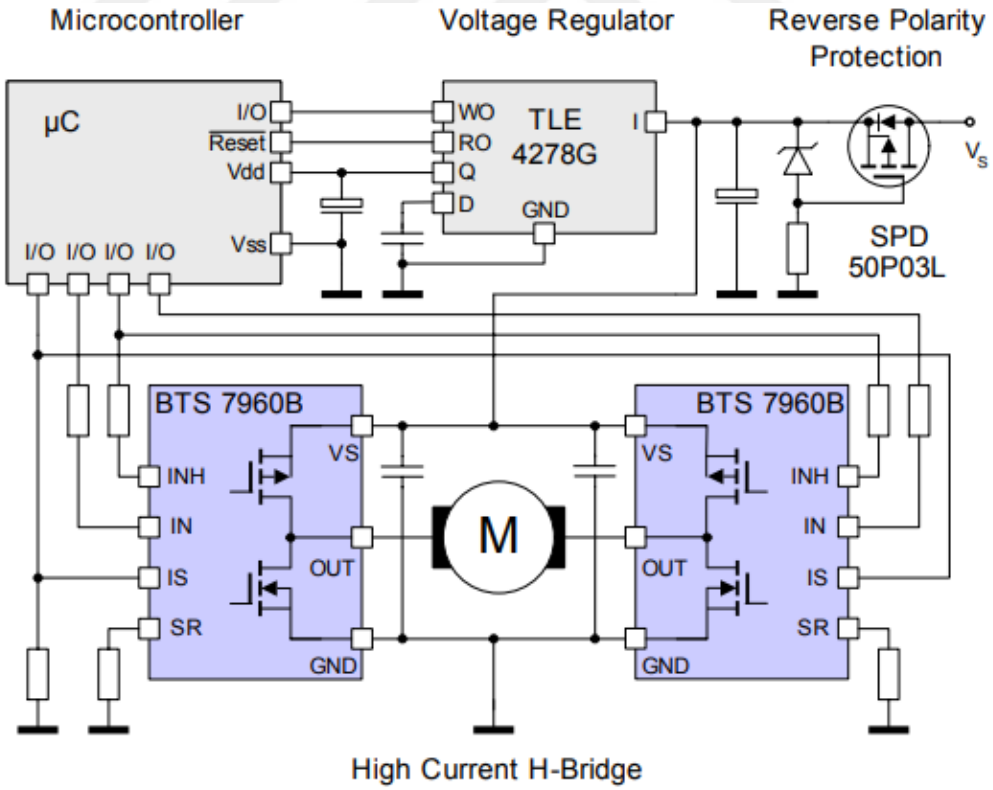

Figure 83: Block diagram of BTS7960B

WEILO M60 Digital Loadcell Indicator

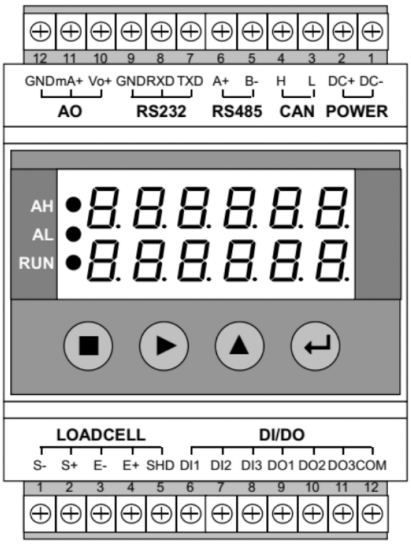

Figure 84: Weilo M60 Digital Loadcell Indicator

Table 26: Weilo M60 Digital Loadcell Indicator specifications

Specifications	
Parameter	**Value**
Accuracy Class	1/30000
Display	6+6 Red LED
Status Indicator	3 Status LEDs
Keypad	4 Function Key Buttons
ADC	Δ-Σ Delta-Sigma 24 bits
A/D Conversion Speed	3200Hz, 1600Hz, 800Hz, 400Hz, 200Hz, 5Hz
Resolution	1/1,000,000 Internal Count
Load Cell Input	0~12.5mV
Linearity & TC	0.005% at Full Capacity
Supply Voltage	5V DC Max 150mA
Number of Load Cells	Max 8 x 350Ω
Connection Type	4 or 6 Wire
Calibration	With reference dead load or by mV/V value
Filtering	9-Level Digital Filter
Digital Inputs	3 Digital Inputs
Outputs	3 Normally Open Transistor Outputs
Serial Port	COM1-RS232, COM2-RS485, COM3-CanBus / MODBUS[ASCII/RTU]
Analog Output 1	0-20mA, non-linearity: 0.05%FS
Analog Output 2	0-10V, non-linearity: 0.05%FS
Enclosure	DIN Rail Type, Dimensions: 71.5×113×57.9mm [W×H×D]
Weight	Approx. 197g
Power Supply	DC12~24V, 3W

Tare and Zero Calibration

To calibrate the device, several steps must be followed. These steps include accessing the calibration settings in the settings panel, calibrating the zero value without loading, and entering and saving the loaded value into the device during loading. To perform this calibration, the following steps can be followed:

1. Press the "square" button on the keyboard to enter the settings.
2. Press the "right arrow" button once to access the F2.CAL (system calibration) heading. Press the "ENTER" button to enter this setting.
3. Use the "right arrow" button to scroll through the menu until "ZERO" appears on the screen. This setting is entered when "ZERO" appears. Press the "ENTER" button while there is no load on the load cell to save the zero value.
4. Use the "right arrow" button to scroll through the menu until "LOAD" appears on the screen. This setting is entered when "LOAD" appears. The value of the loaded weight is entered on the screen. While the load cell is still loaded, press the "ENTER" key to save the value.
5. Press the "square" symbol on the keypad to access the main display screen. The load cell is now weighing at the correct value. This can be verified by applying loads at different values.

After calibration, while entering any value on the indicator screen, the displayed value can be scanned by long-pressing the "right arrow" symbol.

HX711 Signal Amplifier

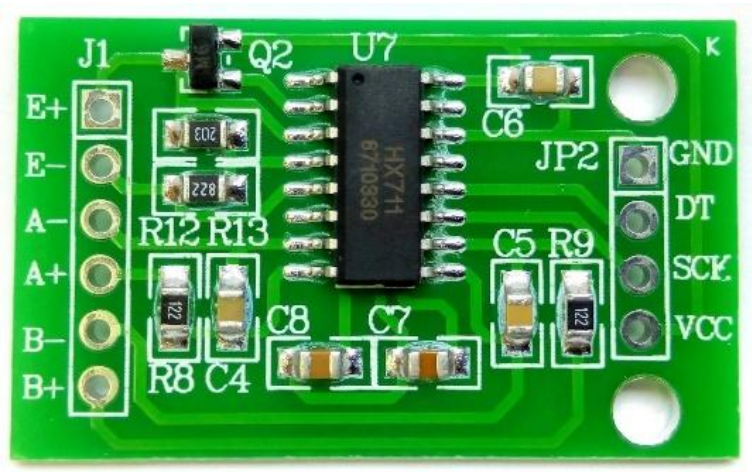

Figure 85: HX711 Signal Amplifier

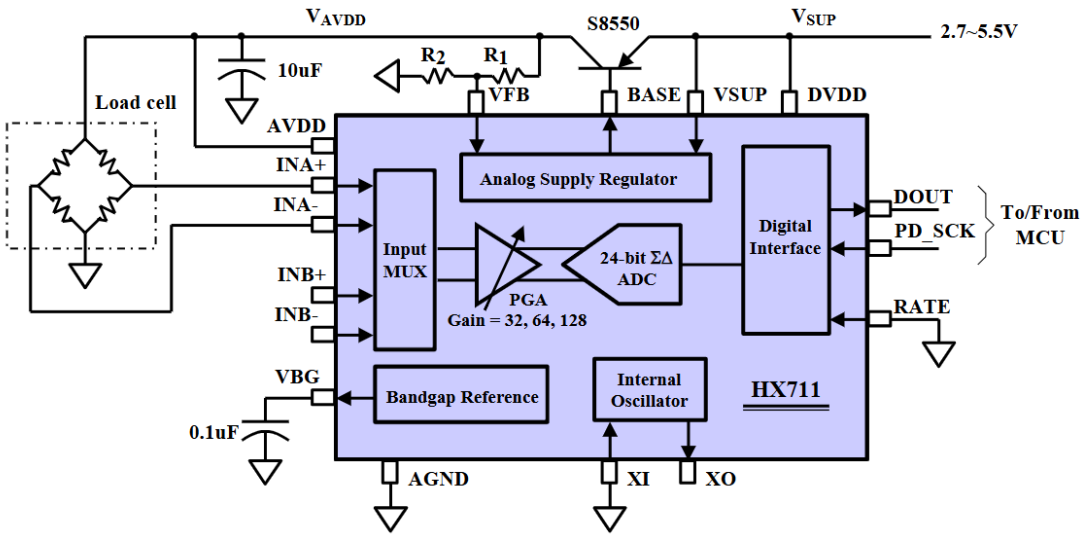

Figure 86: HX711 Block Diagram

RS232 to TTL Convertor

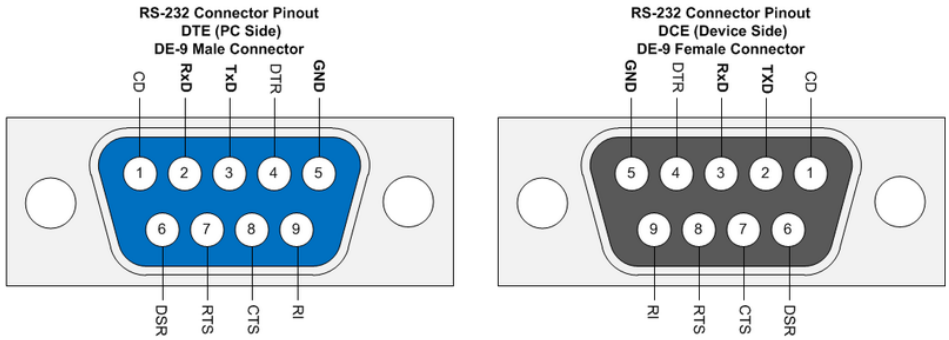

Figure 87: RS232 DB9 Connector Pinouts

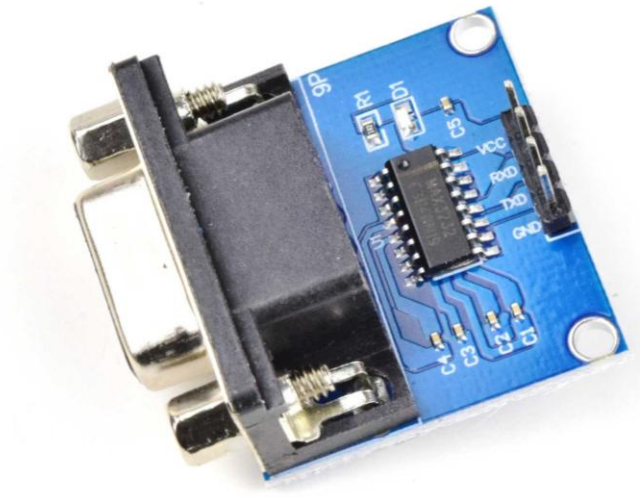

Figure 88: RS232 to TTL convertor

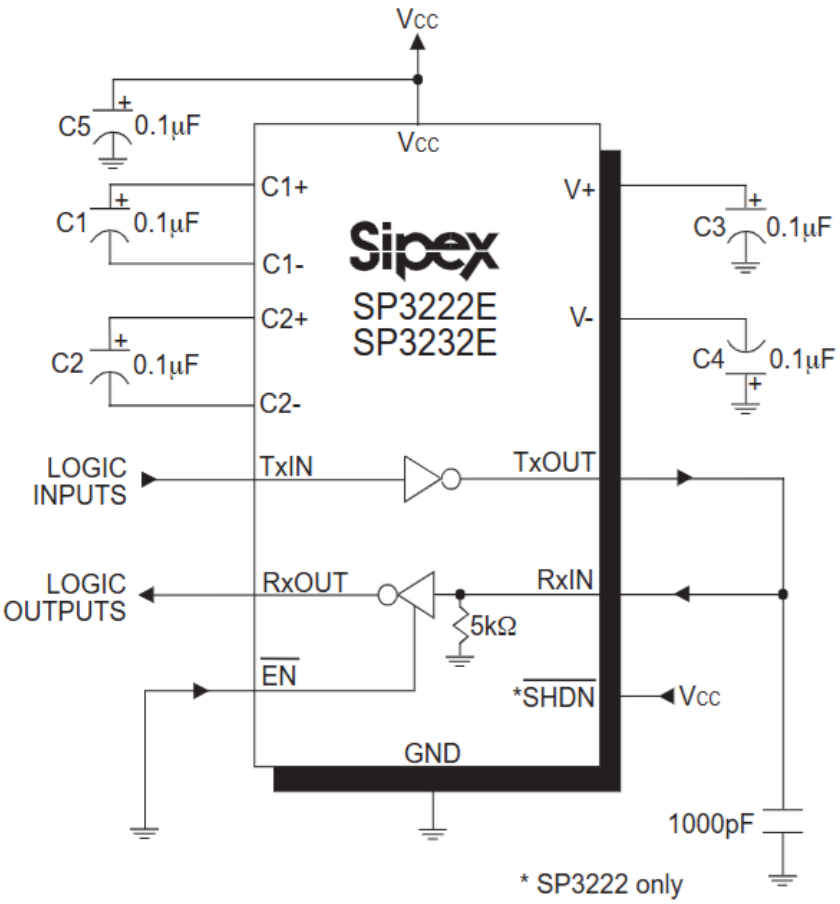

Figure 89: Electrical Diagram of TTL convertor

SD Card Module

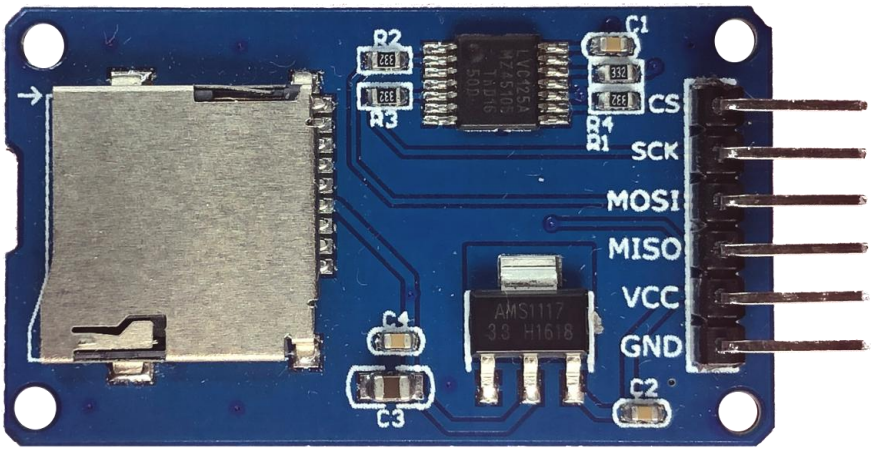

Figure 90: Sd Card Module