A Hopping Two-Wheeled Segway

A Major Qualifying Project Report submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science

Submitted by

Samuel Mwangi
Robotics Engineering

Christian DeFranco
Robotics Engineering

December 21, 2022

Prof. Siavash Farzan, Advisor
Robotics Engineering Department

ABSTRACT

Small electric vehicles have revolutionized the means of transport in the recent years as a sustainable and convenient alternative to traditional modes of transportation in urban areas. Environmental problems such as global warming brought about by carbon emissions from vehicles has pushed for the need of more sustainable and easily accessible means of locomotion such as electric bicycles, electric scooters, and hoverboards, among many other small vehicles that are used in large industries and urban areas. Two-wheeled mobile robots (such as Segway) provide a high level of mobility and maneuverability and have great potential to be adopted in real-life settings. Compared with other small electric vehicles, Segways have a greater potential in being adapted as the future mode of short distance transportation due to their integration of robotics allowing safety features such as self-balancing and automated speed control. However, the current Segway platforms available in the market are unable to navigate unstructured environments. The goals of this project are two-fold: to develop an embedded control system for a commercial Segway to be driven autonomously, and to retrofit the Segway with a jumping mechanism to enable it to travel on uneven terrains. The final design offers a new class of two-wheeled mobile robots that employ jumping as a means of locomotion in non-continuous terrains.

ACKNOWLEDGEMENTS

Our team would like to thank several individuals for making this project a success. Firstly, we would like to thank our project advisors Prof. Siavash Farzan (RBE) for coming up with the idea and providing guidance during the project's execution. We would also like to thank Worcester Polytechnic Institute Robotics Engineering Department for providing us with a conducive and supportive atmosphere to conduct our project. Additionally, we would like to thank Mechanical Engineering student Daniel Tribaldos for his assistance with the waterjet. Lastly, we would like to acknowledge ODrive Robotics for providing general guidance through answering questions about implementation of the hardware and software components for the project.

AUTHORSHIP

Chapter/Section		**Author**
Abstract		Samuel Mwangi Christian DeFranco
Introduction		Samuel Mwangi
Background		Christian DeFranco
Design and Development	System Overview	Samuel Mwangi
	Embedded System Development and Electrical Design	Samuel Mwangi
	Main Processor	Samuel Mwangi
	Brushless DC Motor Drivers	Samuel Mwangi
	Inertial Measurement Unit (IMU) Sensor	Samuel Mwangi
	Battery	Samuel Mwangi
	Drive Mechanism	Samuel Mwangi
	Self Balancing Control System	Samuel Mwangi
	Hardware Design	Christian DeFranco
System Testing and Validation	Self–balancing Mechanism Testing	Samuel Mwangi
	Jumping Mechanism Testing	Christian DeFranco
Discussions		Samuel Mwangi Christian DeFranco
Conclusions		Samuel Mwangi Christian DeFranco
Recommendations		Samuel Mwangi Christian DeFranco

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Recent developments in technology have changed the means of short distance transportation. Advancements in technology such as integration of microchips in devices has especially revolutionized and eased the means of travelling. The 21st century has seen the invention of electric bikes, scooters, electric skateboards, and the Segway (Wikipedia, 2022b), as a faster and more efficient means of short distance transportation. Segways are innovative personal transportation devices that have gained popularity in recent years due to their convenience and efficiency. These self-balancing electric scooters have revolutionized the way we travel short distances, providing a sustainable and eco-friendly alternative to traditional modes of transportation. Segways are particularly useful in urban areas where traffic congestion and limited parking can be an issue. They are also popular in large industries and malls for their ease of use and ability to cover long distances quickly. In addition, Segways have proven to be a reliable option for tourists and sightseers, allowing them to easily navigate through crowded areas and cover more ground in a shorter amount of time. Overall, the use of Segways in transportation has proven to be a valuable addition, offering a fun and efficient way to travel.

A Segway can be considered as an inverted pendulum on a two-wheel car. The inverted pendulum is a well-known model in the fields of dynamics and control theory that is often used to evaluate control strategies (Wikipedia, 2022a). It consists of a pivot pole placed on a cart that is able to move horizontally and is controlled by an electrical servo system. The inverted pendulum model has its center of mass above its pivot point, which makes it unstable and prone to falling over if not supported. However, a control system can be used to monitor the angle of the pole and move the pivot point horizontally beneath the center of mass when it begins to fall, thus keeping the pendulum balanced and upright. By employing this control system, the inverted pendulum can be stabilized and maintained in an upright position (Boubaker, 2013).

The Segway is a popular personal transportation device that has revolutionized the way we move around in urban and suburban areas. In this project, we are concentrating on designing a more versatile Segway with distinctive features that are revolutionizing personal mobility for both consumers and enterprises. The current small vehicle market only offers two directional movement vehicles. Those are vehicles that can only move

forward and backwards. These vehicles have been successful, but they are limited to be operated by a user and can only be driven on *flat* terrains on controlled environments such as factories, malls, and tarmacked roads.

The inability of Segways to navigate uneven terrains has limited its utility in certain situations. These limitations inspired our main project goal which is to revolutionize the current Segway model by modifying it and integrating new designs allowing more functionalities to serve a broader community. In this project, we will be exploring the possibility of adding a jumping mechanism to the Segway to allow it to overcome obstacles and navigate uneven terrains. This will involve designing and building a jumping mechanism, integrating it into the Segway, and testing its performance in various terrain conditions. Additionally, we will design an embedded system for autonomous control and stabilization of the platform, allowing the Segway to be driven without a user in charge.

In summary, the current project has two main objectives. First is to build an automated self-balancing system for the current Ninebot Segway models, i.e., to allow operation of the Segway without necessary manpower. Second is to customize the current Segway models by designing and installing a jumping mechanism allowing three-dimensional movement to avoid possible obstacles. To achieve this objective, we modified the current Ninebot S Segway model (Segway Inc, 2022) by pulling it apart, salvaging all usable components followed by designing and installing a jumping mechanism and self-balancing control system.

Our project aims to address the challenges of urban congestion and environmental issues with a unique solution: a single-occupant vehicle with advanced dynamics, capabilities, and sensory technologies. Building on the foundation of existing small electric vehicles, we plan to customize a Segway to retain its original appearance, feel, and operation, while incorporating additional functionalities. This approach allows for a quick development process and the opportunity to create a fully functional vehicle that can address the needs of modern cities and urban spaces.

Chapter 2

BACKGROUND

2.1 Applications of a two-wheeled jumping robot

Two-wheeled jumping robots offer several potential benefits over traditional Segways and other types of high-mobility robots. These benefits include improved maneuverability, increased versatility, and a more dynamic range of speed.

One of the primary advantages of a Segway-style robot is improved maneuverability. Because these robots only have two wheels, they can turn and move in ways that would be impossible for traditional four-wheeled or legged robots. With only two points of contact with the ground, zero-point turns are made possible, which means that the robot can turn in place much quicker than alternative designs. This allows them to navigate tight spaces and obstacles with ease, making them well-suited for use in environments where traditional robots may struggle.

Additionally, hopping Segways can jump and move in three dimensions, giving them a much wider range of motion than traditional Segways. This allows them to easily traverse rough terrain and obstacles, making them well-suited in a variety of both indoor and outdoor environments. For example, hopping Segways could be used in search and rescue operations, allowing rescue workers to quickly and easily navigate difficult terrain to reach stranded individuals.

Furthermore, hopping Segways are much more versatile than traditional Segways. Because of their increased range of motion, these robots can be used for a wider variety of tasks. They could be used for delivery services, for example, allowing packages to be quickly and easily transported to locations that may be difficult for traditional vehicles to reach. Many delivery robots exist today, but few can navigate obstacles such as stairs and rough terrain. Hopping Segways could also be used for private and public security. From law enforcement to area surveillance, these robots could quickly and accurately maneuver in ways that would be difficult for other types of robots that are in use today.

2.2 Current Solutions

This section discusses other highly maneuverable jumping robots that exist today. While there are no previous WPI robotic projects that explore this concept, there are a few

examples around the world. It is important to review other examples of this technology.

2.2.1 Ascento

Ascento, a two wheeled jumping robot designed by Ascento Robotics, is currently in the pre-start-up phase. Based in Switzerland, this team of engineers has developed a two-wheeled robot that can jump and self-balance (Klemm et al., 2019).

Figure 2.1: Ascento Robot

Similarly, to the objective set in this project, the Ascento team set out to design an agile two-wheeled jumping robot in a compact design.

2.2.2 Tencent Ollie

Developed by tech company Tencent based in China, Ollie is a wheel-legged robot capable of various difficult maneuvers. The robot, shown in Figure 2.2, can drive, jump, and even perform flips with the help of a third leg.

Designed to adapt to uneven ground and perform jumping maneuvers, Ollie uses trajectory planning before jumps to maximize the joint motor performance. A characteristic unique to Tencent Ollie is its tail, which allows it to build up more momentum when jumping. The addition of this feature helps the robot perform flips while in the air.

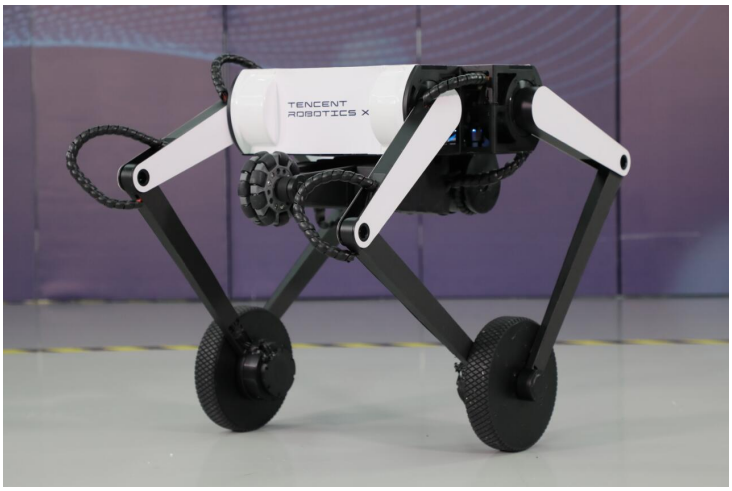

Figure 2.2: Tencent Ollie Robot

2.3 Proposed Designs

Across all existing designs, jumping is achieved through rapid extension and retraction of the robot's legs. By keeping as much of the mass as possible towards the top of the robot, Initial acceleration is more difficult, but retraction is made easier. This was taken into consideration when designing our mechanism. Our challenge, however, was the need to keep as much of the original Segway intact as possible. This made it increasingly difficult to transfer mass to the top of the Segway, as the chassis which contains all the electrical and mechanical components consisted of a large chunk of magnesium-alloy.

While moving some of the mass from the chassis to the top of our linkage would help facilitate jumping, our team had a few concerns. Firstly, the wheels of the Segway Ninebot S make up approximately eight kilograms of the Segway's total 12.8-kilogram weight. This is because the integrated motors built into the wheels are designed to provide enough torque to move a 220 lb. Human. This mass was immovable because of its need to maintain contact with the ground. Secondly, building a robust linkage capable of making a 11.4-kilogram Segway jump could end up adding too much weight to the chassis and therefore hindering jumping.

All these factors and concerns were considered when designing our jumping mechanism. After moving the battery and some electrical components to the counter mass on top of the linkage, the fixed weight made up approximately 11.4 kilograms.

Chapter 3

DESIGN AND DEVELOPMENT

3.1 System Overview

Our main goal is to customize, design and develop a hopping Segway suitable for maneuvering all terrains autonomously including avoiding obstacles through a jumping mechanism. We began by researching and studying the current available Segway in the market and identifying their weaknesses and improvements. We were then provided with a base model of the Ninebot S Segway which would be our starting prototype. Segway Inc has established itself as a pioneer in the rapidly developing small electric vehicles such as electric scooters and other Segway models.

However, Segway Inc is not open-source and does not provide information on their products, therefore most components of the Ninebot S Segway are not reusable for other projects. For this project we aimed to design a jumping mechanism as an add on for the current Segway model whilst maintaining the current shape and chassis design. Due to the time constraint of this project and limited availability of components we hoped to reuse as many parts as possible of the Ninebot S Segway model. After stripping down the provided Segway model we identified the reusable components, namely, Chassis frame, brushless motors, Wheels, and the lithium-ion battery. The embedded system, sensors, and control system are all black box copyrighted to Segway Inc and therefore we would have to research and figure out suitable replacement parts for these components.

Our two main objectives for this project were:

i) Design and integrate an autonomous self-balancing system while maintaining functionality of the current Ninebot S Segway system.

ii) Design, prototype and incorporate a jumping mechanism to the current Segway system.

After salvaging parts from the Ninebot S Segway system, we listed known specifications from Ninebot S Segway and design requirements that we hope to accomplish by the end of this project as shown below.

Category	Item	N3M240 Parameters
Weight	Payload	88 – 220 lbs. (40 – 100 kgs)
	Net Weight	Approx. 28 lbs. (12.8kg)
Rider Requirements	Ages	$16 \sim 50$ years
	Height	4'3" – 6'6" (130 – 200cm)
Vehicle	Maximum Speed	Approx. 10 mph (16 km/h)
	Typical range	Approx. 13.7 miles (22 km)
	Maximum slope	Approx. 15°
	Traversable terrain	Hard roads, flat concrete roads, slopes of less than 15 degrees, steps not higher than 1 cm, and dips no more than 3cm wide
	Operating temperature	$14 - 104°F$ $(-10 - 40°C)$
	Storage temperature	$-4 - 122°F(-20 - 50°C)$
	Permitted Charging temperature	$32 - 104°F(0 - 40°C)$
	IP rating	IP54
Battery Pack	Rated Voltage	54.8 V
	Maximum charging voltage	59.5 V
	Rated capacity	236 Wh
	Smart BMS	Overvoltage/Under Voltage/Short Circuit/Overheating Protection. Auto-sleep/Wake-up
	Maximum continuous discharge power	1000W
	Charging Temperature	$32 - 104$ °$(0 - 40°C)$
Motor	Rated power	400 x 2 W
	Maximum power	800 x 2 W

Table 3.1: Summary of key designs requirements and specifications.

3.2 Embedded System Development and Electrical Design

The main computer/heart of the Segway is the Raspberry Pi 4. In this project, the Raspberry Pi is also the main controller of the Segway. It is powered by a solar power bank via Type - C charging port. Raspberry Pi facilitates communication by receiving sensor data from the IMU sensor or commands wirelessly via SSH and in return it sends out signals to the ODrive motor driver.

The wheels used in this project are high current 400W brushless DC motors. For that matter, we selected ODrive V3.6 as the most suitable motor driver to power the wheels. ODrive motor drive has a peak current of 120A per motor and a DC power voltage input range of 12V to 56V. For it being a high current motor driver, a 2 Ohm break resistor is

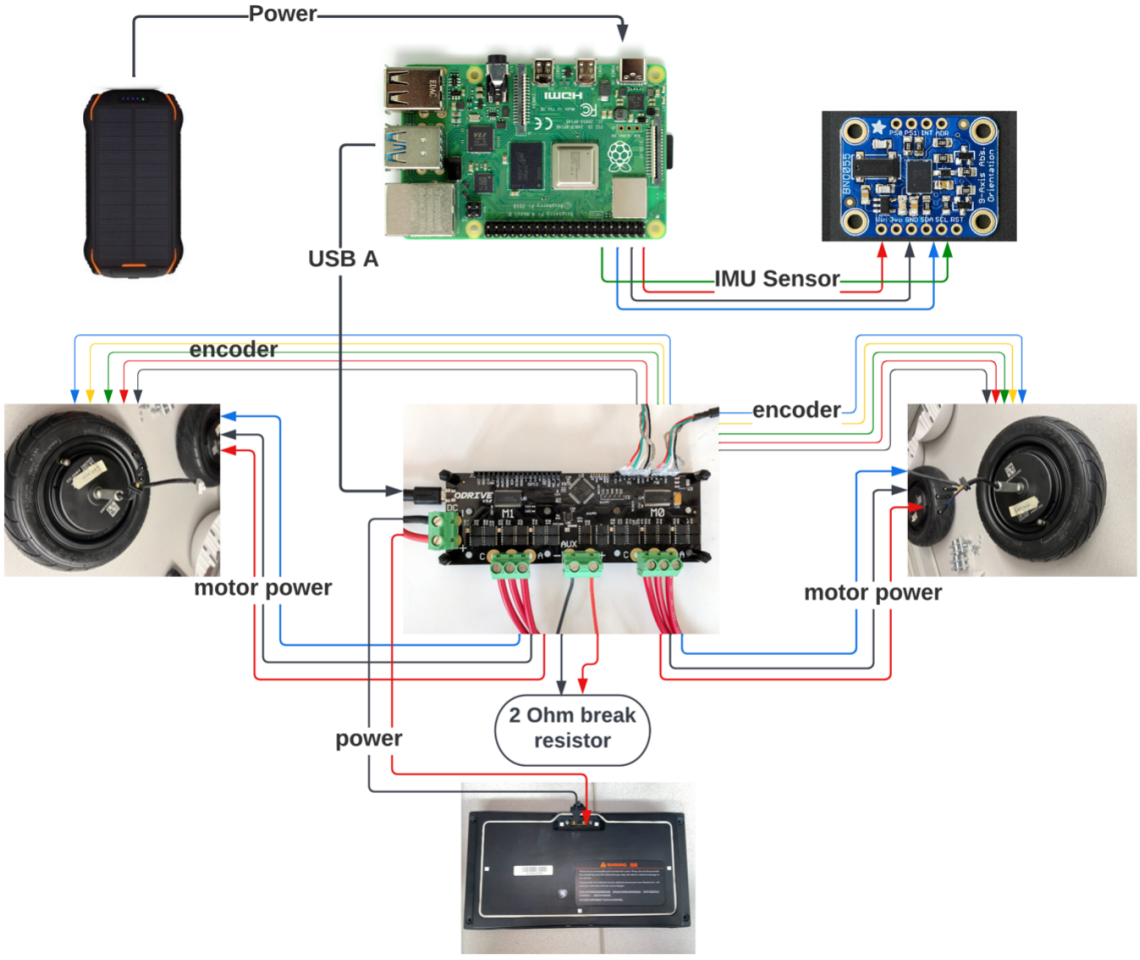

Figure 3.1: Motor and Motor driver wiring diagram.

attached to it to reduce noises caused by high current. The ODrive motor driver is then powered by the original Segway lithium battery which has a rated voltage of 54.6 V and a max current of 14.59A. Figure 3.1 above shows the main system's wiring diagram.

Details of components used:

Following are the list of components used in the making of the Segway:

1. Raspberry Pi 4B
2. ODrive V3.6 56V motor driver
3. 2 Ohm Break resistor.
4. BNO055 Adafruit 9-DOF IMU

5. 400W Ninebot S brushless DC motors.

6. 54.6V Segway battery

7. Solar powered battery

3.2.1 Main Processor

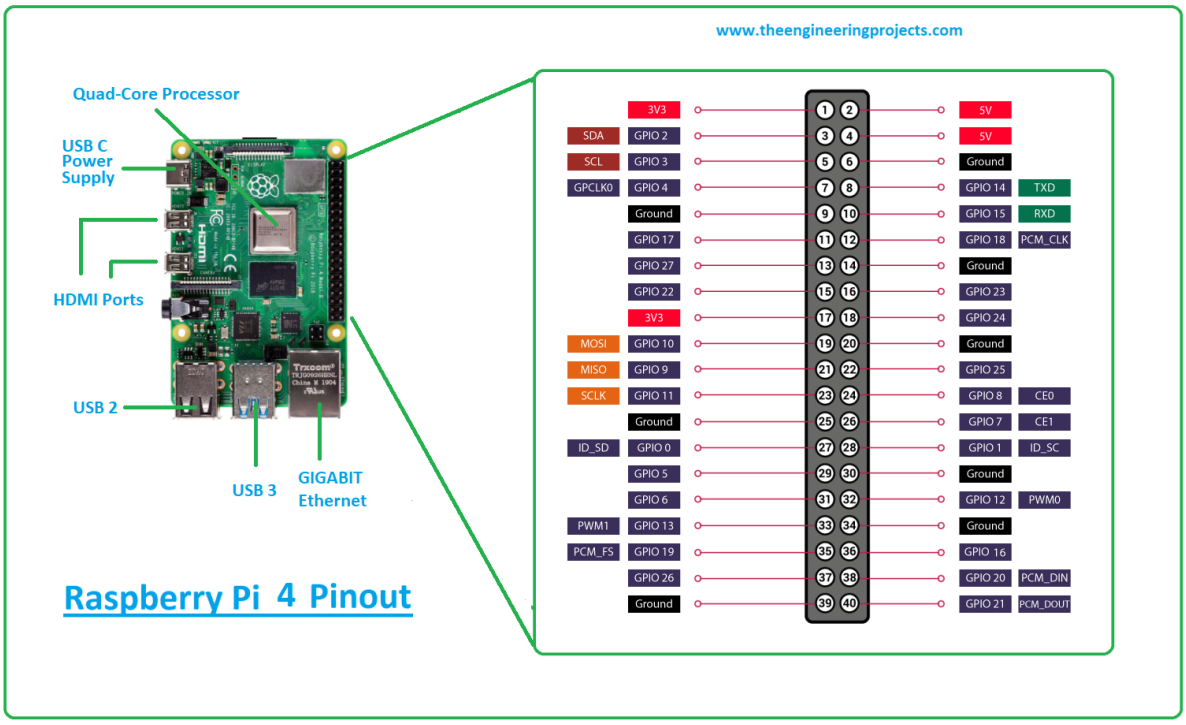

Figure 3.2: Raspberry Pi 4 and its pinout diagram.

We used the Raspberry Pi 4 as the main processor for the embedded control system. The Raspberry Pi 4 is a credit card sized open-source computer. It is a compact, fully working computer that may be connected to a keyboard, mouse, and computer monitor. It has a specialized processor, memory, and graphics driver, which are all characteristics of a PC. It even features a customized version of Linux, named Raspberry Pi OS, as its operating system. However, the Raspberry Pi does not offer storage. In this case, we used a 128 GB SD card to store Raspberry Pi operating system. Additionally equipped with Bluetooth, ethernet, and Wi-Fi connectivity, Raspberry Pi enables file transfers over the internet and wireless communication via SSH. Figure **??** above shows features of the Raspberry Pi 4 and its pinout diagram.

Features:

- 64-bit Quad Core processor
- Large RAM (Random access memory) of 8 GB
- Processor speed - 700MHz-1.5GHz
- Has 40 input/output pins
- Equipped with Bluetooth and Wi-Fi connectivity.
- It contains everything – CPU (Central Processing Unit), GPU (Graphics Processing Unit), Ethernet port, GPIO (General-purpose Input/Output) pins and a type C power connector

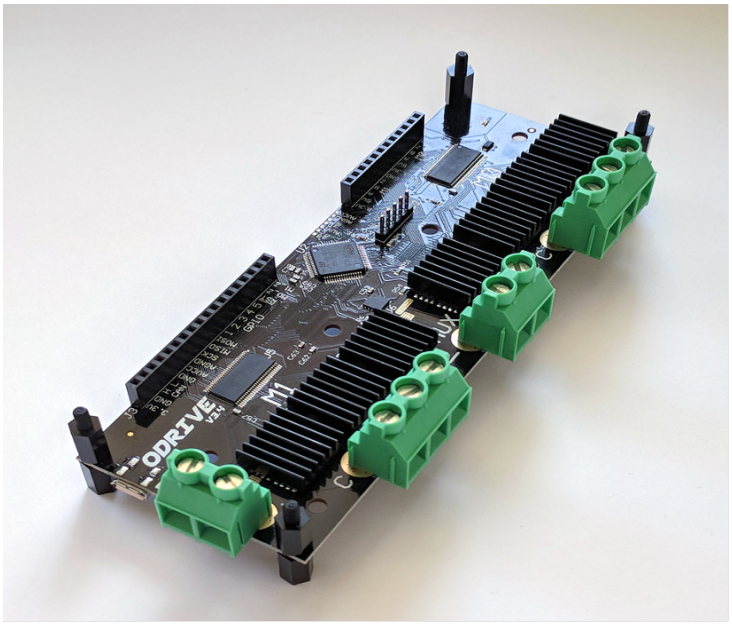

Figure 3.3: ODrive Motor driver with Connectors.

3.2.2 Brushless DC Motor Drivers

To drive the two main brushless motors of the Segway autonomously, we used the ODRIVE V3.6 driver board. ODrive V3.6 is an open-source brushless motor drive board which supports two hall effect sensored brushless motors. For this project, we use the ODrive to power and drive the two brushless as well as read data from each motor's hall sensor. The three-motor phases are wired into the ODrive directly using the 3 phase screw terminals. In the same way the hall sensors are plugged into their corresponding pins. i.e., A, B, C phases and powered via the 5V pin as shown in the chart below.

Encoder	**ODrive**
Red	5V
Yellow	Hall_A
Blue	Hall_B
Green	Hall_C
Black	GND

This data is then relayed to the Raspberry Pi via USB connector. The board comes with an inbuilt firmware which supports the following:

- Position, Velocity and Current Control modes
- Absolute encoders
- Hall effect sensors
- Realtime USB communication (>4000 floats/sec) to Python host program. Which you can interface to your own application
- Automatic identification of motor parameter i.e., Bus Voltage, current, Inductance, and resistance
- PWM input (can be interfaced with a remote control)
- UART communication.

It should be noted that brushless motors do not have brakes and for that reason, braking/stopping the motors is only possible through negative current. This is the amount of current allowed to flow back into the power supply. This current is wasted current and only leads to noises/heat in motor driver board. To mitigate this noise from braking motors we wired a 2 Ohm brake resistor to the ODrive, and its main purpose is to consume all the negative current from braking motors and dissipate it in form of heat.

It should be also mentioned that hall sensors are very sensitive, and they are affected by noises from high current motors. For that reason, we solder some 22nF capacitors from the hall signals to the GND (as shown in the figure below) to filter out as much noise as possible. Without capacitors, the hall sensors give errored feedback.

3.2.3 Inertial Measurement Unit (IMU) Sensor

The BNO055 IMU (Adafruit Industries, 2022), as shown in Figure 3.4, is an electronic device that uses a combination of accelerometer and gyroscope to measure and report a body's specific force. In our model the IMU is responsible for self-balancing mechanism by determining the Segway's orientation. The Raspberry Pi receives data from the IMU and outputs readable yaw, pitch and roll of the robot. The BNO055 is wired to the Raspberry Pi 4 via I2C interface. I.e., The SDA (Serial data) and SCL (serial Clock) of the IMU connects with SDA and SCL of the Raspberry Pi 4 respectively. It is then powered by the Raspberry Pi 4 via the 5V pin.

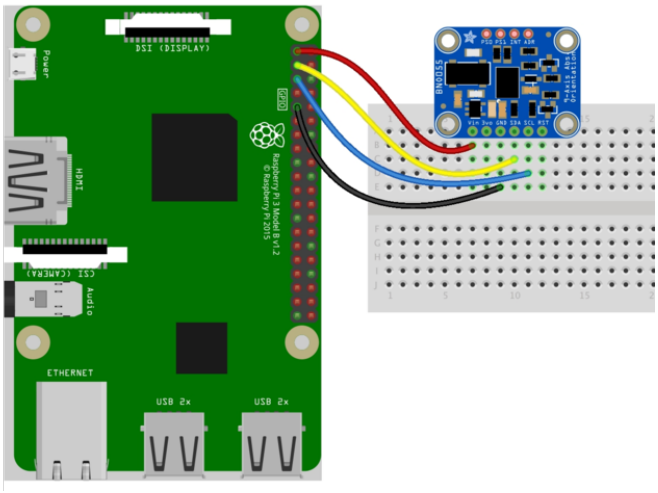

Figure 3.4: BNO055 IMU on a breadboard attached to Raspberry Pi (Adafruit, 2022).

3.2.4 Battery

Figure 3.5: Lithium-ion Battery removed from Ninebot S Chassis.

Ninebot S default battery is a lithium-ion battery with a rated voltage of 54.8V and rated capacity of 236 Wh. It has a smart BMS controller installed whose main purpose is to protect the battery from Overvoltage, Undervoltage, Short circuit and Overheating. In our model we use this battery to power the ODrive motor driver. The positive terminal of the battery connects to the positive terminal of ODrive and the same applies to the negative terminal. It is good to note that this battery has been designed specifically for the Ninebot S Segway board and due to this, the inbuilt smart BMS controller is not usable for our model. In place of the Smart BMS function we use ODrive's Controller to determine the voltage, current, inductance and resistance of the battery. Figure 3.5 shown above shows a top and bottom view of the Segway's battery.

3.2.5 Drive Mechanism

Figure 3.6: Segway Ninebot S model Left Wheel.

The wheel design includes 70/65 tubeless tires with an inbuilt brushless dc motor. Each wheel has an outer diameter of 6.3 inches and an inner diameter of 6.3 inches. In order to cut costs but also meet the requirements of a light weight but stable design, original wheels from the Ninebot S Segway (shown in Figure 3.6) were most suitable for this design. Each wheel weighs an approximate mass of 3.91 Kg, and it can support a maximum load of 70 kg.

Tire specifications:

- Size: Approx. 22.5 * 15.5 * 8cm / 8.9 * 6.1 * 3.1 inch
- Weight: Approx. 523g
- Max load 70 kg (155 lbs.) at 210 kPa

- 2 ply rating
- Design: Tubeless
- Material: Rubber
- Casing material: Combination of fiber and plastic

Brushless DC Motors: Appropriate motor selection depended on the speed control, reliability, cost, and maximum load. In this case we had to decide between brushed direct current (dc) motors and brushless direct current (dc) motors.

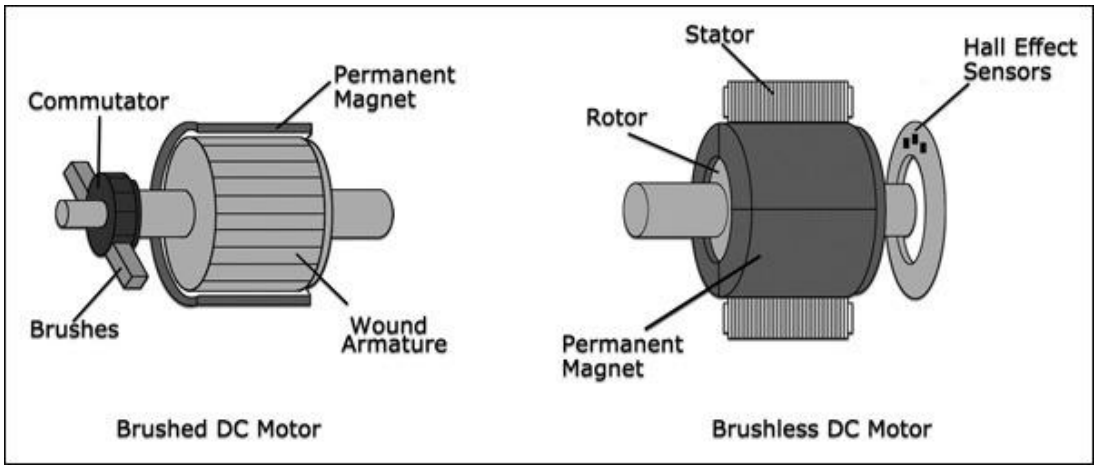

Figure 3.7: Comparison figure of Brushed and Brushless DC Motor (Vovyo Technology, 2021).

A brushless DC motor is a type of electric motor that has no brushes. The brushes are sliding devices which allow the electricity to flow into the engine's copper coil. The brushless motors have no sliding elements on the collector, so this completely solves the problem of maintenance and replacement of brushes and other electrical contacts. Brushless motors do not make noises from friction or sparking, in contrast to typical DC motors that utilize brushes to function. This is a remarkably interesting feature of brushless motors because it allows them to be suitable for use in environments where sparks are dangerous.

For example, in environments where ignitable gases are present. This type of motor is a rotor with permanent magnets. It could not be otherwise since there are no contacts on the rotor. Its stator has electromagnets excited by windings and coils. This is a significant difference compared to the traditional brush is DC motor since the latter can have both stator and rotor, composed of electromagnets to increase the magnetic field, and therefore have a higher mechanical power density by unit of weight of the motor. The torque of

the brushless motor is due to the magnetic interaction imposed by the electromagnet on the permanent magnets. The electromagnet has a specific sequence of excitation that forces the permanent magnets to move in an angular direction that is always in the same direction and is as constant as it can be under the same excitation. Obviously, the sequence is cyclical, and the rotor continues its rotation until it's powered, repeating the sequence from time to time. To be more specific, rotation of the electromagnet inside the stater is such that the magnets of the rotor are attracted to the electromagnet, without ever reaching them, as the excitation is always, and only of those coils not yet reached.

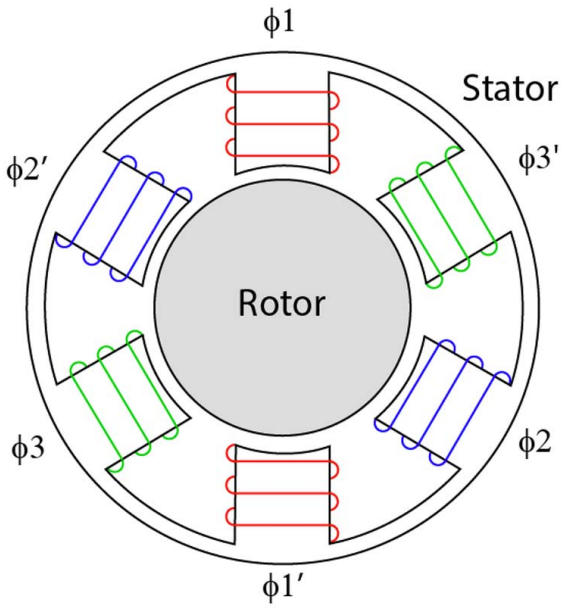

Figure 3.8: A three-phase BLDC motor. The stator has three separate coils that are displaced from one another by 120 electrical degrees. (Allied Motion, 2022).

Figure 3.8 above shows how the coils are positioned on a BDLC motor. When electricity flows into coil one, the opposite poles of the rotor and stator are attracted to each other. When the rotor is getting close to coil one, the electricity flows in coil two. When the rotor is getting close to coil two, the electricity flows in coil three. Later, in the coil one, the electricity will flow again, but with opposite polarity. This process is repeated continuously inside the engine, ensuring a constant rotation of the rotor. In the case of more coils, always multiples of two, it makes sense to use a sequence that uses them all to increase the magnetic flux to which the permanent magnets are subjected and turning off only the opposite electromagnet. This simple foresight increases the torque, that is the power that the motor can supply without modifying its design criteria. The greater the number of coils the smoother is the rotation of the rotor, which will just have a constant

torque.

To know the precise movement in which to excite the correct sequence of electromagnet which will allow continuity of the rotary motion for this purpose, an electronic encoder/-controller is typically used. This controller features a sensor that can determine the precise location of the rotor magnets, allowing it to choose which coil to activate by adhering to the proper excitation sequence. The modern brushless motor sensors to use the hall effect. In the figure below we can observe the configuration of a standard sensor.

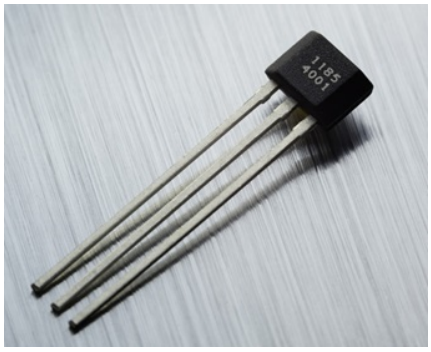

Figure 3.9: 3-pin Hall sensor (MLX92232).

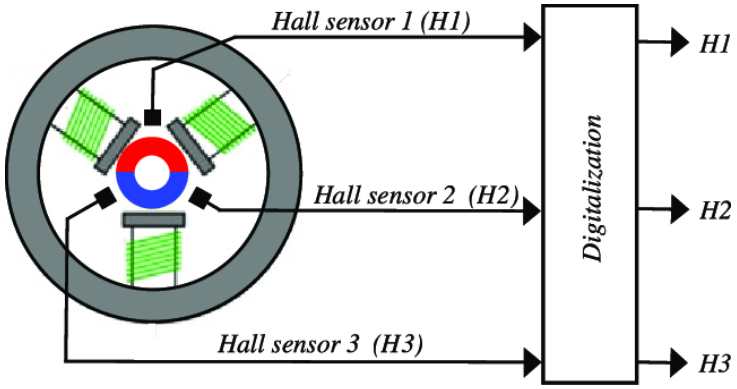

Figure 3.10: Block diagram of Hall-effect of hall-sensor and digitalization (Fernandez et al., 2016).

This is a simple to use device. It can drive two or four or eight or more electromagnets. We can summarize the main strength points of the brushless DC motor brushless motor as noiseless, reliable and has an exceptionally low MTBF parameter, which is the predicted elapsed time between its inherent failures. It provides low energy costs and does not require any maintenance. It has a more competitive price than traditional brush motors of the same power. However, the brushless motor also requires an electronic driver to

control the right excitation sequence using permanent magnets. The specific power is lower than the conventional motors, equipped with stator and rotor.

We intended to have full control of the wheels' speed and in this case brushless direct current (DC) motors were the most suitable for the system. On average, Brushless DC motors typically have a 20–30% higher efficiency than brushed models. This is because they use magnets rather than brushes to produce the desired effects. The appropriate motor choice was based on the device's desired speed and the max load the powered wheels would support.

3.3 Hardware Design

Figure 3.11: Prototype version of the Ninebot S with attached jumping mechanism.

This section reviews the main subsystem that allows the robot to jump with a small form factor, suitable for indoor and outdoor environments. Structural components were created with the drawbacks of increasing mass in mind. For this reason, a lightweight aluminum linkage with 3-D printed nylon components was chosen. To prove this concept, a real-world prototype was developed to demonstrate jumping and stabilized driving in

multiple experiments.

The proposed system consists of a three and four bar-style linkage attached to the original chassis of the Segway Ninebot. This linkage design allows for vertical linear motion, keeping a counter mass perpendicular to the ground as shown in Figure 3.11. All mounting hardware was designed to attach directly to where the knee control steering bar was originally located. No additional machining or modification, apart from some disassembly, is needed to attach the jumping mechanism to chassis of the robot.

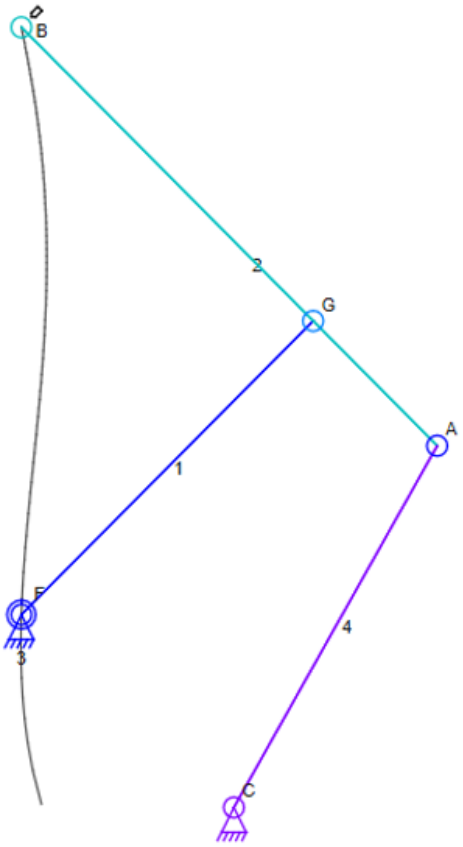

Figure 3.12: Simplified three-bar linkage.

Figure 3.12 above shows the simplified version of our final linkage. Joint C and F are mounted to the chassis of the Segway, and point B is mounted to the counter mass. In order to keep the counter mass parallel to the chassis ground, a four-bar linkage was built into the design. The complete linkage design can be shown in Figure 3.13 below. It is important to keep the counter mass parallel to the chassis in order to minimize the number of external forces on the Segway. In our experience with the NXT prototype, extending the linkage shown in Figure 3.12 results in the Segway rotating backwards mid jump.

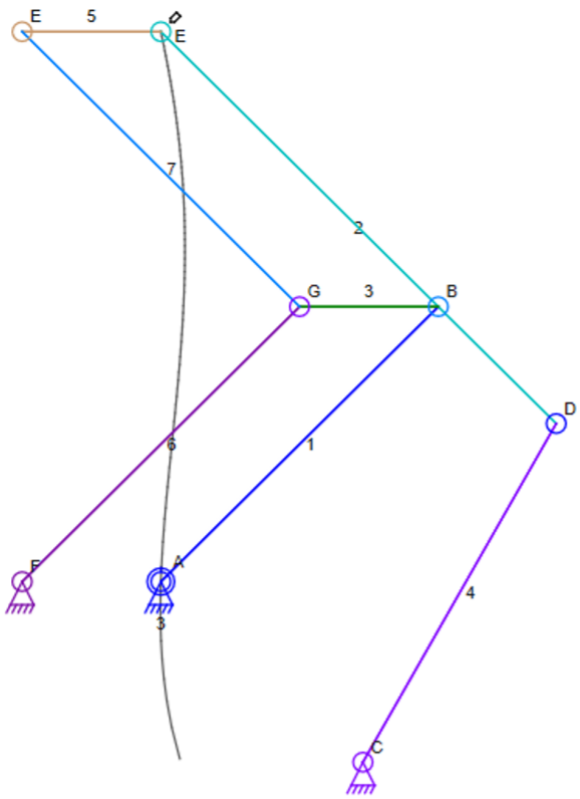

Figure 3.13: Full linkage with additional double-reverse four bars. Near-linear vertical motion is represented with a black line.

To provide smooth vertical movement, a brushed DC motor was selected to drive the linkage from the hip joint. Torsion springs were installed at joint D to counteract the system's own weight and reduce the strain put on the motor. This means when the system is idle, little effort is required to extend the linkage upwards. The gearbox of the linkage prevents the system from moving on its own.

Following are the list of components used in the making of the Segway's jumping mechanism:

1. AmpFlow 12V Brushed Motor
2. Qunqi L298N Motor Drive Controller
3. Arduino UNO REV3
4. Micro Limit Switches
5. 12V Lipo-Battery

6. 8x19x6mm Rubber Sealed Precision Ball Bearings
7. 3/8" Bore Stainless Steel Axle Shaft Collars
8. ABN Compression & Extension Springs
9. T651 Aluminum
10. A36 Mild Carbon Steel alloy

Linkage components were designed in Solidworks and manufactured on a variety of CNC and waterjet machines. During assembly, ball bearings were press-fit into the links and held into place with stainless steel shaft collars. In order to balance weight with strength, linkage components were cut from ¼ inch aluminum sheets. Additional components such as spacers were 3D printed out of PA6 nylon filament.

The design of our jumping mechanism was developed through careful analysis of physics and robot kinematics. Our method of jumping consists of rapidly accelerating a counter mass to a point where it rapidly decelerates. This creates an upwards force that allows the chassis to lift off the ground and achieve weightlessness. Put simply, to achieve weightlessness at a given trajectory, the upwards force created by the rapid deceleration of the extending linkage should cancel out with the downwards force of gravity on the entire system. While this won't create enough force to allow the Segway to jump, it is the first step in solving for the forces needed.

The downwards force of gravity on the entire system can be represented as:

$$F = mg = (11.4\ kg)(9.8\ m/s^2) = 111.72\ N$$

The upwards force created when the linkage extends should equal the downwards force of gravity at any given moment

$$111.72\ N = (m_{\text{counterweight}})(\Delta V/\Delta t)$$

To better describe the motion of the robot, we defined the reference frame as the point of full extension of the linkage. Since this is the moment the limit switches disengage the motors and the counterweight velocity becomes 0, the Segway should become weightless for a given period.

In addition to the kinematic equations, we needed to define the constraints on the robot. More specifically, we needed to identify the mass of both the chassis and the counterweight

as well as the forces acting upon it. The link AB connected to the hip motor experiences forces proportional to the Segway's mass counteracting the torque provided by the motor. In order to drive this linkage, we need to identify the moment of inertia of the link and its required angular acceleration.

The moment if inertia for the link comes out to be

$$I = 9.16^2 * 4.2\,\text{kg} = 352.40\,\text{kgm}^2$$

If Torque is equal to the moment of inertia multiplied by the angular acceleration of link, BE, the 0.66-0.32 horsepower motor chosen for this application should produce the approximate 1450 ft lb. of torque needed. The battery chosen to power this motor was a 5200 mAh LiPo battery. This gave us approximately 1/4 of an hour of continuous testing between charges.

In order to control the hip motor of the linkage, we needed a few extra components. The Arduino uno was chosen as the dedicated microcontroller for the jumping mechanism. An Arduino can easily power a small electronic device such as an LED, since the 20mA or so from the pins is more than enough. However, since our application requires us to power a large 12V DC motor, we needed a motor controller.

This is where the L298N motor controller comes in. This motor controller was chosen for many reasons, but mostly because it is well known for its simplicity and ease of use in prototyping. While the L298N motor driver can control up to two motors at a time, our application only requires one. Figure 3.14 shows the L298N motor controller with appropriate labels for each input and output.

The L298N board has 12V and 5V terminals. The +12V pin is where the motor power is attached since it can accept voltages ranging from +7V DC to +35V DC. The board also allows a +5V source from the +5V terminal. This allowed us to connect to the Arduino Uno which needed a five-volt source as power. Motor controller A and B inputs connected directly to the microcontroller as well, with the motor terminals connecting to motor terminals three and four. To provide speed control, we fed PWM signals into the motor enable pins. The speed of the motor varied based on the width of the pulses.

The L298N motor driver follows the H-Bridge configuration, which is useful for controlling the direction of the DC motor we used. A schematic of an H-Bridge can be found in Figure 3.15.

In an H-bridge configuration, the motor rotates in the direction set by the switches. When S1 and S4 are on, the left motor terminal is more positive than the right terminal and

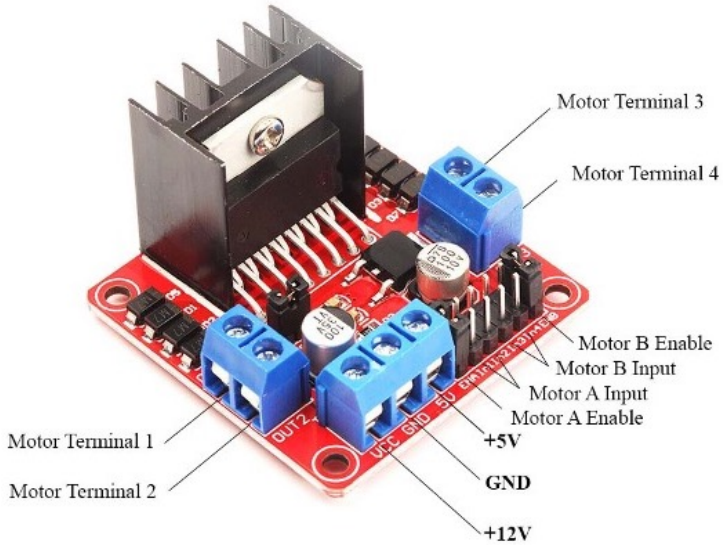

Figure 3.14: L298N Motor Controller.

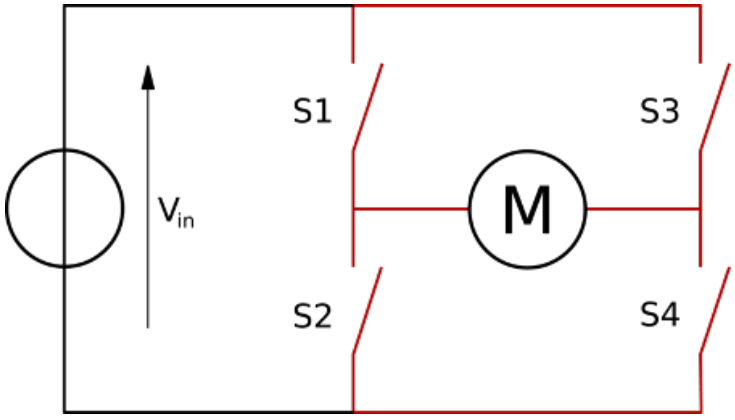

Figure 3.15: H-Bridge Configuration Example.

the motor rotates in a certain direction. To switch direction, S2 and S3 are powered on. This makes the right terminal more positive than the left one.

Another benefit of using an H-Bridge is that you can provide a separate power supply to the motors. This was crucial for our application, since the 5v power source from the Arduino was not nearly enough to power our DC motor.

Figure 3.16 contains the basic wiring diagram for our jumping mechanism. While our linkage required the DC hip motor to run at full speed for a short period of time, this circuit gave us the ability to control the motor's speed. This was done by connecting the motor enable pins to the PWM enabled pins 10 and 5.

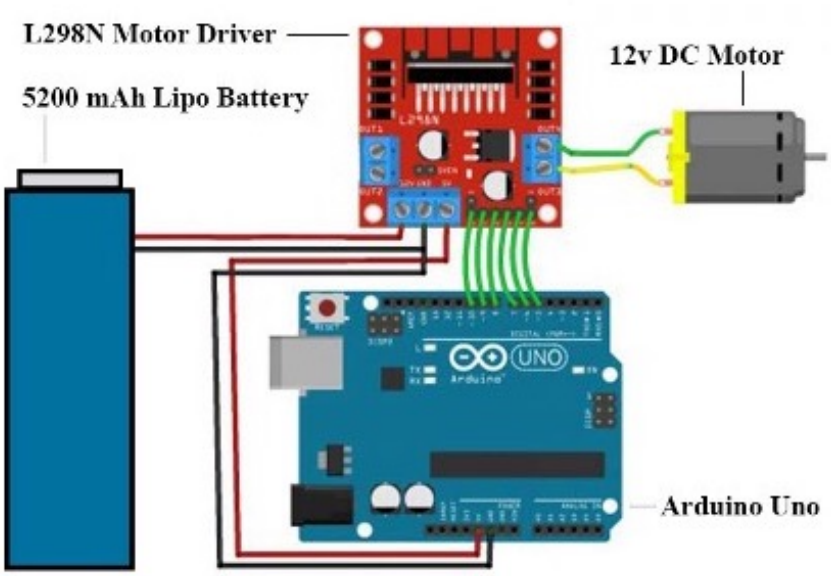

Figure 3.16: Wiring Diagram for Motor Speed Control.

3.4 Self Balancing Control System

3.4.1 Overview

The design of the Segway is based on the model of an inverted pendulum and a cart. The inverted pendulum is a model that has its center of mass above its pivot point (Xiong, 2019), as shown in Figure 3.17.

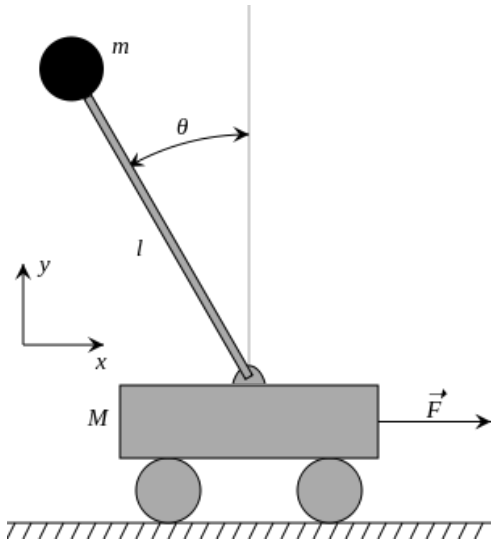

Figure 3.17: A model of an inverted Pendulum on a cart.

For that reason, the body is unstable and without subsidiary help, it topples due to

gravity and inertia force. It is possible to balance the robot by driving it backwards at a certain speed every time it leans forward too much or driving it forward at a certain speed every time it leans forward. However, for it to stabilize and stay in its inverted positions even when in motion, a feedback control system is needed to monitor the tilt angle of the pole and move the pole back under the pivot when it starts to topple. A tangible representation of the inverted pendulum and cart model is the Segway.

3.4.2 Controller Implementation

To achieve a self-balancing system, we implemented a PD controller which is a proportional derivative controller. Following the same concept of the inverted pendulum and a cart model, we balance the robot by driving it in the direction of its of its fall. i.e., if the Segway is leaning forward, we drive the motors forward and the same concept applies if the Segway is leaning backwards. To determine direction of fall, we need to determine the tilt angle θ of the center pole from the zero point as shown in the figure below. For accurate feedback, we also need to determine the speed at which the Segway is falling.

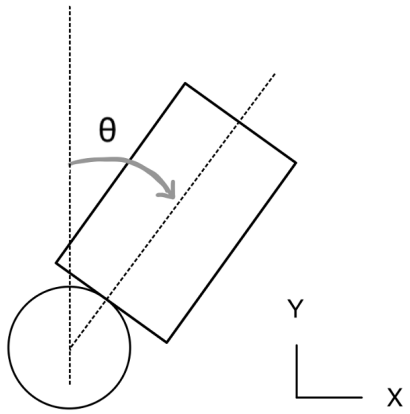

Figure 3.18: Model of Segway tilting forward.

The microcontroller receives an input from the user i.e., the desired tilt angle. This defines the desired standing position of the Segway. It is important to note that although an absolute straight position is desired for a balancing system, it is not easily attainable due to uncertainties of the IMU sensor and inertia due to motion. in this case we had a leeway of $\pm 1^{\circ}$.

3.4.3 Control Scheme

For smooth balancing of Segway, we need a PID controller. Without The PID controller, motion to attain a balancing point would be erratic due to overshooting making it impos-

sible to balance and remain in that position

The overall PID equation is

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

where the error signal received is the input.

Proportional: Rise time and steady state inaccuracy are both decreased by the proportional component. This indicates that the system will achieve its peak value more quickly and that the steady state error will be small when it does so. But it causes the peak overshoot to rise. The proportional component only considers the robot's current angle and causes the motors to move in the same direction as the robot is falling for our self-balancing robot. The robot's motors move more quickly the further it deviates from its aim. When the P-component is employed alone, the robot may initially stabilize, but the system will eventually overshoot, oscillation, and fall.

$$K_p e(t)$$

Derivative: The derivative component shortens the settling time and overshoot. This is through dampening any unwanted oscillations making sure the Segway does not overshoot to the target. Additionally, the system will attain its stable state faster. However, the rising time and the steady state error are unaffected.

$$K_d \frac{de(t)}{dt}$$

Integral: The rising time sped up in this section, and steady state inaccuracy is eliminated. However, it lengthens the settling time and peak overshoot.

$$K_i \int_0^t e(t)dt$$

3.4.4 Software Integration

The First step is to command the ODrive motor driver to set the wheels in motion. The ODrive motor driver comes with pre–installed control modes, namely:

Filtered Position Control – This mode allows you to input a desired position in meters and the motor driver sets the wheels into motion for the given distance. However, in this mode there is minimal control of velocity as the motor controller drives the wheels as hard as it can to raw setpoints which may result in jerky movement. In this case, you can use a second order position filter for smoother motion with the following commands (ODrive Robotics, 2021).

```
axis.controller.config.input_mode = INPUT_MODE_POS_FILTER (activate
   the filter)
```

```
axis.controller.config.input_filter_bandwidth = 2.0 [Hz] (1/2 of your
    setpoint command rate)
```

Trajectory Control – This mode enables you to seamlessly accelerate, coast and decelerate the wheels from one position to another. This mode allows you to input coasting speed, maximum acceleration in turns / sec^2, and maximum deceleration in turns / sec^2. With these inputs, you can adjust the feedback gains more aggressively when using a trajectory to reject disturbance and maintain smooth motion. However, this mode is only available in position control mode. The following are the commands used in this mode:

```
odrv0.axis0.trap_traj.config.vel_limit = <Float>
odrv0.axis0.trap_traj.config.accel_limit = <Float>
odrv0.axis0.trap_traj.config.decel_limit = <Float>
odrv0.axis0.controller.config.inertia = <Float>
```

Controller.config.inertia is a value that links motor torque with acceleration (measured in spins per second). It starts out as 0. It is optional but, if properly tuned, can enhance your system's response.

Circular position control mode – This mode is useful for continuous incremental position movement. For example, a robot rolling indefinitely. In this mode the controller tracks the position of the wheels within only one turn of the motor. Input pos is automatically wrapped around into the appropriate value if it is increased outside of this range (for example, via step or dir input). The following command is used in this mode.

```
odrv0.axis0.controller.config.circular_setpoints_range = <N>
```

Velocity control mode – this mode allows you drive the wheels by inputting a constant speed. The following commands are used in this mode.

```
odrv0.axis0.controller.config.control_mode =
   CONTROL_MODE_VELOCITY_CONTROL (sets the control mode)
odrv0.axis0.controller.input_vel = <turn/s>
```

Torque control mode – this mode allows you to set the wheels under a constant torque. To set this mode the following commands are used.

```
odrv0.axis0.controller.config.control_mode =
   CONTROL_MODE_TORQUE_CONTROL
odrv0.axis0.motor.config.torque_constant = 8.23 / [rpm / V]
```

In our Segway, we aim to have full control of the wheel's velocity. Therefore, the only modes suitable for this are Torque Control and Velocity Control. Both modes work superbly in controlling the wheel motion. However, we plan to implement an IMU for self-balancing of the Segway and in this case the most suitable control mode is Velocity Control. Velocity control mode sets the wheels at a constant speed. It is good to note that this mode assumes an ideal environment free from inertia forces and friction. This means that if we drove the Segway in a normal environment, it would never be able to attain the given set velocity due to force of inertia and friction between the wheels and the floor. For that reason, we created a PI Controller for our velocity control mode. The PI controller is as shown below. The closed-loop diagram showing the PD control process is shown in Figure 3.20.

```
vel_error = vel_cmd - vel_feedback,
current_integral += vel_error * vel_integrator gain,
current_cmd = vel_error * vel_gain + current_integral +
   current_feedforward.
```

where $e(t) = vel\_error$, $Kp = vel\_integrator$ gain and $Ki = current\_integral$.

Now that we have the motors in motion, our next step is to balance the Segway. As shown in Figure 3.19, the jumping mechanism fixated on the chassis raises the center of gravity above the pivot making the system unstable. Due to that, we need an algorithm to counter the inverse forces, i.e., Inertia. This is through consolidating the IMU data with the velocity controller. The BNO055 IMU gives a wide range of data but for this project we only need the Euler angle i.e., the tilt angle to determine position of center rod and the gyroscope reading (rad/sec) i.e., the angular velocity. This data is easily obtainable by calling the following command in python

```
adafruit_bno055.BNO055_I2C(i2c).format(sensor.euler[2])          ...
   for Euler angle
adafruit_bno055.BNO055_I2C(i2c).format(sensor.gyro[0])           ...
   for gyroscope reading
```

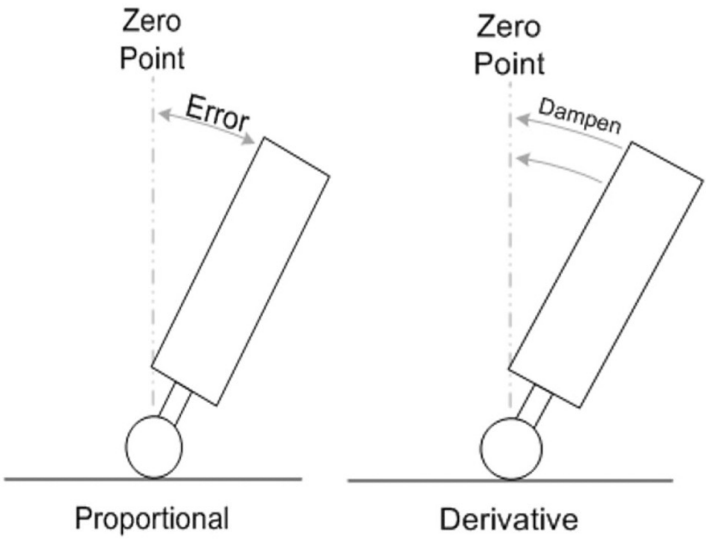

Figure 3.19: Model of Segway showing error in unbalanced position.

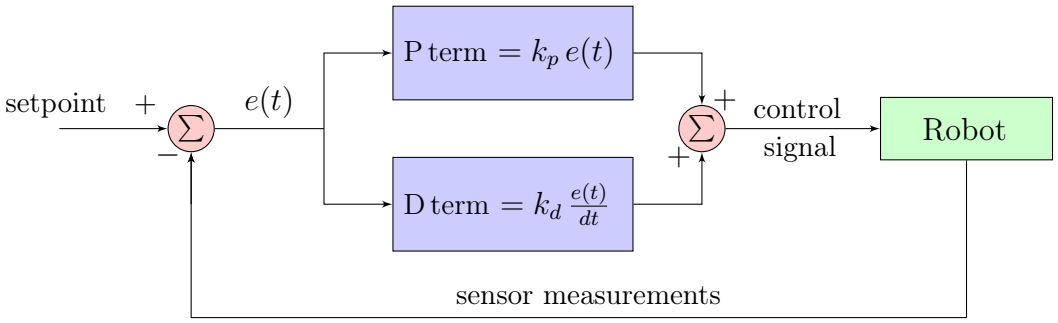

Figure 3.20: Closed-loop diagram showing the PD control process.

The next is to create a function that checks for the position of the Segway, balances it, and drives the motors at the desired velocity. Our self-balancing controller is straightforward. It is based on two parameters:

1. `KP_BALANCE`: Determines how much the robot reacts to leaning forward or backward (CharlestonChIMP, 2022). A high value might lead to tilt oscillations. This value is multiplied with tilt angle of the Segway i.e., Euler Angle

2. `KD_BALANCE`: Determines how much the robot reacts to the rate of tilt change. ("ChIMP: This IS the Droid You've Been Looking For!") It generally dampens oscillations, but a value too high might lead to increased oscillations (mainly due to time lags in the control loop). This value is multiplied with the gyro reading of the IMU.i.e., the angular velocity of the center rod.

By summing up these two values and multiplying them with a desired speed we zero the tilt error balancing the Segway. Once the Segway is balanced, `tilt_limit_exceeded` is set to False allowing the Segway to continue its drive at the given velocity. For continuous autonomous self-balancing we created a loop within the function that constantly checks if the tilt angle is exceeded or not. Anytime the Segway exceeds this tilt angle, the balancing controller is called, setting the Segway back to balanced position. The Block diagram in Figure 3.21 shows how the whole control system is integrated.

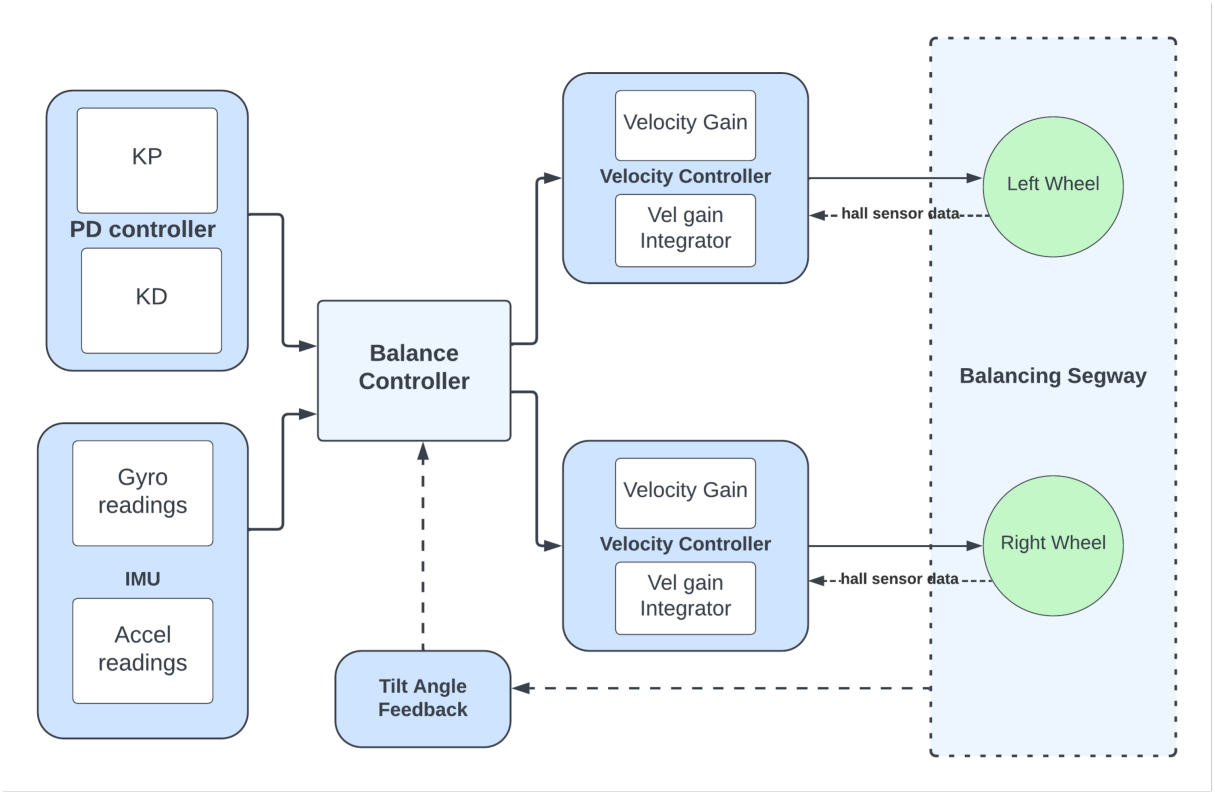

Figure 3.21: Block diagram of the integrated control system.

Chapter 4

SYSTEM TESTING AND VALIDATION

On completion of our custom Segway design, we tested the design's sturdiness and important performance indicators in relation to the original design objectives. Since the project was not completed in harmony, we conducted testing focusing on two main sections: Self balancing mechanism and Jumping mechanism.

4.1 Self-balancing Mechanism Testing

In this section, our focus was on the Segway's autonomous mobility. I.e., its ability to maneuver autonomously in response to changes in slope, speed, and presence of obstacles. To begin with, we calibrated the wheels following instructions given by ODrive Robotics then tested for the response of the wheels to varying current and braking force by clamping the wheel on a table (as shown in Figure 4.1), observing, and documenting the wheels behavior. The wheel was connected to an ODrive motor driver powered by a regulated DC power supply. We used a Raspberry Pi controller simulator on a laptop in place of the Raspberry Pi computer. The main purpose of the Raspberry Pi simulator was to send commands and read data from the ODrive motor driver.

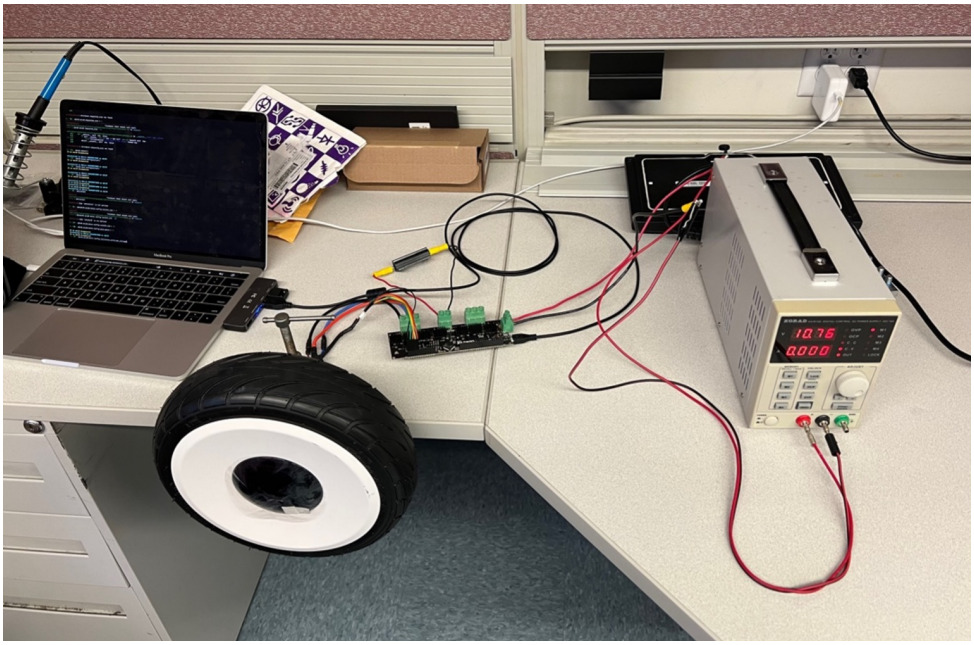

Figure 4.1: Wheel test. Left Segway wheel clamped on a table.

After testing for the wheels behavior to varying power, our next step was to test for the IMU. BNO055 absolute sensor is an open source, and it provides code to calibrate the IMU using the `.getCalibration` function in the `Adafruit_BNO055` library. After running the code, a sensor demo WebGI model, as shown in Figure 4.2, is displayed showing the real-time position and orientation of the IMU in a 3D environment. Our focus on this testing was to determine the orientation of the IMU and its sensitivity to force. After calibrating and testing the IMU response to force, we created a function to drive the wheels depending on the orientation of the IMU and tested it.

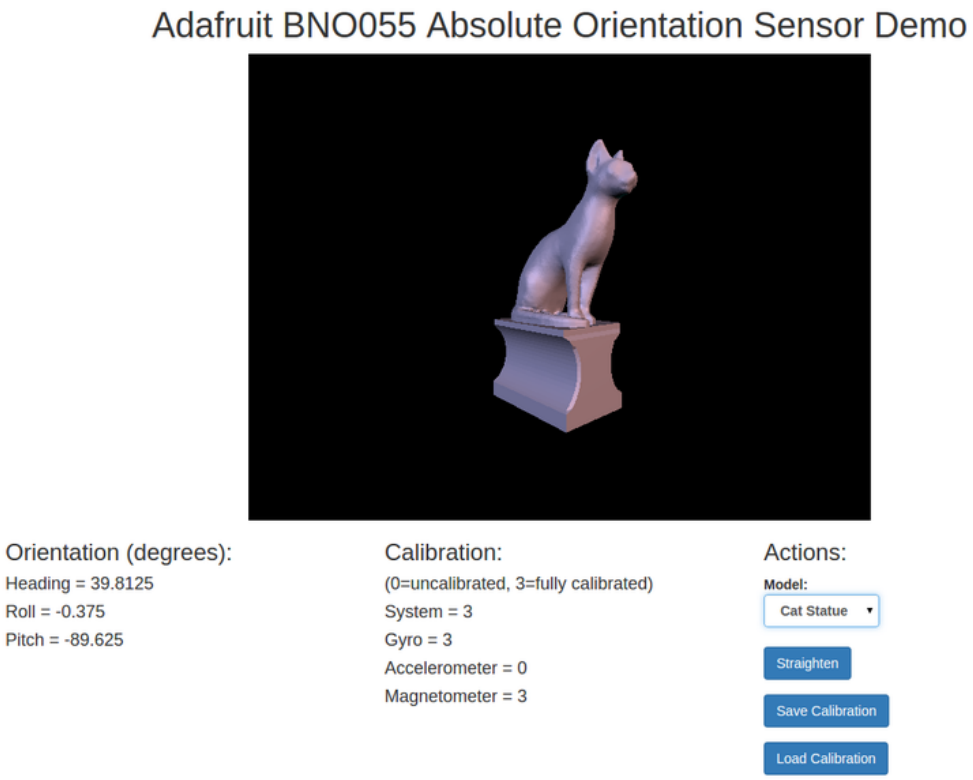

Figure 4.2: WebGl model used for BNO055 calibration.

After testing and determining the functionality of both the wheels and IMU our final step was to assemble the wheels and the IMU on the chassis and then test for Self-balancing. This test was carried out at the WPI robotics laboratory (closed environment with a flat floor surface). This was the most extensive part of testing as we had to do was testing and adjusting our PI controller values determining the most stable and responsive values.

To begin with we tested our function `drive_with_IMU`. The main purpose for carrying this test is to first check if the IMU is properly installed and aligned with the chassis by

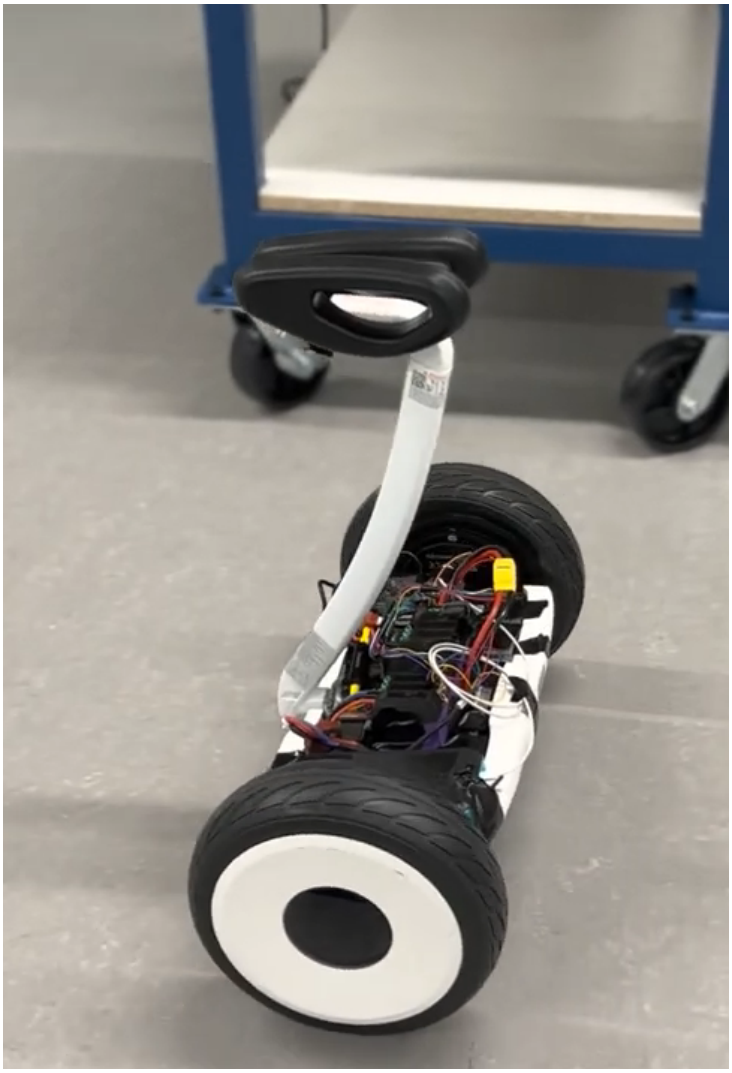

Figure 4.3: The Segway balancing during testing.

ensuring a 90-degree roll angle when the Segway is balancing at zero point. The function `drive_with_IMU` is to ensure the wheels and the IMU sensor are synchronized. That is, as seen in Figure 23, the Segway drives forward when tilted forward and in the same way it drives in reverse when tilted backwards and stops at zero point. In this test, the Segway was quite responsive to the direction of tilt. However, some responses were delayed and for that reason we decided to include a PID control to speed up the motor reaction to change in tilt. To avoid the Segway overshooting, this PID controller was limited to a roll angle between 60 and 120 degrees. This test was a success, and the Segway was consistently responsive.

After confirming that IMU and the wheels were responsive we carried out our last test which was to integrate our main function to autonomously balance the Segway. This test

was carried out on a flat surface at the WPI robotics laboratory. We carried out test runs adjusting our KI values optimizing the Segway response to tilt angle feedback. This test was successful, and the Segway was able to Self-balance itself autonomously and drive in a straight line, as shown in Figure 4.3.

We also tested the Segway on a rough terrain outside the WPI robotics laboratory. In this test, the Segway was also able to autonomously self-balance. However, this test was limited due to speed of our IMU sensor and the Segway to was not capable of self-balancing in presence of obstacles such as gravel and other materials on a rough terrain.

A video of our testing can be found at the following YouTube link:
`https://youtu.be/l_QTfdDgOqA`

4.2 Jumping Mechanism Testing

Throughout our prototyping phase, many smaller tests were run to study the feasibility of this goal. We developed a half-scale version of the robot and linkage in both Lego NXT and VEX which saw promising results. Our Lego NXT prototype can be seen in Figure 4.4. It is a 1 to 3 scale mockup of our final protype. Weight proportions were made equal to that in the original Segway.

Once the final mechanism was developed, our team was faced with the difficult problem that the added mass in addition to the wheel mass made it difficult to jump past a few inches. In parallel with redesigning certain components to reduce the weight of the system, we began testing the mechanism attached to the chassis without the wheels. We began testing with a counter mass less than that of the system weight. This is because of the additional components such as the motor and links.

To test our system, we mounted a ruler to the wall adjacent to the Segway and set up a cellphone with slow motion capture to record a rough trajectory of the robot. A crash pad was placed below the robot to prevent damage to the chassis. Since the wheels make up most of the system's mass, we saw success with the jumping mechanism achieving close to our goal jump height of 7 1/2 inches.

In order to test the system, we programmed a simple Arduino sketch in the built-in IDE. This sketch uses a limit switch to turn off the motors when the linkage is at full extension. The code block below is an example of the code used to test the height of the robot's jump at 3/4 max speed.

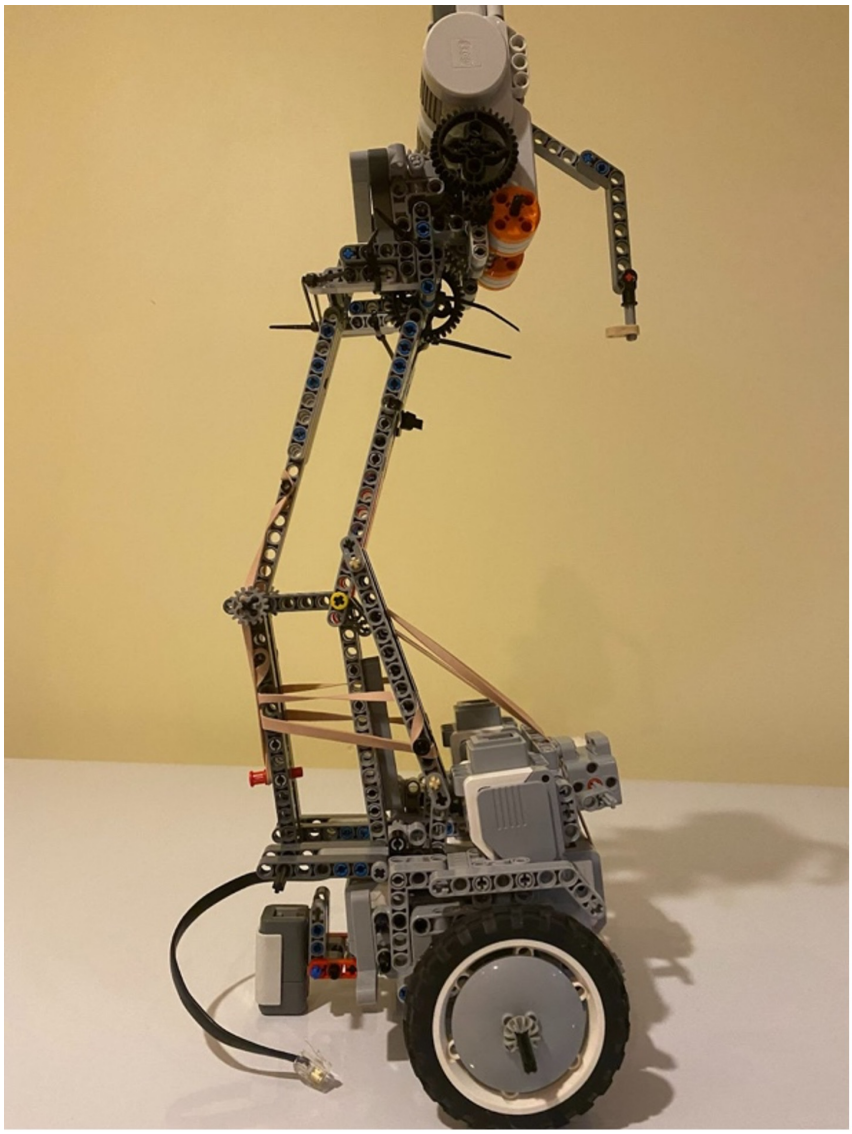

Figure 4.4: Lego NXT prototype.

```
#include <ezButton.h> //limit switch control

//Motor Connections
#define EnA 10
#define EnB 5
#define In1 9
#define In2 8

ezButton limitSwitch(7);
```

```
void setup()
{
    Serial.begin(9600);
    limitSwitch.setDebounceTime(50); // set debounce 50 milliseconds

    // Motor Controls set as outputs
    pinMode(EnA, OUTPUT);
    pinMode(EnB, OUTPUT);
    pinMode(In1, OUTPUT);
    pinMode(In2, OUTPUT);
}

void extendLinkage()
{
    // turn on motor
    digitalWrite(In1, HIGH);
    digitalWrite(In2, LOW);
    // set speed
    analogWrite(EnA, 200);
    delay(2500);
}

void loop()
{
    limitSwitch.loop();
    extendLinkage();
    if(limitSwitch.isPressed())
    // Turn off motor
    digitalWrite(In1, LOW);
    digitalWrite(In2, LOW);
    delay(100);
}
```

Chapter 5

DISCUSSIONS

5.1 Stabilization and Control Performance

On testing the motor, we determined a stable driving speed of 1 spin/s to 4 spins/s. Any speed beyond 4 spins/s led to vibration of the motors and the constant speed was not easily attainable. It is also good to note that the higher the speed of the motor the harder it is to brake. It was easy to break the wheels at speed below 3 spins/s without overheating the 2-ohm braking resistor. Also, overpowering the motor leads to overheating which could result in destroying the heat sensitive hall sensors/reducing their effectiveness. The IMU gave consistent reading in all three axes, (roll yaw and pitch). However, sudden impacts on the IMU caused sudden jumps of around 20 to 90 degrees and resulted in a hard to recover drift due to accumulation of errors. When this happens, the IMU becomes less reliable making it difficult to balance the Segway, unless it is recalibrated. To ease the process of recalibration, we created a function that calibrates both the IMU and the motors every time we run the main program.

We were successful in autonomously balancing and driving the Segway in a straight line, with a tilt angle of $\pm 5°$. Due to mechanical uncertainties of the IMU, it was strenuous to attain a perfect zero point in some tests. We also did not achieve a 100 percent success rate in balancing the Segway robot, as the IMUs were affected by noise. The Segway robot was only able to self-balancing when starting at low speeds of around 1 spin/s. Higher motor speed only resulted in rapid jerk of the center pole. A probable reason for this is the IMU was not fast enough to counter the force of inertia during start. The BNO055 IMU is limited to 100 Hz frequency which makes it hard to detect sudden vibrations. For better understanding, the shorter the execution time the better it is for the control algorithm. At 5ms loop time the Segway is quite jerky and as you increase the loop time the system gets stable. We were able to stabilize the Segway at around 10 ms loop time. However, although we stabilized our system with this loop time, we were limited, as this is the furthest loop time possible to achieve with the BNO055. Having a faster IMU would be ideal to achieve smoother stabilization and a faster reaction. A list of substitutes for this IMU is included in Appendix B.

We tested the Segway on a rougher terrain to determine the effectiveness of the self-balancing mechanism. However, due to the BNO055 IMU limitations, we were not suc-

cessful in balancing the Segway except for a few seconds.

5.2 Jumping Mechanism Performance

While testing the hip motor, we observed the added weight of the linkage, and the motor exceeded our expectations. While designing the linkage, our initial calculations were based on a much lighter mechanism weight. This proved to have a significant effect on the initial jump height of the robot. Even at full power, the Segway was only capable of making it an inch or so off the ground.

We took steps to lighten the linkage and redesign the Segway to move more of the mass to the counterweight with no success. Our calculations suggested that increasing the counterweight mass may also help with increasing the jump height, however any attempt to do this only resulted in increased strain on the single motor we were using. In one experiment, our efforts to increase the counterweight mass resulted in the fracturing of the main link in the assembly.

We concluded that a major reworking of the mechanism design was necessary to achieve our jump height goal. Under the limitations of the time left in our project, we decided to test the jumping mechanism without the wheels of the Segway attached. While this was not our goal, it allowed us to collect enough data to make observations and suggestions for future designs. These tests were achieved by mounting a ruler to the wall adjacent to the Segway and setting up a cellphone with slow motion capture to record a rough trajectory of the robot. At full power, the Segway was recorded jumping a maximum of $6\,1/2$ inches. While this was very close to our optimal jump height, we were not able to achieve these results with the wheel motors attached to the chassis.

In conclusion, we saw limited success with the full-scale prototype. The combination of the excess wheel mass and the weight of the linkage required to make the robot jump drastically decreased our achievable jump height. With our jump height goal at the start of the project set at $7\,1/2$ inches, we we were underwhelmed its ability to achieve $6\,1/2$ inches without the wheel mass. After much reflection, our team believes the original calculation included a much lighter linkage mass. Through the many revisions and additions to the jumping mechanism linkage, the mass was increased so much that it began to affect the performance of the system. Another observation made was our inability to add more mass to the counterweight. The brushed motor chosen could not provide enough torque to drive an increased counterweight mass upwards. For this reason, we were forced to keep the counter mass at approximately three kilograms. Both design flaws explain why the wheels had to be removed to see any conclusive results.

For future iterations of this design, we would recommend a complete redesign of the motor mechanism that drives the linkage. While we are confident that this linkage can achieve a jump height of 7 1/2 inches, the motor and gearbox add too much weight to the chassis. This could be fixed by driving the linkage from the top, which would require a redesign of the motor mount and gearbox.

Another modification that would greatly increase the robot's jump height is the replacement of the wheel motors. A breakdown of the Segway Ninebot S' mass is as follows:

- Overall: $\sim$ 12.8 kg
- Battery: 1.98629 kg
- Wheels: 7.81038 kg (3.90519 kg each)
- Chassis: 3.40694 kg

For normal applications, the wheel motors that come with the Segway are necessary to move a person standing on the robot. Since our application does not need to support a human, a much lighter, more mobile wheel could be substituted. In the timeframe of this project, our team was not able to find or design lighter wheels for the Segway.

Chapter 6

CONCLUSIONS

This project was a success in that it met almost all our design requirements and, in many ways, even exceeded them. Additionally, to confirm their effectiveness, all parts – mechanical and electrical – were thoroughly evaluated as a whole system in actual field-testing settings. Except for the project going slightly over budget and the project scope being trimmed to create more attainable software goals within the timeframe, preliminary estimations for the project's basic scope, budget, and timeline were closely followed.

The achievement of self-balancing in our custom Segway proves that it is possible to achieve autonomous balancing using locally available components. Although the Segway did not have a 100 percent success rate in self - balancing, the few times it balanced and drove in a straight line prove that with better components i.e., better/faster IMU sensors, it is possible to achieve a fully autonomous Segway robot. The second-best solution to achieving fully autonomous self-balancing robot would be integration of a function to filter out the IMU noise caused by sudden impact. A possible filter algorithm as suggested in (Capriglione et al., 2021) would be a Kalman Filter. Due to the limited time, we were not able to integrate or test this filter in our system. However, we are confident that the addition of this filter to the overall control system would aid in better performance for the Segway.

Through testing of our jumping mechanism, we concluded that the current state of our design can't reach our jump goal height of 7 1/2 inches. From the data we were able to collect during the tests we ran, we do believe that our goals are achievable through a slight redesign of the hip motor assembly and replacement wheel motors. As mentioned in the jumping mechanism performance section of this report, the added weight of the jumping mechanism and the existing weight of the wheel motors proved too much for the hip motor of our linkage to handle. With that being said, we believe that our project has laid the groundwork for future designs of this concept.

Focusing on logistics, this capstone project involved designing, fabricating, wiring, assembling, integrating, field testing, and software development for a Hopping Segway robot. Since this project was the work of three college students, it necessitated careful management of important dates, progress, and financial considerations. Overall, the project largely adhered to the suggested plan and time frame. As the project moved along, one

shift was the decrease in software targets. Initially, ROS simulation and control was intended to be used. However, it was ultimately decided to be outside of scope for the project due to extra effort needed on motor control software. All other software objectives were accomplished.

Due to the Covid-19 pandemic and a worldwide shortage of chips, there was delay in shipping time and therefore, Manufacturing began significantly late than anticipated, going against the project timetable, since we anticipated that manufacturing would take more time and effort due to the overall design's complexity increasing (machining, welding, and assembly). By beginning the production process as soon as possible, the project could be finished on schedule. Due to unanticipated delays in the production process, we were around three weeks behind schedule at the beginning of D-term. This resulted in a little less time for field testing, but we pushed back our documentation timeline, so there was still enough time to validate and thoroughly test the Segway.

Overall, this was a successful project, allowing us to apply various skills learnt in class such as electrical design and mechanical design, software engineering, and critical thinking skills on a real time Segway system.

This project would not have been possible without the knowledge acquired from our undergraduate experiences at WPI. Courses such as Unified Robotics I through IV gave us the hands-on experience needed to design a linkage, implement a PD controller for self-balancing, and begin simulating the Segway in ROS and Gazebo. Our team was able to develop practical skills such as time management and organizing a large project through other group projects like our IQP's and graduate-level courses. Without the research and problem-solving experience, we obtained through conducting literature reviews and data analysis in our time here at WPI, we would not have been as successful. All these experiences at WPI were essential to making this project a success and we are thankful for our professors and faculty help along the way.

6.1 Recommendations and Future Works

To begin with, we recommend use of higher level or Industrial level IMUs for this experiment. Self-Balancing was a major pillar in this project as the Jumping mechanism depends on it. Reaction speed of the sensor plays a huge role in self balancing the robot and IMUs with frequency greater than 100 HZ would be recommended. An example of this type of IMU is the Vector Nav – 100 IMU.

We also recommend integration of a Kalman filter algorithm (Kim and Bang, 2018) for

more effective self-balancing. Although adding a filter algorithm would delay the realization of the main function, it is possible to better estimate the state of the system, leading to higher success rates in self-balancing the Segway and reducing overload on the motors.

Implementing additional sensors would also be highly recommended especially as a possible solution to avoiding obstacles. Lidar Sensors would be a good fit in this experiment. In the beginning of this project, we aimed to implement ROS simulation of the Segway System but due to limited time we were not able to accomplish this goal. Implementing a ROS system is highly recommended in future especially as an alternate testing and control method of the Segway. ROS Simulation would also benefit demonstrations in circumstances where the physical model is not applicable. The Odrive motor driver used in this project is ROS enabled and it would greatly benefit from this implementation.

In designing a jumping mechanism from scratch for the Segway, we made a few observations that could help future work on this style of robot. First, before designing a jumping mechanism, as much weight as possible should be removed from the robot. When modifying an existing product like the Ninebot S for this application, many existing components are no longer needed. For example, the built-in wheel motors could be replaced with a lightweight and mobile design. Since we no longer needed to move the weight of a 220 lb. Human, the overly powerful wheels end up just harming the effectiveness of the jumping mechanism.

Additionally, we would recommend that future teams complete a stress-analysis of the links in the mechanism before prototyping. With our time limitations, we were not able to analyze the effects the forces had on individual links. This led to one of our linkage components fracturing during testing which set us back.

We believe that with a few modifications, our design can achieve a jump height of approximately 7 1/2 inches. However, the changes required to make this a reality bring into question whether it would be easier to design a Segway from scratch instead. Since we redesigned the self-balancing system and would have to replace the wheels, it may be more cost and time effective to start from scratch as opposed to modifying the Segway Ninebot S.

BIBLIOGRAPHY

Adafruit (2022). *Adafruit BNO055 Absolute Orientation Sensor*. URL: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/ (visited on 12/15/2022).

Adafruit Industries (2022). *Adafruit Unified BNO055 Driver (AHRS/Orientation)*. URL: https://github.com/adafruit/Adafruit_BNO055 (visited on 12/15/2022).

Allied Motion, Inc. (2022). *Choosing Between Brush and Brushless DC Motors*. URL: https://www.alliedmotion.com/choosing-between-brush-and-brushless-dc-motors/ (visited on 12/15/2022).

Boubaker, Olfa (2013). "The Inverted Pendulum Benchmark in Nonlinear Control Theory: A Survey". In: *International Journal of Advanced Robotic Systems* 10.5, p. 233. DOI: 10.5772/55058. eprint: https://doi.org/10.5772/55058. URL: https://doi.org/10.5772/55058.

Capriglione, Domenico et al. (2021). "Experimental Analysis of Filtering Algorithms for IMU-Based Applications Under Vibrations". In: *IEEE Transactions on Instrumentation and Measurement* 70, pp. 1–10. DOI: 10.1109/TIM.2020.3044339.

CharlestonChIMP (2022). *ChIMP: This IS the Droid You've Been Looking For!* URL: https://www.instructables.com/ChIMP-This-IS-the-Droid-Youve-Been-Looking-For/ (visited on 12/15/2022).

Fernandez, Daniel et al. (2016). "Permanent magnet temperature estimation in PM synchronous motors using low cost hall effect sensors". In: *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, pp. 1–8. DOI: 10.1109/ECCE.2016.7855349.

Kim, Youngjoo and Hyochoong Bang (2018). "Introduction to Kalman Filter and Its Applications". In: *Introduction and Implementations of the Kalman Filter*. Ed. by Felix Govaers. Rijeka: IntechOpen. Chap. 2. DOI: 10.5772/intechopen.80600. URL: https://doi.org/10.5772/intechopen.80600.

Klemm, Victor et al. (2019). "Ascento: A Two-Wheeled Jumping Robot". In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 7515–7521. DOI: 10.1109/ICRA.2019.8793792.

ODrive Robotics (2021). *Control Modes*. URL: https://docs.odriverobotics.com/v/0.5.4/control-modes.html (visited on 12/15/2022).

Segway Inc (2022). *Ninebot S*. URL: https://www.segway.com/ninebot-s/ (visited on 12/15/2022).

Vovyo Technology (2021). *Brushless Motor VS Brushed Motor*. URL: https://www.vovyopump.com/brushless-motor-vs-brushed-motor/ (visited on 12/18/2022).

Wikipedia (2022a). *Inverted pendulum*. URL: https://en.wikipedia.org/wiki/Inverted_pendulum (visited on 12/15/2022).

Wikipedia (2022b). *Segway*. URL: https://en.wikipedia.org/wiki/Segway (visited on 12/15/2022).

Xiong, Wenjing (2019). *Inverted Pendulum*. URL: https://weijiang-xiong.github.io/project/inverted-pendulum/ (visited on 12/15/2022).

Appendix A

DESIGN SPECIFICATION SHEET

Category	Item	N3M240 Parameters
Dimensions	Length × width	260×548 (mm)
	Height[1]	595 (mm)
Weight	Payload	88~220 lbs (40~100 kg)
	Net weight	Approx. 28 lbs (12.8 kg)
Rider Requirements	Ages	16 ~ 50 years
	Height	4'3"–6'6" (130~200 cm)
Vehicle	Maximum speed	Approx. 10 mph (16 km/h)
	Typical range[2]	Approx. 13.7 mi (22 km)
	Maximum slope	Approx. 15°
	Beginner mode	Will be automatically disabled after you have traveled 1 km. You can re-enter beginner mode at any time through the App
	Traversable terrain	Hard roads, flat concrete roads, slopes of less than 15 degrees, steps not higher than 1 cm, and dips no more than 3 cm wide
	Operating temperature	14–104°F (-10–40°C)
	Storage temperature	-4–122°F (-20–50°C)
	Permitted charging temperature	32–104°F (0–40°C)
	IP rating	IP54
Battery Pack	Rated voltage	54.8 V⎓
	Maximum charging voltage	59.5 V ⎓
	Rated capacity	236 Wh
	Smart BMS	Overvoltage/ Under voltage/Short Circuit/Overheating Protection, Auto-Sleep/Wake-up, detailed information of battery can be checked with App.
	Maximum continuous discharge power[3]	1000 W
	Charging Temperature	32–104°F (0–40°C)
Motor	Rated power	400×2 W
	Maximum power	800×2 W
App connection	Compatible with	Android 4.3; iOS 8.0 or later
	Wireless Connectivity	Bluetooth 4.2

Table A.1: The specification sheet for the Ninebot S Segway model.