COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

BACHELOR'S DEGREE FINAL PROJECT

TIME BALL

Author: Gerardo Gavilanes Domínguez

Advisor: Joseph Skovira

Madrid

I declare, under my responsibility, that the Project presented with the title

Time Ball

in the ETS of Engineering - ICAI of the Universidad Pontificia Comillas in the

4th academic year is my authorship, original and unpublished and

it has not been previously submitted for other purposes.

The Project is not plagiarism of another, totally or partially, and the information that has been

taken from other documents is duly referenced.

Signed: Gerardo Gavilanes Domínguez Date: 25/06/2023

Project delivery authorized

THE PROJECT ADVISOR

Signed: Joseph Skovira Date: 25/06/2023

[illegible]

[illegible]
Gerardo Gavilanes Domínguez

COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

BACHELOR'S DEGREE FINAL PROJECT

TIME BALL

Author: Gerardo Gavilanes Domínguez

Advisor: Joseph Skovira

Madrid

TIME BALL

Author: Gavilanes Domínguez, Gerardo.
Advisor: Joseph Skovira
Collaborating Entity: ICAI – Universidad Pontificia Comillas and Cornell University

ABSTRACT

A Time Ball in New York City had stopped functioning properly, what prompted a project aimed at developing a new mechanism. Although the original mechanism was not replaced, an extensive study was conducted to explore the development approach. As a result, a functional prototype was successfully created, providing valuable guidance for future endeavors in replacing the Time Ball with an improved mechanism.

Keywords: Time Ball, Prototype Development, Motor Controller, Raspberry Pi, Motor.

1. Introduction

A time ball is a large ball that was traditionally used as a time signal, most commonly by being dropped or raised at noon. It was a common feature in port cities during the 19th and early 20th centuries and served the purpose of synchronizing time on both ships and on land. The ball was elevated to the top of a mast or tower and then dropped at an exact moment, typically signaled by a loud noise like a cannon shot. This allowed ships to calibrate their marine chronometers accurately and enabled people on land to set their watches correctly. However, the use of time balls has now largely been replaced by more advanced timekeeping techniques such as radio and satellite signals. Despite the prevalence of electronic time signals supplanting time balls, certain time balls persist as historical sites that draw in tourists. The Greenwich Time Ball, situated in the Royal Museum and illustrated in Figure 1, serves as an exemplary instance of this.

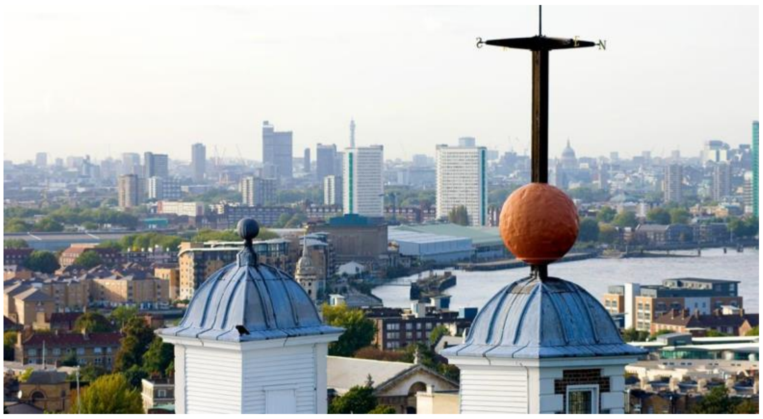


Figure 1. Greenwich Time Ball [1]

A specific Time Ball located in New York City had ceased working properly. The project to develop a new mechanism for this Time Ball was initiated after advisor Joseph Skovira, was offered the opportunity to replace this old mechanism.

2. Project definition

Initially, the project involved determining the exact features and functionalities required for the Time Ball system. This included defining the precise timing sequence for raising and dropping the ball, establishing the height positions, considering safety measures, and ensuring accurate synchronization with noon time.

Once the features were determined, the project focused on deliberating and selecting the necessary components and technologies to implement the Time Ball system in New York City. This involved carefully considering the mechanical, electronic, and control elements required for the operation of the Time Ball, ensuring they met the determined features and specifications.

Following the component deliberation, the project moved forward with the development of a functional prototype. The prototype served as a proof of concept, demonstrating the feasibility and effectiveness of the proposed control system. It involved designing and assembling the mechanical structure, integrating the electronics and control systems, and implementing the necessary code to control the Time Ball's movements.

3. System description

The prototype of the Time Ball control system was developed to imitate the behavior of a real Time Ball. It followed a specific sequence: 10 minutes before noon, the ball was lifted to the middle position on the post. Then, 5 minutes before noon, it was raised even higher to the top position. Finally, at precisely noon, the ball was dropped.

The prototype consisted of three main components: mechanical, electronics, and code. Each component underwent enhancements to achieve the desired functionality.

The mechanical aspect of the prototype involved crafted wooden structures for the main posts. The movement of the Time Ball was led by a specifically selected motor connected to gears. A chain mechanism with interconnected chain rings allowed precise control of the ball's movement along the post. The carrier, a sturdy wooden structure with wheels, provided stability and facilitated smooth rolling of the ball. Neodymium magnets on the carrier interacted with sensors for precise tracking and control.

The electronic aspect of the prototype included a Raspberry Pi 3, sensors, and a motor controller. The Raspberry Pi was connected to a TFT screen for display purposes. A breakout cable was used to establish the connection between the Raspberry Pi and the breadboard. The sensors used were Hall Effect Sensors strategically positioned along the ball's path. Magnets on the carrier triggered these sensors as the ball moved.

The motor controller played a crucial role in the electrical aspect of the prototype. It was necessary because the Raspberry Pi alone could not supply the required current for the motor. This motor controller enabled the Raspberry Pi to control the motor's speed and direction through the generation of PWM signals. The motor controller was connected to the motor and the Raspberry Pi, ensuring proper and efficient control of the motor's movements.

The prototype employed three Python scripts: main.py, gui.py, and alert.py. main.py controlled the motor controller, managing the ball's movement with specified timings and implementing a timeout for safety. It logged actions in "time_ball_log.txt." gui.py created a Pygame graphical interface on the TFT screen, displaying date, time, and remaining time until noon. It provided an IP address retrieval feature for remote connection. alert.py checked the log file for timing inconsistencies and sent email notifications with error messages if necessary.

The diagram depicted in Figure 2 illustrates the interconnectedness of various components.

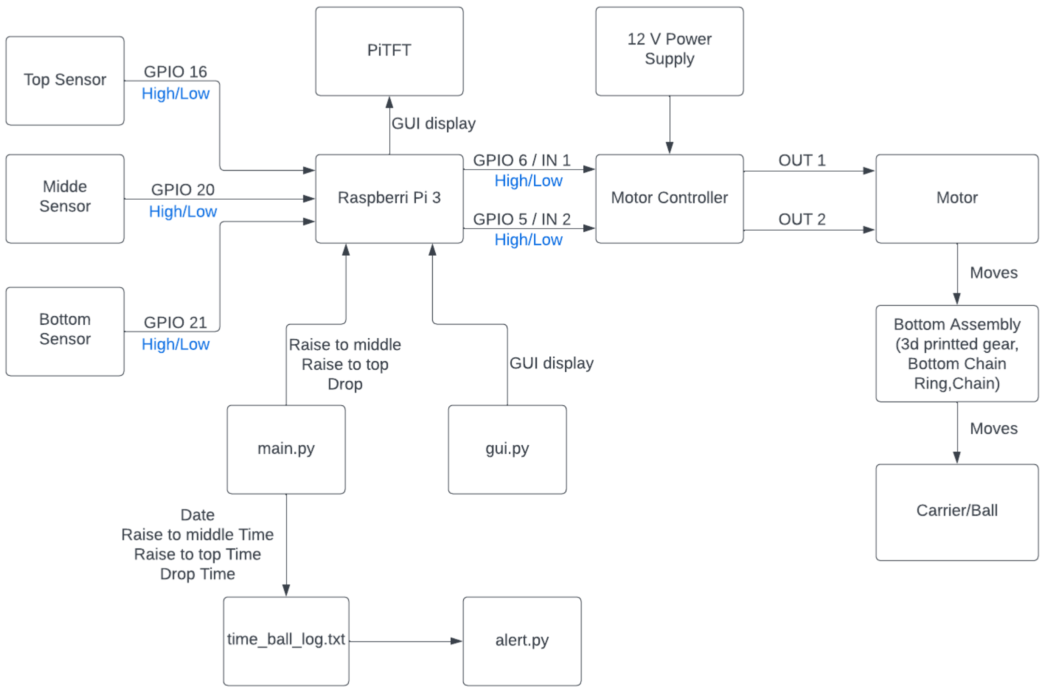


Figure 2. System Diagram

4. Results

The Time Ball prototype underwent successful development and testing, yielding the following outcomes:

Accurate Timekeeping: The prototype seamlessly synchronized with the designated time source, providing precise time indications, and establishing its reliability as a time indicator.

Reliable Mechanical Operation: The mechanical components ensured consistent and controlled movement of the Time Ball along the posts, accurately positioning it at designated time intervals.

Responsive Sensor Detection: Strategically positioned Hall Effect Sensors effectively detected the presence of magnets on the carrier, providing precise information about the ball's position.

Effective Motor Control: The motor controller and Raspberry Pi facilitated smooth and precise motor operation, ensuring accurate execution of desired movements.

User-Friendly Interface: The graphical interface displayed the current time, remaining time until noon, and other relevant information, with the added convenience of retrieving the Raspberry Pi's IP address.

Error Detection and Notification: The alert.py script promptly detected inconsistencies in timing, generating email notifications to address potential issues.

Figure 3 displays the labeled components of the prototype.

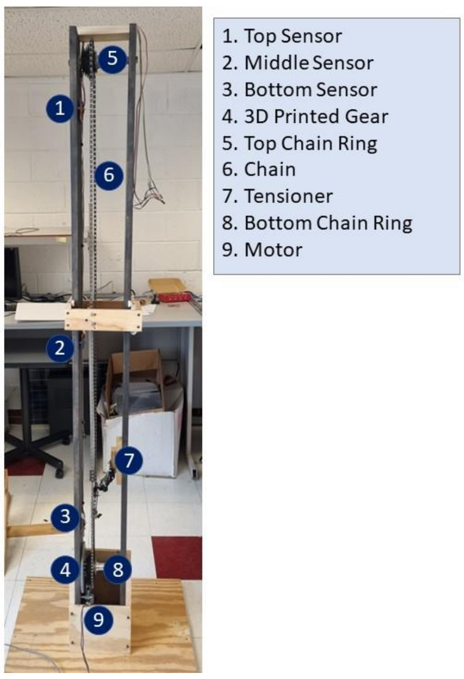


Figure 3. Prototype with some of the components labeled.

5. Conclusions

The development and successful execution of the Time Ball prototype have demonstrated its primary functionality and established it as a model for future implementation in the New York project. While there were opportunities to incorporate additional features, the prototype has laid a strong foundation for further advancements.

Potential future implementations can enhance the capabilities of the Time Ball prototype. These include: developing a web page with live streaming capabilities, integrating limit switches and implementing timed system activation to conserve energy.

References

[1] The Greenwich Time Ball. (n.d.). Royal Museums Greenwich. Last accessed on 20/06/2023. https://www.rmg.co.uk/royal-observatory/attractions/greenwich-time-ball

TIME BALL

Autor: Gavilanes Domínguez, Gerardo.
Director: Joseph Skovira
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas y Cornell University

RESUMEN DEL PROYECTO

Una esfera de tiempo presente en la ciudad de Nueva York dejó de funcionar correctamente, lo que originó un proyecto destinado a desarrollar un nuevo mecanismo para sustituir a la misma. Aunque el mecanismo original no llegó a ser reemplazado, se realizó un estudio de cara a una posible sustitución de este. Como resultado, se creó con éxito un prototipo funcional, brindando un valioso punto de partida para futuros proyectos con el mismo fin.

Palabras clave: Time Ball, Desarrollo de prototipo, Controlador de motor, Raspberry Pi, Motor.

1. Introducción

Una esfera horaria es una bola de gran tamaño que se utilizaba tradicionalmente como señal horaria, generalmente dejándola caer o alzándola al mediodía. Característica común en las ciudades portuarias durante los siglos XIX y principios del XX, la cual permitía la sincronización de la hora tanto en los barcos como en tierra. La bola se elevaba hasta la parte superior de un mástil o una torre y luego se dejaba caer en un momento exacto, generalmente señalado por un ruido fuerte como un disparo de cañón. Esto permitía a los barcos calibrar con precisión sus cronómetros marinos y permitía a las personas en tierra ajustar correctamente sus relojes. Sin embargo, el uso de las esferas horarias ha sido reemplazado en gran medida por técnicas de medición del tiempo más avanzadas, como son señales de radio y satélite. A pesar de la prevalencia de las señales electrónicas, las cuales han desbancado a las esferas horarias, algunas de ellas persisten como atracciones turísticas. La Esfera Horaria de Greenwich, situada en el Museo Real e ilustrada en la Imagen 1, sirve como ejemplo.

Imagen 1. Greenwich Time Ball [1]

Una esfera de tiempo ubicada en la ciudad de Nueva York dejó de funcionar correctamente. A mi director, Joseph Skovira, se le brindó la oportunidad de reemplazar este viejo mecanismo, lo que originó el proyecto para desarrollar un nuevo mecanismo.

2. Definición del proyecto

Inicialmente, el proyecto exigió determinar las características y funcionalidades exactas requeridas para el sistema de la bola de tiempo de Nueva York. Esto incluía definir la secuencia de tiempo precisa para elevar y liberar la bola, establecer las cotas de altura, consideraciones para las medidas de seguridad y asegurar una sincronización precisa con el mediodía, permitiendo soltar la bola en el instante preciso.

Una vez determinadas las características, el proyecto se centró en seleccionar los componentes adecuados y tecnologías necesarias para implementar el sistema en la ciudad de Nueva York. Esto implicaba considerar cuidadosamente los elementos mecánicos, electrónicos y de control necesarios para el funcionamiento de la esfera de tiempo, asegurándose de que cumplieran con las características y especificaciones determinadas.

Ya determinados los componentes, el proyecto avanzó hacia el desarrollo de un prototipo funcional. El prototipo servía como una prueba de verificación, demostrando la viabilidad y efectividad del sistema de control propuesto. Involucraba diseñar y ensamblar la estructura mecánica, integrar la electrónica y los sistemas de control, e implementar el código necesario para controlar los movimientos de la bola de tiempo.

3. Descripción del sistema

El objetivo del prototipo era el de imitar el comportamiento de una esfera de tiempo real. Seguía una secuencia específica: 10 minutos antes del mediodía, la bola se elevaba hasta la posición central del poste. Luego, 5 minutos antes del mediodía, se alzaba aún más hasta alcanzar el tope del poste. Finalmente, al mediodía exacto, la bola se dejaba caer.

Tres componentes principales: mecánico, electrónico y código, conformaron el prototipo.

Unas estructuras de madera para los postes principales conformaban la parte mecánica. Un motor guiaba el movimiento de la bola, mediante de su conexión con engranajes. Un mecanismo de cadena con anillos de cadena interconectados permitía un control preciso del movimiento de la bola a lo largo del poste. Se desarrolló una estructura de madera con ruedas, para proporcionar estabilidad y facilitar un deslizamiento suave de la bola a través del poste. Unos imanes de neodimio pegados en dicha estructura interactuaban con sensores para un seguimiento y control precisos.

La Raspberry Pi 3, sensores y un controlador de motor constituían la sección electrónica del prototipo. La Raspberry Pi estaba conectada a una pantalla TFT con fines de visualización. Se utilizó un cable de conexión para establecer la conexión entre la Raspberry Pi y la placa de pruebas. Los sensores utilizados fueron sensores de efecto Hall estratégicamente ubicados a lo largo del recorrido de la bola. Los imanes activaban estos sensores a medida que la bola avanzaba.

El controlador de motor desempeñó un papel crucial en el aspecto eléctrico del prototipo. Era necesario porque la Raspberry Pi por sí sola no podía suministrar la corriente requerida para el motor. Este controlador de motor permitió que la Raspberry Pi controlara la velocidad y dirección del motor mediante la generación de señales PWM. El controlador de motor se conectaba al motor y a la Raspberry Pi, asegurando un control adecuado y eficiente de los movimientos del motor.

El prototipo hizo uso de tres scripts de Python: main.py, gui.py y alert.py.

main.py se encargaba de dirigir el controlador de motor, gestionando el movimiento de la bola con tiempos específico e implementando un tiempo de espera por temas de seguridad. Registraba las acciones en "time_ball_log.txt".

gui.py creaba una interfaz gráfica Pygame en la pantalla TFT, mostrando la fecha, hora y tiempo restante hasta el mediodía. Proporcionaba una función de obtención de dirección IP para conexión remota.

alert.py verificaba el archivo de registro en busca de inconsistencias de tiempo y enviaba notificaciones por correo electrónico con mensajes de error en caso de ser necesario.

El diagrama representado en la Imagen 2 ilustra la interconexión de los distintos componentes.

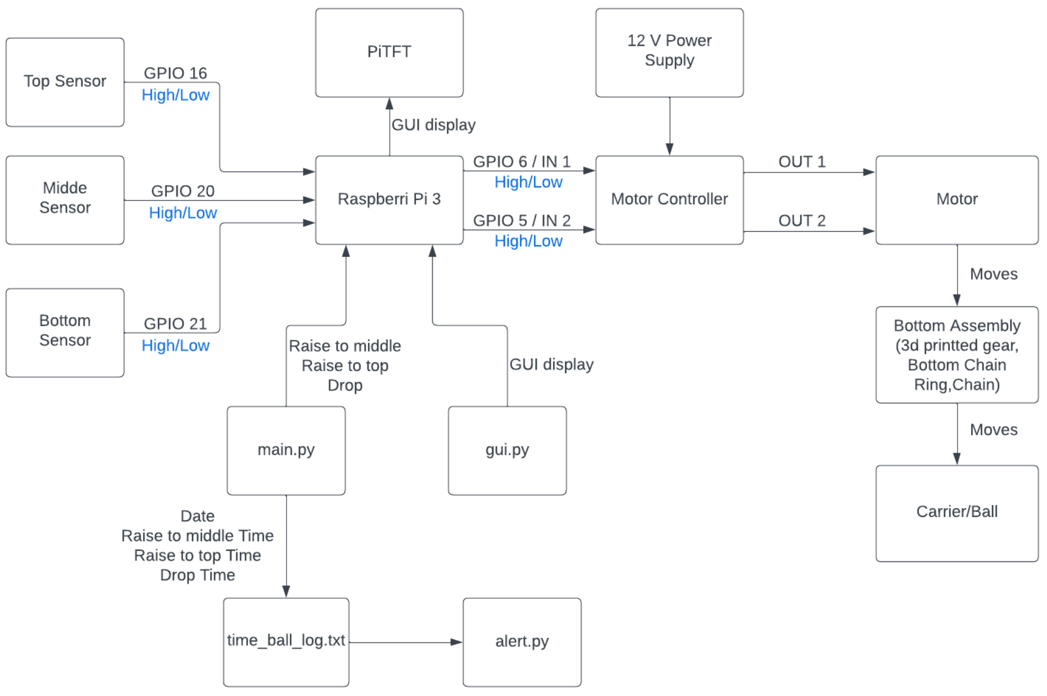


Imagen 2. System Diagram

4. Resultados

El prototipo se desarrolló exitosamente, alcanzándose los siguientes logros:

Precisión en la medición del tiempo: El prototipo se sincronizó perfectamente con la fuente de tiempo designada, proporcionando indicaciones precisas del tiempo y demostrando su confiabilidad como indicador de tiempo.

Operación mecánica fiable: Los componentes mecánicos permitieron un movimiento constante y controlado de la bola del tiempo a lo largo de los postes, posicionándola con precisión en los instantes designados.

Detección precisa con sensores: Los sensores de efecto Hall estratégicamente posicionados detectaron de manera efectiva la presencia de imanes en la bola, proporcionando información precisa sobre la posición de esta.

Control efectivo del motor: El controlador de motor y la Raspberry Pi facilitaron un funcionamiento suave y preciso del motor, asegurando la ejecución precisa de los movimientos deseados.

Interfaz fácil de usar: La interfaz gráfica mostraba la hora actual, el tiempo restante hasta el mediodía y otra información relevante, con la conveniencia adicional de mostrar la dirección IP de la Raspberry Pi.

Detección y notificación de errores: El script alert.py detectaba rápidamente inconsistencias en el tiempo, generando notificaciones por correo electrónico para abordar posibles problemas.

La Imagen 3 muestra los componentes del prototipo.

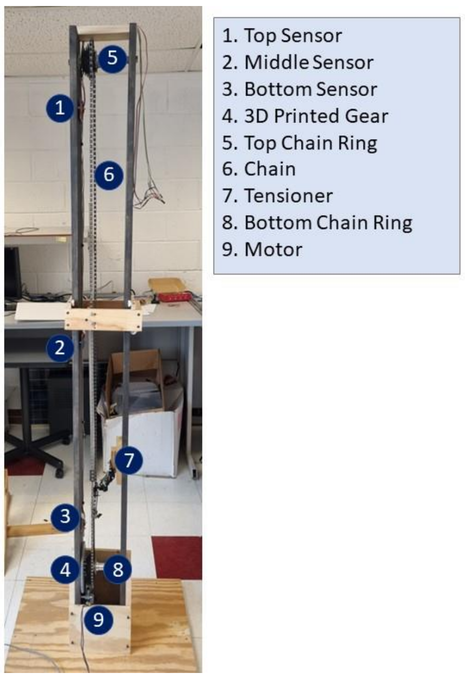


Imagen 3. Prototype with some of the components labeled.

5. Conclusiones

El desarrollo y la exitosa ejecución del prototipo de la esfera de tiempo han demostrado su funcionalidad principal y lo han establecido como un modelo para futuras implementaciones en el proyecto de Nueva York. Aunque hubo oportunidades para incorporar características adicionales, el prototipo ha sentado una base sólida para futuros avances.

Las posibles implementaciones futuras pueden mejorar las capacidades del prototipo. Estas incluyen: desarrollar una página web con capacidades de emisión en vivo, integrar interruptores de límite e implementar una activación del sistema programada para ahorrar energía.

Referencias

[1] The Greenwich Time Ball. (n.d.). Royal Museums Greenwich. Last

accessed on 20/06/2023.

https://www.rmg.co.uk/royal-observatory/attractions/greenwich-time-ball

Table of Contents

Table of Figures

CHAPTER 1. INTRODUCTION

The concept of time balls holds significant historical significance in the synchronization of time for both land and sea. In the past, time balls were large balls that were raised or dropped at noon, serving as a time signal for ships and individuals in port cities during the 19th and early 20th centuries. This practice allowed for accurate time calibration, enabling ships to set their marine chronometers and people on land to adjust their watches accordingly.

Although advanced timekeeping techniques such as radio and satellite signals have largely replaced time balls, they continue to exist as historical landmarks that attract tourists. In the heart of New York City, the Time Square Time Ball stands as a prominent example of this timekeeping tradition (see Figure 1). As the centerpiece of the renowned New Year's Eve celebration, it symbolizes the passage of time and marks the beginning of a new year.

Figure 4. Time Square Time Ball [1]

In the scope of this project, the focal point was directed towards a specific Time Ball situated in New York City, which had encountered operational challenges, leading to its loss of functionality. In response to this predicament, an initiative was undertaken to develop a novel mechanism for the Time Ball. The project's fundamental objective was driven by the recognition of the significance attached to the preservation of historical landmarks and the upholding of precise timekeeping traditions. Therefore, the endeavor aimed to replace the outdated mechanism with a contemporary and operational alternative.

The primary aim of the project was the creation of a functional prototype representing the Time Ball mechanism, intended to serve as a guiding framework for prospective implementation.

CHAPTER 2. DESCRIPTION OF THE TECHNOLOGIES

This chapter provides a comprehensive description of the technologies, protocols, and specific tools that will be employed throughout the project to facilitate the reading and understanding of the system's implementation. The following technologies and components will play a vital role in the development and operation of the Time Ball mechanism:

Raspberry Pi 3:

The Raspberry Pi 3, a versatile single-board computer, will serve as the central component of the system. It possesses substantial computational capabilities and interfaces necessary for controlling peripherals and sensors. The Raspberry Pi 3 will be responsible for executing the control logic and overseeing the overall operation of the Time Ball mechanism.

TFT Screen:

A TFT (Thin Film Transistor) screen will be utilized as the graphical interface for displaying pertinent information. Connected to the Raspberry Pi 3, the TFT screen will provide real-time updates on the current time, remaining time until noon, and other significant details. It offers a clear and user-friendly display, enabling convenient interaction with the Time Ball system.

Hall Effect Sensor:

Strategically positioned along the Time Ball's trajectory, Hall Effect sensors will utilize the Hall Effect phenomenon to detect the presence of magnets on the carrier, thereby providing precise information regarding the Time Ball's position.

PWM (Pulse Width Modulation):

Pulse Width Modulation is a technique employed for regulating the speed and intensity of electrical signals. In this project, PWM will be utilized to control the motor's movements.

By generating PWM signals, the Raspberry Pi 3, in conjunction with a motor controller, will govern the motor's speed and direction, facilitating precise and seamless operation of the Time Ball mechanism. Figure 5 provides an example of a PWM signal, illustrating the modulation of the signal's width to achieve different levels of motor control.

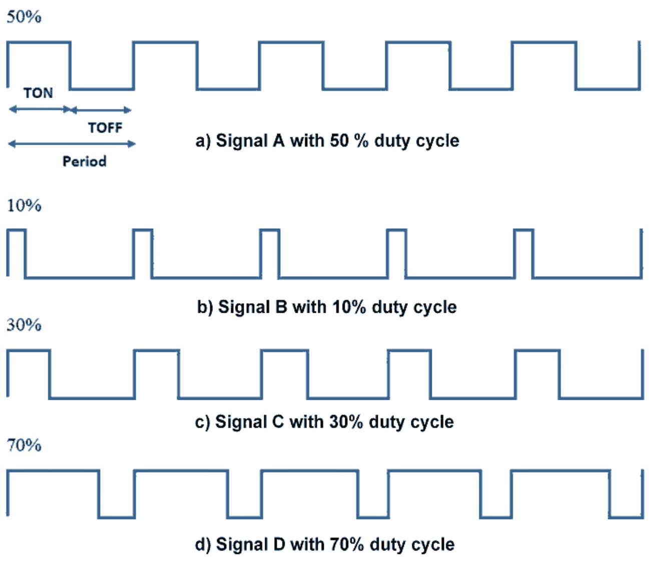

50%
TON
TOFF
Period
a) Signal A with 50 % duty cycle
10%
b) Signal B with 10% duty cycle
30%
c) Signal C with 30% duty cycle
70%
d) Signal D with 70% duty cycle

Figure 5. PWM Signals [2]

Python Libraries:

Several Python libraries will be utilized to streamline the development process and enhance the functionality of the Time Ball system. These libraries include:

- RPi.GPIO: This library provides access to the GPIO (General Purpose Input/Output) pins of the Raspberry Pi, enabling seamless communication with external devices and sensors.
- Time and Datetime: These libraries offer various functions and methods for handling time and date data, ensuring accurate timekeeping and scheduling of the Time Ball's movements.
- Sys: The Sys library provides access to system-specific parameters and functions, enabling the system to manage command-line arguments and interact with the operating system.

- Pygame: Pygame is a widely adopted library for Python game development. In this project, Pygame will be utilized to create a visually appealing graphical interface on the TFT screen, displaying relevant information and enhancing the user experience.
- Socket: The Socket library facilitates network communication between devices, enabling the establishment of connections for remote access or control of the Time Ball system.
- Netifaces: Netifaces offers a simplified interface for retrieving network interface information. It will be employed to obtain the IP address of the Raspberry Pi, facilitating remote connection and monitoring.
- smtplib: The smtplib library enables the sending of email notifications. It will be utilized by the alert.py script to dispatch error messages in the event of timing inconsistencies or system faults.

CHAPTER 3. STATE OF THE ART

The State of the Art in the scope of this project reveals a lack of modernized control systems for currently operational Time Balls, primarily due to the outdated mechanisms in place. In fact, it is noteworthy that many time balls are no longer operational, further exacerbating the scarcity of information regarding the direction and extent of modernization efforts in this field. Consequently, this project stands out as a unique and innovative endeavor that seeks to fill this gap and provide a fresh perspective on the modernization of Time Ball systems.

Prior to embarking on the development of any project, it is crucial to inquire about existing solutions and research work that may have already achieved the desired results. In the case of Time Balls, the dearth of comprehensive and up-to-date information highlights the significance of this project. By addressing the shortcomings of current systems and introducing novel features, this project aims to revolutionize the concept of Time Balls and establish a new standard for their operation and control.

The absence of comparable solutions on the market further emphasizes the novelty and potential impact of this undertaking. By undertaking this project, we are not only fulfilling a technological void but also paving the way for future advancements in the field of Time Ball systems.

CHAPTER 4. PROJECT DEFINITION

4.1 JUSTIFICATION

The Time Ball project offers numerous compelling reasons for potential buyers, clients, and investors to engage with this innovative endeavor. By addressing the limitations of existing Time Ball systems and introducing novel features, this project presents a unique value proposition that sets it apart.

The following aspects serve as the key justifications for the project:

- Addressing Limitations: The Time Ball project aims to overcome the outdated mechanisms and limited availability of operational time balls. By introducing a modernized control system, the project fills a gap in the market and presents a unique and innovative perspective in this field.
- Accurate Timekeeping: The primary objective of the Time Ball system is to determine noon for New York City every day of the year. With precise timing sequences, controlled motor movements, and accurate synchronization with noon time, the project ensures reliable and consistent timekeeping for the city.
- Safety and Reliability: The project emphasizes the implementation of fault-tolerant mechanisms and safety measures.
- Enhanced Visibility and Engagement: By integrating a webcam linked to a web page, the Time Ball project offers real-time visibility and engagement for a broader audience. This feature allows users to witness the Time Ball's descent on special occasions or events remotely, expanding its reach and impact beyond the physical location.

4.2 OBJECTIVES

The objectives of the Time Ball project can be summarized as follows:

- To accurately determine noon for New York City every day of the year, raising the ball ten minutes before noon and lowering it exactly at noon.
- To develop an assembly consisting of a tower-mounted post with a motor-controlled ball that traverses the post under precise control.
- To ensure fault tolerance and reliable operation in all weather conditions and temperatures.
- To implement monitoring and alerting mechanisms for timely fault detection and maintenance.
- To provide a visually captivating spectacle that becomes an iconic symbol for the city and fosters a sense of community.
- To incorporate a webcam linked to a web page, enabling real-time display and engagement for a broader audience.

4.3 METHODOLOGY

The Time Ball project followed a systematic approach to achieve its objectives. The methodology included the following stages:

- Determining Features and Functionality: The project began by precisely defining the features and functionalities required for the Time Ball system. This involved establishing the timing sequence, height positions, safety measures, and synchronization requirements with noon time.
- Component Selection: Deliberation and selection of the necessary components and technologies were crucial for the successful implementation of the Time Ball system. Careful consideration was given to mechanical, electronic, and control elements to ensure they met the project's determined features and specifications.
- Prototype Development: The project advanced to the development of a functional prototype to validate the feasibility and effectiveness of the proposed control system. This stage involved designing and assembling the mechanical structure, integrating

electronics and control systems, and implementing the required code to control the Time Ball's movements.

By following this methodology, the Time Ball project aimed to ensure the project's success and achieve its objectives effectively. The comprehensive approach provided a solid foundation for the subsequent stages, ensuring a well-defined path towards the realization of a modernized and captivating Time Ball system.

4.4 PLANIFICATION

Phase 1: Determining Features and Functionality (First Semester)

Phase 2: Component Selection and Prototype Development

CHAPTER 5. DEVELOPED SYSTEM

5.1 SOLUTION'S APPROACH

Prior to commencing the prototype development, a comprehensive examination of various approaches was conducted, encompassing all pertinent aspects of the New York Time Ball project.

5.1.1.1 RAISING MECHANISM

Regarding the raising mechanism, three alternatives had been evaluated: a chain-driven motor, a hydraulic mechanism, and a system of pulleys.

The second option was dismissed due to its intricate nature, while the third option was also rejected because concealing the mechanism within the post was deemed necessary. Consequently, a motor-driven mechanism with chains was chosen.

Thus, upon activation of the signal, the motor will elevate the ball to the pinnacle of the post and uphold its position solely through mechanical principles that counteract the force of gravity.

5.1.1.2 TOWER STRUCTURE AND BALL DESIGN

The tower structure necessitates ample internal space to conceal the raising mechanism, ensuring its invisibility from external view.

To achieve smooth and precise ball descent, the chains must facilitate its gradual descent and bring it to a halt in the correct position. Consequently, friction should not be relied upon within the post to halt the ball's movement, as this approach would prove inadequate given the challenging weather conditions prevalent in New York City.

The material chosen for the post must possess exceptional resistance to extreme weather conditions, including strong winds, heavy rain, snowfall, and freezing temperatures.

Optimal material for the ball should be one that imparts lightweight properties, hence, carbon fiber stands as an ideal candidate for this purpose.

Considering that the ball will remain atop the post for an extended duration, approximately 10 minutes, it is crucial to account for the occurrence of violent gusts of wind during this period. Such gusts may exert force on the ball, leading to oscillations in the post. To mitigate this issue, the ball will incorporate perforations, enabling the passage of wind through its interior and thereby counteracting the exerted "push force."

A comprehensive assessment of the post's load-bearing requirements assumes significance in this context.

5.1.1.3 ELECTRONIC COMPONENTS

Considering the straightforward integration of a webcam with Raspberry Pi and the convenient extraction of time from the Linux system, it is deemed appropriate to employ a Raspberry Pi as the controller for the motor.

The Raspberry Pi will retrieve the accurate New York time from the Linux system. Ten minutes prior to noon, the program will activate the motor to elevate the ball to the top position. At precisely noon, the program will initiate the motor to smoothly lower the ball.

As the lifting time (duration for the ball to reach the top) and dropping time (duration for the ball to reach the bottom) hold no significance, the system does not necessitate closed-loop controls, rendering precise timing unnecessary.

However, precision becomes crucial when ensuring that the ball is released from the top and begins its descent precisely at noon.

Furthermore, the Raspberry Pi will perform data logging and will be connected to a webcam that will provide live streaming to a dedicated web page.

To ascertain the ball's position, the post will incorporate sensors or limit switches positioned along its edges, enabling the program to accurately determine the ball's location within the post, be it at the top, bottom, or anywhere in between.

5.1.1.4 FAULT WARNING

Given the possibility of potential flaws in the ball's movement, it is imperative to monitor the motor's activation during both the ascent and descent phases. Prolonged duration in either direction would indicate a potential malfunction. Similarly, it is crucial to verify that the ball is released precisely at the designated time, necessitating accurate monitoring.

To address these concerns, the incorporation of sensors becomes pivotal in ensuring proper sequencing and pace of the ball's movement. However, one question arises: What if a sensor malfunctions? To mitigate this risk, a redundancy approach will be adopted, employing three sensors instead of one. Data from all three sensors will be compared within predefined tolerance limits. If one sensor's measurements deviate from the other two, the latter two will be considered accurate, thereby indicating a potential failure in the former sensor. In such cases, appropriate action will be taken to rectify the faulty sensor, and a notification will be issued to prompt the presence of a technician.

In the event of any detected faults, a warning message will promptly alert a qualified technician.

5.2 PROTOTYPE

5.2.1 OVERVIEW

The prototype aims to imitate the behavior of a real Time Ball. It follows a specific sequence: 10 minutes before noon, the ball is lifted to the middle position, halfway up the post. Then, 5 minutes before noon, it is raised even higher to the top position. Finally, at precisely noon, the ball is dropped.

The system is composed of three components: mechanical, electronic, and code. Additional features have been incorporated into the prototype, and each section discusses these enhancements.

The diagram depicted in Figure 2 illustrates the interconnectedness of various components.

Figure 3 displays the labeled components of the prototype.

5.2.2 MECHANICAL ASPECT

Wood Structure

All the structure for the project was handmade using wood. The two main posts were sandpapered and varnished.

Motor

The selected motor (depicted in Figure 6), resembling those commonly employed for car window operation, will serve the purpose. Notably, when unpowered, it will effectively sustain the position of the chain in direct opposition to the force of gravity. The DC motor possesses the advantageous characteristic of being able to alter its rotation direction by simply reversing the polarity of the applied voltage. To facilitate motor control, a motor controller will be utilized, as the motor's current requirements exceed the capacity of the Raspberry Pi to provide independently. This arrangement ensures proper and efficient control of the motor by the Raspberry Pi through the motor controller.
The motor employed 1.2 amperes.

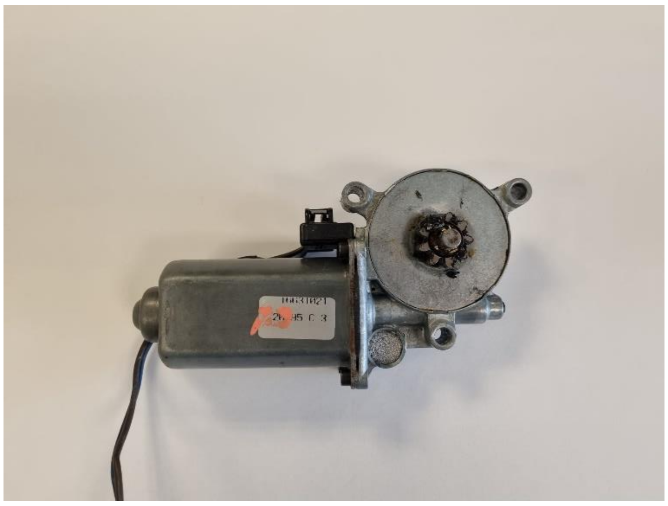


Figure 6. Motor

Chain Mechanism

The primary method of raising and lowering the ball involves a bicycle chain that has two chain rings. We adjusted the chain to the appropriate length for our specifications and secured it to the correct tension using a tensioner, which can be observed in Figure 7.

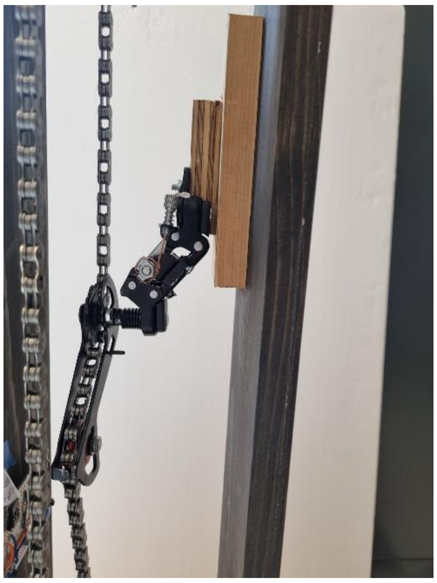


Figure 7. Chain Tensioner

The carrier is fastened to the chain. One chain ring is connected to the upper part of the post (Figure 8), while the other chain ring is connected to the lower part of the post. The chain is attached to both chain rings.

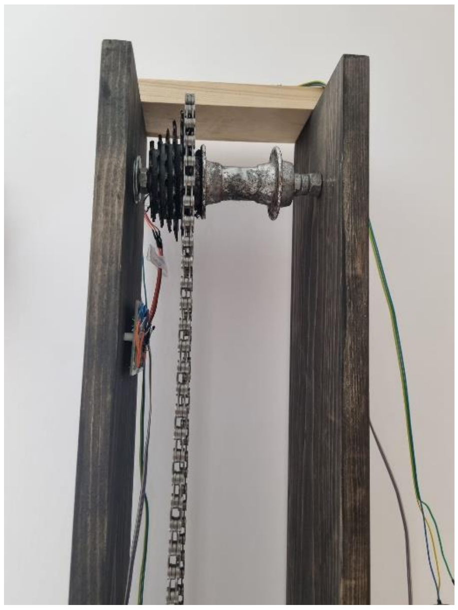


Figure 8. Top Chain Ring and Chain

The motor will power the chain ring located at the bottom, resulting in the movement of the chain, which in turn moves the carrier.

To ensure safety, a direct connection between the motor and the bottom chain ring is avoided. Instead, the motor will be responsible for driving a 3D printed gear, as seen in Figure 9, which will be securely attached to the chain ring. Through several iterations, the correct dimensions for this 3D printed gear were determined.

So, Figure 10 shows the bottom assembly, which consists of the motor, the 3D printed gear and the bottom chain ring.

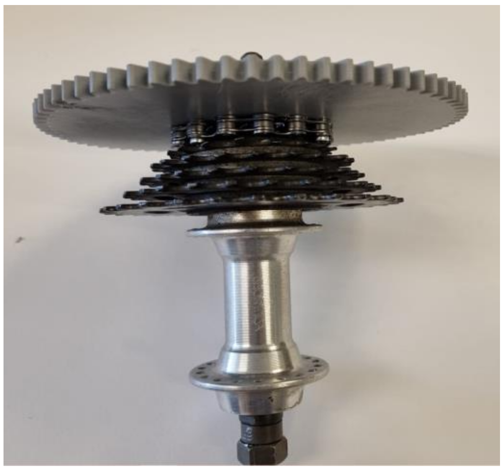


Figure 9. 3D Gear Attached to the Bottom Chain Ring

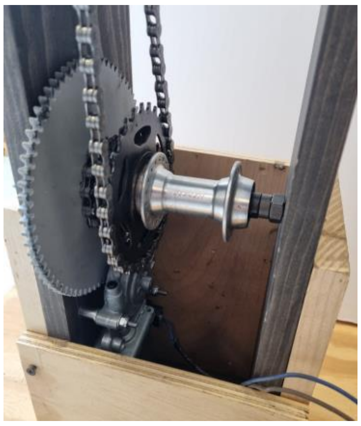


Figure 10. Bottom Assembly

Carrier

The wooden cubic structure, known as the carrier, is raised, and lowered by the chain. The ball encompasses this structure. Figure 11 shows a picture of it.

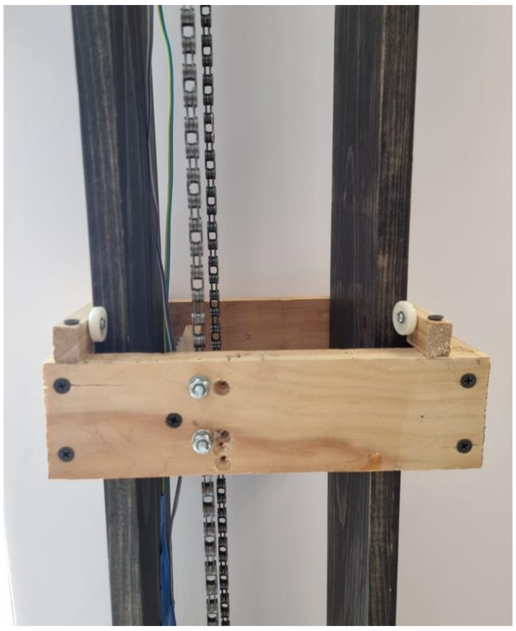


Figure 11. Carrier

Contained within the carrier is a wooden plank that accommodates neodymium magnets, Figure 12. As the ball moves, these magnets approach and interact with sensors located nearby, Figure 13.

The carrier is equipped with wheels, enabling it to roll across the posts.

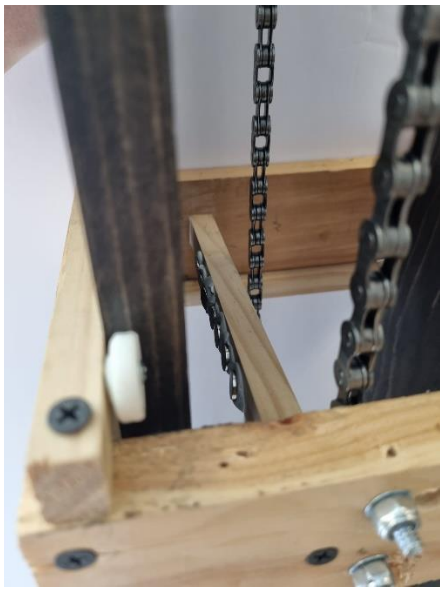


Figure 12. Magnets

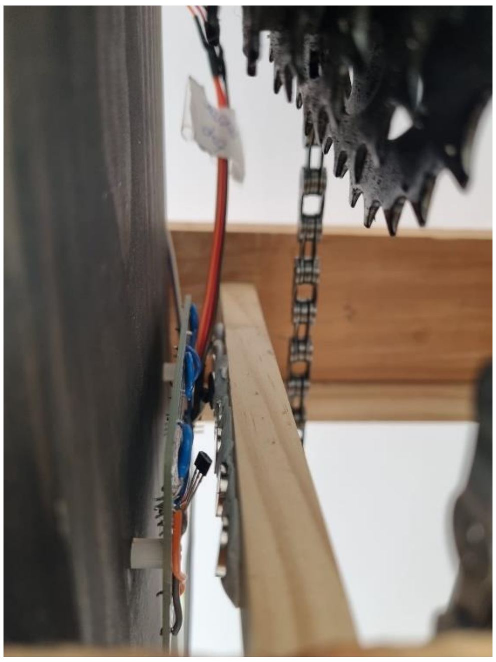


Figure 13. Magnets trigger sensors

5.2.3 ELECTRONICS

The Raspberry Pi, sensors and motor driver form the electronic aspect of the project. We will talk about them in detail in the following sections.

Raspberry Pi

For the project, a Raspberry Pi 3 and a TFT screen were utilized for display purposes. To establish the connection, a breakout cable was employed to link the pins to a Breadboard. The specific pins used in the setup are as follows:

AIN1Pin: GPIO 6

AIN2Pin: GPIO 5

Topsensor_pin: GPIO 16

Midsensor_pin: GPIO 20

Bottomsensor_pin: GPIO 21

Motor Controller

The Raspberry Pi alone is unable to supply the required current for the motor, which is why a motor controller is necessary. Initially, a small DC motor was utilized for testing purposes before employing the actual motor. The TB6612FNG motor controller, capable of handling up to 1.2 A, was chosen for controlling this motor. It was anticipated that switching to the real motor would not necessitate changing the motor driver. To control the motor, a PWM signal is generated from the Raspberry Pi, which regulates the motor's speed. Additionally, two output signals are utilized to modify the motor's direction.
After confirming that the circuit was functional, all the necessary headers for the inputs and outputs of the Raspberry Pi were soldered onto it, as exhibited in Figure 14.

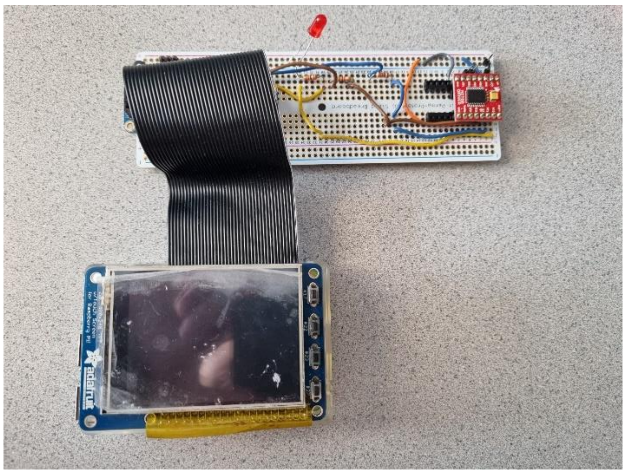


Figure 14. Soldered Circuit for controlling the motor, and all the necessary headers for the Raspberry Pi

Upon transitioning to the actual motor, we encountered issues with the motor controller, which did not operate correctly. Consequently, we needed to replace the motor driver with one capable of handling higher currents. The MC33926, with a capacity of 3 Amps, was selected as the alternative motor driver. Figure 15 illustrates a layout of the MC33926 motor driver.

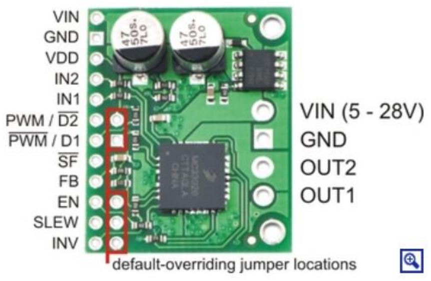


Figure 15. MC33926 ***Layout*** *[3]*

The new motor controller exhibited some differences in behavior compared to the previous one, requiring additional time and effort to get it functioning correctly. On the right side of the controller, the VIN (with a voltage range of 5-28V) and GND pins are connected to an external 12V power supply. The other two pins on the same side, OUT1 and OUT2, are connected to the motor's two wires.

The left side pins had to be connected the following way:
GND → Raspberry Pi GND
VDD → 3.3 V Raspberry Pi
$\overline{D2}$ → High (3.3 V, disable it)
$D1$ → Low (3.3 V, disable it)

EN → High (3.3 V)

Fortunately, successful motor control was achieved through the utilization of two distinct methods, each tailored to accommodate varying speeds.

Full Speed:
The Raspberry Pi will generate two output signals, namely IN1 and IN2. One of these signals will be set to a high state, while the other will be set to a low state. By altering the states of these signals, the direction of the motor can be changed.

Half Speed:
The Raspberry Pi will generate two output signals, IN1 and IN2. One of the pins will receive a PWM signal, while the other will receive a low signal. By altering the states of these

signals, the direction of the motor can be changed. Additionally, to achieve half speed, a duty cycle of 50 can be used.

The current motor driver functions flawlessly with the actual motor, enabling it to operate at full speed, which is deemed reasonable.
Once the new motor controller was soldered (Figure 16), it ceased functioning, and there was insufficient time available to investigate the reason behind the malfunction.

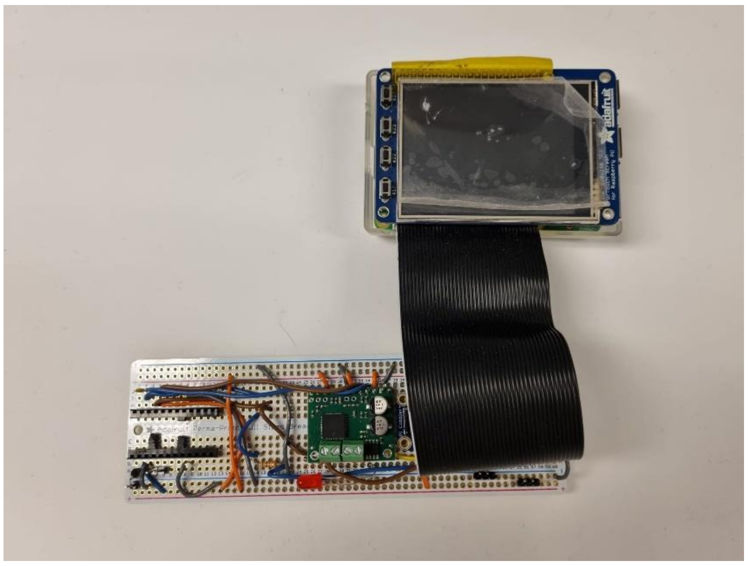


Figure 16. Circuit for the MC33926 motor controller

<u>Sensors</u>
The implementation involved the utilization of Hall Effect Sensors, which were strategically positioned along the path of the ball's movement. Magnets were affixed to the carrier, and as the ball traversed its trajectory, these magnets would approach the sensors, triggering their activation.
Considering the desired stopping points of the ball at the bottom, middle, and top of the post, sensors were installed at each of these positions. To enhance system reliability, a cluster of three sensors was employed for each specific location. This redundancy approach ensured that the system would maintain its functionality even if one of the sensors experienced a failure, thus enhancing overall reliability.

The non-latching AH1815 Hall Effect Sensor was selected for the project. Each sensor consisted of essential connections such as ground, power supply, and output, what is represented in Figure 17. To ensure proper functioning of the sensors, a capacitor and a resistor were incorporated. The supply pin of the sensor was connected to the 3.3 V output sourced from the Raspberry Pi, while the resistor was implemented using the internal pull-up resistor feature available within the Raspberry Pi. This configuration allowed for accurate and reliable behavior of the sensors within the system.

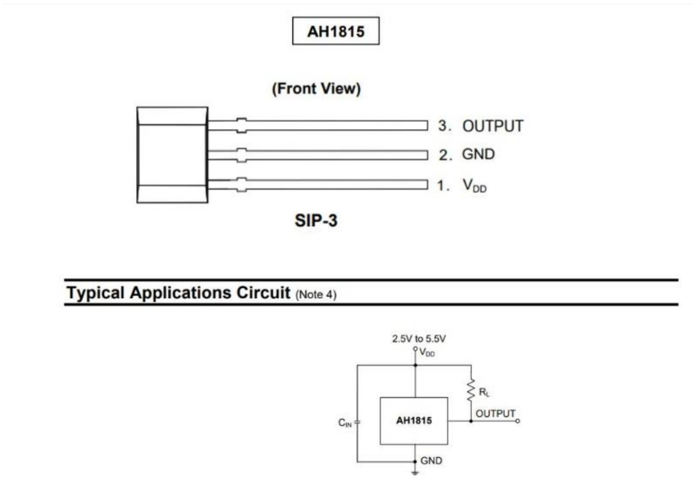


Figure 17. AH1815 Datasheet [4]

A circuit was meticulously constructed for each of the three designated positions where the ball was intended to stop. Within each circuit, three sensors were interconnected to operate in unison, effectively functioning as a single unit.

Following comprehensive testing to ensure their proper functionality, these three circuits were securely soldered and subsequently affixed to the post, thereby completing their installation, see and Figure 18 and Figure 19.

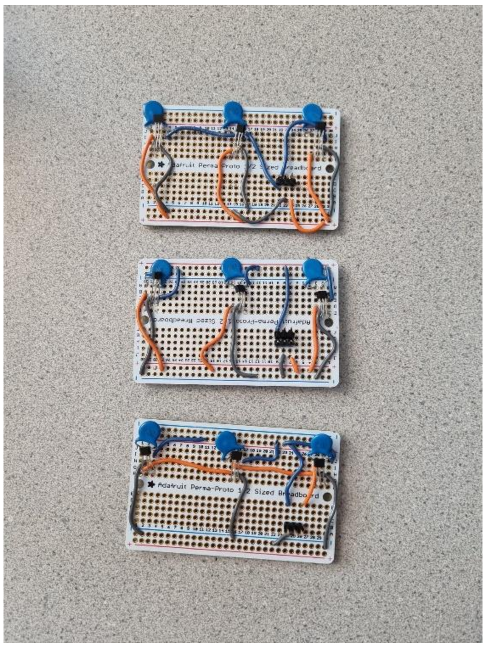


Figure 18. Three Soldered Circuits

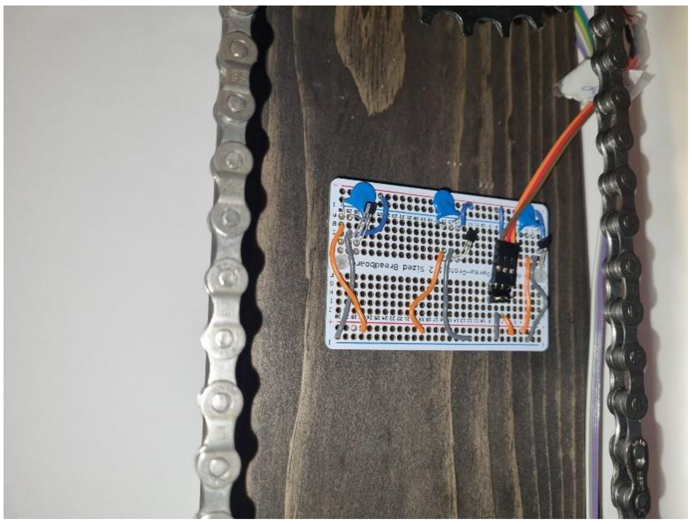


Figure 19. Hall Effect Sensor Circuit Attached to the Top of the Post

Within each circuit, there are three distinct headers, namely the ground header, the 3.3 V power supply header, and the output header, shown in Figure 20. These headers are connected to the Raspberry Pi through cables, establishing the necessary electrical connections between the components.

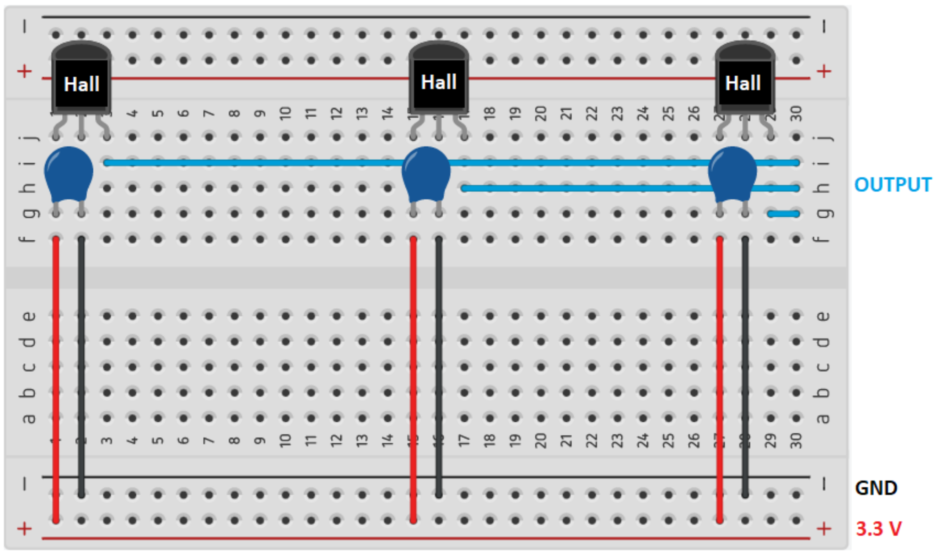


Figure 20. Circuit Diagram

5.2.4 CODE

Python code was used for the Raspberry Pi. All the code is in Appendix II. The main three scripts for the project are:

. main.py

For the MC33926 motor controller.

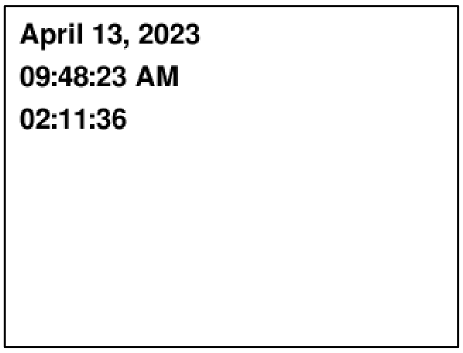
The program assumes that the ball begins at the bottom position. 10 minutes prior to noon, the program raises the ball to the middle position. Then, 5 minutes before noon, the ball is further elevated to the top position. Finally, precisely at noon, the program initiates the drop of the ball.

For each state, a timeout is implemented, which means that if the process exceeds the specified timeout duration, the program will shut down as a safety precaution.

Furthermore, the script records the precise date and time of each action in a text file named "time_ball_log.txt."

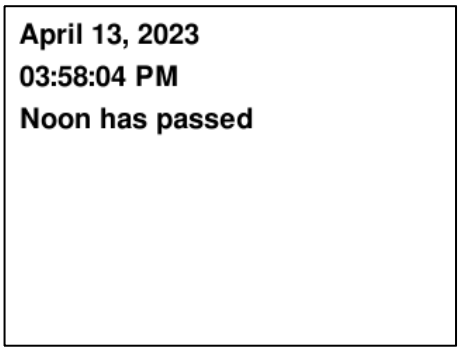
. gui.py

This script creates a graphical interface using Pygame on the TFT screen plugged to the Raspberry Pi. The interface displays the current date and time, as well as the time remaining until noon, Figure 21. If it is already past noon, a message indicating that noon has passed is shown instead, Figure 22. Additionally, when a button in the TFT is pressed, the code retrieves the IP address of the Raspberry Pi and displays it on the screen, Figure 23. This feature is convenient for easily connecting to the Raspberry Pi remotely from a laptop or other devices.



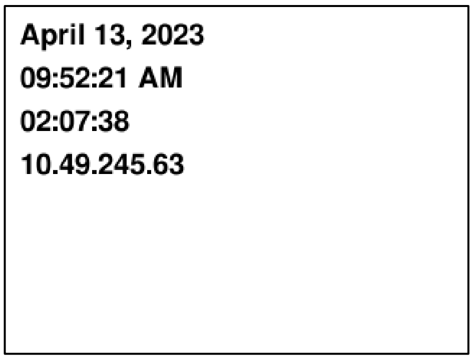
```
April 13, 2023
09:48:23 AM
02:11:36
```

Figure 21. Display. Time to noon



```
April 13, 2023
03:58:04 PM
Noon has passed
```

Figure 22. Display. After noon



```
April 13, 2023
09:52:21 AM
02:07:38
10.49.245.63
```

Figure 23. Display. Showing IP

. alert.py

The script examines the log file containing the actions and their corresponding timestamps to verify if they occur at the expected times. If any inconsistencies are detected, it sends an email notification containing an error message.

<u>Code for Testing</u>

Some code that was useful for testing and might be useful in the future.

. demo_full_speed.py

For the MC33926 motor controller.

The program assumes that the ball begins at the bottom position. It follows a sequence of actions, starting with a wait time of "raise_halfway_time" seconds before moving the ball to the middle position. Subsequently, it waits for "raise_top_time" seconds and elevates the ball to the top of the post. Then, it waits for "drop_time" seconds and releases the ball, causing it to drop back to the bottom position. For each state, a timeout is implemented, which means that if the process exceeds the specified timeout duration, the program will shut down as a safety precaution.

. testing.py

For the TB6612FNG motor controller.

Makes the ball go from bottom to top and vice versa.

. motor_half_speed.py

For the MC33926 motor controller.

Runs the motor forward at half speed.

. motor_full_speed.py

For the MC33926 motor controller.

Runs the motor forward at full speed.

. top_to_bottom_half_speed.py

For the MC33926 motor controller.

Makes the ball go from bottom to top and vice versa at half speed.

. top_to_bottom_full_speed.py

For the MC33926 motor controller.

Makes the ball go from bottom to top and vice versa at full speed.

. demo_firstmc.py

For the TB6612FNG motor controller.

The program assumes that the ball begins at the bottom position. It follows a sequence of actions, starting with a wait time of "raise_halfway_time" seconds before moving the ball to the middle position. Subsequently, it waits for "raise_top_time" seconds and elevates the ball to the top of the post. Then, it waits for "drop_time" seconds and releases the ball, causing it to drop back to the bottom position. For each state, a timeout is implemented, which means that if the process exceeds the specified timeout duration, the program will shut down as a safety precaution.

It also displays on the TFT, the action taking place, or the time next to the next action.

CHAPTER 6. RESULTS ANALYSIS

This chapter presents an analysis of the most significant results obtained throughout the project, taking into consideration the successful development and testing of the Time Ball prototype. The outcomes achieved demonstrate the project's effectiveness in meeting its objectives and highlight the notable contributions in various aspects.

Accurate Timekeeping:

The prototype exhibited seamless synchronization with the designated time source, providing precise time indications and establishing itself as a reliable time indicator. This achievement ensures the accurate calibration of time, benefiting both maritime and terrestrial applications.

Reliable Mechanical Operation:

The mechanical components employed in the prototype demonstrated consistent and controlled movement of the Time Ball along the posts. The accurate positioning of the Time Ball at designated time intervals guarantees reliable time signaling and reinforces the functionality of the mechanism.

Responsive Sensor Detection:

Strategically positioned Hall Effect Sensors effectively detected the presence of magnets on the carrier, providing precise information about the position of the Time Ball. This responsiveness ensures the accuracy and reliability of the time signaling process.

Effective Motor Control:

The motor controller, in conjunction with the Raspberry Pi, facilitated smooth and precise motor operation, ensuring the accurate execution of desired movements. This efficient motor control mechanism enables the Time Ball to perform its time signaling function reliably.

User-Friendly Interface:

The graphical interface of the prototype presented the current time, remaining time until noon, and other relevant information. Additionally, the interface provided the convenience of retrieving the Raspberry Pi's IP address, enhancing the user experience and accessibility of the system.

Error Detection and Notification:

The alert.py script promptly identified inconsistencies in timing, triggering email notifications to address potential issues. This error detection and notification mechanism ensures the system's integrity and allows for timely troubleshooting and maintenance.

Figure 3 illustrates the labeled components of the prototype, providing a visual representation of the integrated system.

CHAPTER 7. CONCLUSIONS AND FUTURE DIRECTIONS

The development and successful execution of the Time Ball prototype demonstrated its primary functionality and established it as a model for future implementation in the New York project. In discussions surrounding the real New York Time Ball, various features and technologies were explored to enhance its capabilities and ensure its seamless integration into the project.

During the development process, opportunities arose to incorporate additional features into the prototype, which laid a strong foundation for further advancements. The extensive research, design, and testing processes not only validated the feasibility of the concept but also provided valuable insights for future iterations.

One aspect of future progress involves the establishment of a dedicated web page equipped with live streaming functionality. Incorporating a web-based platform would enable users to remotely access up-to-the-minute updates and observe the Time Ball's descent during noteworthy occasions or events. This addition would not only broaden the Time Ball's influence and significance but also allow a wider audience to actively participate in the project. Attaching a camera to the Raspberry Pi would facilitate the seamless implementation of this enhancement.

Moreover, incorporating limit switches as standalone elements presents a compelling path for advancement. These independent limit switches would establish a direct connection with the motor, serving as a crucial safety measure. In the event of a malfunctioning hall-effect sensor, the limit switches would promptly halt the motor's operation, effectively safeguarding the prototype.

To optimize energy usage and promote sustainability, timed system activation can be introduced. By integrating a timed activation mechanism, the Time Ball's operation can be adjusted to specific time periods or events, conserving energy during periods of low activity. This enhancement would not only contribute to the overall energy efficiency of the project but also extend the lifespan of the system's components.

In conclusion, the development and successful execution of the Time Ball prototype have laid a strong foundation for the New York project, establishing its primary functionality and reliability. This achievement paves the way for future enhancements, including the development of a web page with live streaming capabilities, the integration of independent limit switches, the implementation of timed system activation, and the attachment of an additional ball component to the carrier. By considering these potential future implementations, the Time Ball project can continue to evolve, captivating audiences and symbolizing the passage of time.

BIBLIOGRAPHY

[1] Bleiberg, L. (2022, February 25). The invention that inspired a New York tradition. *BBC Travel*. Last accessed on 18/06/2023. https://www.bbc.com/travel/article/20191212-the-invention-that-inspired-a-new-york-tradition

[2] *PWM in ARM LPC2148 | ARM7-LPC2148*. (n.d.). © 2018 ElectronicWings. Last accessed on 12/06/2023. https://www.electronicwings.com/arm7/lpc2148-pwm

[3] *Pololu - MC33926 Motor Driver Carrier*. (n.d.). Last accessed on 13/06/2023. https://www.pololu.com/product/1212

[4] *Hall-Effect Sensor - AH1815 (Non-Latching)*. (n.d.). SEN-14709 – SparkFun Electronics. Last accessed on 20/06/2023. https://www.sparkfun.com/products/14709

APPENDIX I. ALIGNMENT WITH THE SUSTAINABLE DEVELOPMENT GOALS (SDGS)

While the Time Ball project does not directly align with any specific Sustainable Development Goal (SDG), it indirectly contributes to the broader objectives outlined by the SDGs. By preserving historical landmarks, advancing technology, and promoting knowledge dissemination, the project supports sustainable cities and communities (SDG 11) and industry innovation and infrastructure (SDG 9).

Although not directly related, the project embodies the spirit of sustainable development by blending tradition with innovation and benefiting local communities and visitors.

APPENDIX II

Main code:

main.py:

```
import RPi.GPIO as GPIO
import time
import datetime
import sys

# Pin definitions.
AIN1Pin = 6
AIN2Pin = 5

Topsensor_pin = 16
Midsensor_pin = 20
Bottomsensor_pin = 21

#timeouts
raise_halfway_timeout = 13
raise_top_timeout = 13.6
drop_timeout = 26.3

# define a function to exit the program
def exit_program():
    GPIO.output(AIN1Pin, GPIO.LOW)
    GPIO.output(AIN2Pin, GPIO.LOW)
    sys.exit()

def setup():
    # Setup GPIO.
    GPIO.setmode(GPIO.BCM)

    GPIO.setup(AIN1Pin, GPIO.OUT)
    GPIO.setup(AIN2Pin, GPIO.OUT)
    GPIO.setup(Topsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Midsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Bottomsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)


def forward_to_middle():
    start_time_1 = time.time()
    log_entry("Started moving to middle: " +
datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
    Midsensor_status = GPIO.input(Midsensor_pin)
    while Midsensor_status == 1:
        # Going forwards (going up) to the middle.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)
        Midsensor_status = GPIO.input(Midsensor_pin)

        # Check if the elapsed time has surpassed the timeout.
        elapsed_time = time.time() - start_time_1
        if elapsed_time > raise_halfway_timeout:
            print("Timeout reached. Shutting down.")
            # Call a function to close the program.
            exit_program()

    #stop
    GPIO.output(AIN1Pin, GPIO.LOW)

def forward_to_top():
    start_time_2 = time.time()
    log_entry("Started moving to top: " +
datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
    Topsensor_status = GPIO.input(Topsensor_pin)
    while Topsensor_status == 1:
        # Going forwards (going up) to the top.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)


        Topsensor_status = GPIO.input(Topsensor_pin)

        # Check if the elapsed time has surpassed the timeout.
        elapsed_time = time.time() - start_time_2
        if elapsed_time > raise_top_timeout:
            print("Timeout reached. Shutting down.")
            # Call a function to close the program.
            exit_program()

    #stop
    GPIO.output(AIN1Pin, GPIO.LOW)

def backwards():
    start_time_3 = time.time()
    log_entry("Dropped: " + datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S"))
    Bottomsensor_status = GPIO.input(Bottomsensor_pin)
    while Bottomsensor_status == 1:
        # Going backwards (going down).
        GPIO.output(AIN1Pin, GPIO.LOW)
        GPIO.output(AIN2Pin, GPIO.HIGH)

        Bottomsensor_status = GPIO.input(Bottomsensor_pin)

        # Check if the elapsed time has surpassed the timeout.
        elapsed_time = time.time() - start_time_3
        if elapsed_time > drop_timeout:
            print("Timeout reached. Shutting down.")
            # Call a function to close the program.
            exit_program()

    GPIO.output(AIN2Pin, GPIO.LOW)
```

```
def sequence():
    GPIO.output(AIN1Pin, GPIO.LOW)
    GPIO.output(AIN2Pin, GPIO.LOW)

    # Flags
    raised_to_middle = False
    raised_to_top = False
    dropped = False

    # Set up the log file.
    try:
        log_file = open("time_ball_log.txt", "a")
    except:
        print("Unable to open log file")
        return

    while True:
        current_time = datetime.datetime.now().time()

        # 10 minutes before noon, raise the ball to the middle of the
post.
        if current_time.hour == 11 and current_time.minute >= 50 and not
raised_to_middle:
            print("Triggering forward_to_middle()")
            forward_to_middle()
            raised_to_middle = True

        # 5 minutes before noon, raise the ball to the top of the post.
        if current_time.hour == 11 and current_time.minute >= 55 and not
raised_to_top:
            print("Triggering forward_to_top()")
            forward_to_top()
            raised_to_top = True

        # At noon, drop the ball from the top to the bottom.
        if current_time.hour == 12 and current_time.minute == 0 and not
dropped:
            print("Triggering backwards()")
            backwards()
            dropped = True

        # Check if all the actions have been completed
        if raised_to_middle and raised_to_top and dropped:
            break

        time.sleep(1)  # Wait for 1


if __name__ == '__main__':
    setup()
    try:
        sequence()
    except KeyboardInterrupt:
        GPIO.cleanup()
```

gui.py

```
import pygame
import datetime
import socket
import RPi.GPIO as GPIO
import netifaces

def update_time():
    # Get the current date and time
    now = datetime.datetime.now()
    date_string = now.strftime("%B %d, %Y")
    time_string = now.strftime("%I:%M:%S %p")

    # Calculate the time left to noon
    noon = datetime.time(12, 0, 0)
    if now.time() < noon:
        diff = datetime.datetime.combine(now, noon) - now
        hours, remainder = divmod(diff.total_seconds(), 3600)
        minutes, seconds = divmod(remainder, 60)
        time_left =
f"{int(hours):02d}:{int(minutes):02d}:{int(seconds):02d}"
    else:
        time_left = "Noon has passed"

    # Clear the screen
    screen.fill((255, 255, 255))

    # Draw the date, time, and time left to noon
    date_text = font.render(date_string, True, (0, 0, 0))
    screen.blit(date_text, (10, 10))
    time_text = font.render(time_string, True, (0, 0, 0))
    screen.blit(time_text, (10, 40))
    noon_text = font.render(time_left, True, (0, 0, 0))
    screen.blit(noon_text, (10, 70))

    # Update the screen
    pygame.display.flip()


def get_ip():
    global ip_address,ip_displayed
    ip_address =
netifaces.ifaddresses('wlan0')[netifaces.AF_INET][0]['addr']
    ip_displayed = True
    ip_text = font.render(ip_address, True, (0, 0, 0))
    screen.blit(ip_text, (10, 100))
    pygame.display.flip()

def button_callback(channel):
    get_ip()
```

```
pygame.init()

# Create the screen
screen = pygame.display.set_mode((320, 240))
pygame.display.set_caption("Current Date and Time")

# Create the font
font = pygame.font.Font(None, 30)

# Configuring the button
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(17, GPIO.FALLING, callback=button_callback,
bouncetime=300)

# Main loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    update_time()
    pygame.time.wait(1000)

pygame.quit()
```

alert.py

```
import time
import smtplib

def send_alert(message):
    # Set up the SMTP server
    server = smtplib.SMTP('smtp.example.com')

    # Send the email
    server.sendmail(
        "time-ball@example.com",  # From address
        "admin@example.com",      # To address
        message                    # Message
    )

    # Disconnect from the server
    server.quit()


# Set up the log file
log_file = open("time_ball_log.txt", "r")

# Read the log file line by line
for line in log_file:
    # Split the line into action and time
    action, timestamp = line.strip().split(',')
    # Convert the timestamp to a time object
    log_time = time.strptime(timestamp, "%Y-%m-%d %H:%M:%S")

    # Get the current time
    current_time = time.localtime()

    # Check the action and time
    if action == "Started moving to middle" and not (current_time.tm_hour
== log_time.tm_hour - 1 and current_time.tm_min >= 50):
        send_alert("Action 'Started moving to middle' did not occur at
the expected time.")
    elif action == "Started moving to top" and not (current_time.tm_hour
== log_time.tm_hour - 1 and current_time.tm_min >= 55):
        send_alert("Action 'Started moving to top' did not occur at the
expected time.")
    elif action == "Dropped" and not (current_time.tm_hour ==
log_time.tm_hour and current_time.tm_min == 0):
        send_alert("Action 'Dropped' did not occur at the expected
time.")

# Close the log file
log_file.close()
```

Code for testing purposes:

<u>demo_firstmc.py</u>

```
import RPi.GPIO as GPIO
import time
import sys
import pygame

pygame.init()

# Set the width and height of the screen (optional).
screen_width = 400
screen_height = 300
screen = pygame.display.set_mode((screen_width, screen_height))


# Pin definitions.
pwmPin = 26
AIN1Pin = 5
AIN2Pin = 6

Topsensor_pin = 16
Midsensor_pin = 20
Bottomsensor_pin = 21

# Set the times in seconds
raise_halfway_time = 10
raise_top_time = 20
drop_time = 30
```

```
restart_time = 40

#timeouts
raise_halfway_timeout = 20
raise_top_timeout = 20
drop_timeout = 20


duty = 50
frequency = 50


def display_text(text):
    font = pygame.font.Font(None, 36)  # Choose a font and size.
    text_surface = font.render(text, True, (255, 255, 255))  # Create a
surface for the text.
    screen.fill((0, 0, 0))  # Clear the screen.
    text_rect = text_surface.get_rect(center=(screen_width / 2,
screen_height / 2))
    screen.blit(text_surface, text_rect)
    pygame.display.flip()  # Update the screen.


# define a function to exit the program
def exit_program():
    sys.exit()

def setup():
    # Setup GPIO.
    GPIO.setmode(GPIO.BCM)

    GPIO.setup(pwmPin, GPIO.OUT)
    GPIO.setup(AIN1Pin, GPIO.OUT)
    GPIO.setup(AIN2Pin, GPIO.OUT)
    GPIO.setup(Topsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Midsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Bottomsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

    global pwm
    pwm = GPIO.PWM(pwmPin, frequency)

def forward_to_middle():
    display_text("Moving ball to middle...")
    start_time_1 = time.time()

    Midsensor_status = GPIO.input(Midsensor_pin)
    while Midsensor_status == 1:
        # Going forwards (going up) to the middle.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)
        pwm.start(duty)

        Midsensor_status = GPIO.input(Midsensor_pin)

        # Check if the elapsed time has surpassed the timeout.
        elapsed_time = time.time() - start_time_1
        if elapsed_time > raise_halfway_timeout:
            print("Timeout reached. Shutting down.")
            # Call a function to shutdown the system.
            exit_program()
    pwm.ChangeDutyCycle(0)  # The ball is already at the middle.

def forward_to_top():
    display_text("Moving ball to top...")
    start_time_2 = time.time()
    Topsensor_status = GPIO.input(Topsensor_pin)
    while Topsensor_status == 1:
        # Going forwards (going up) to the top.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)
        pwm.start(duty)

        Topsensor_status = GPIO.input(Topsensor_pin)

    # Check if the elapsed time has surpassed the timeout.
    elapsed_time = time.time() - start_time_2
    if elapsed_time > drop_timeout:
        print("Timeout reached. Shutting down.")
        # Call a function to shutdown the system.
        exit_program()

    pwm.ChangeDutyCycle(0)  # The ball is already on top.

def backwards():
    display_text("Moving ball down...")
    start_time_3 = time.time()
    Bottomsensor_status = GPIO.input(Bottomsensor_pin)
    while Bottomsensor_status == 1:
        # Going backwards (going down).
        GPIO.output(AIN1Pin, GPIO.LOW)
        GPIO.output(AIN2Pin, GPIO.HIGH)
        pwm.start(duty)

        Bottomsensor_status = GPIO.input(Bottomsensor_pin)

    # Check if the elapsed time has surpassed the timeout.
    elapsed_time = time.time() - start_time_3
    if elapsed_time > raise_halfway_timeout:
        print("Timeout reached. Shutting down.")
        # Call a function to shutdown the system.
        exit_program()

    pwm.ChangeDutyCycle(0)

def sequence():
    display_text("Waiting for 5 minutes till noon")
    start_time = time.time()
    #Flags
    raised_to_middle = False
    raised_to_top = False
    dropped = False
    restarted = False

    # Assume that the ball starts at the bottom position.
    while True:
        elapsed_time = time.time() - start_time
        time_left_noon = raise_top_time - elapsed_time
        time_left_2 = drop_time - elapsed_time
        time_left_3 = restart_time - elapsed_time

        time_left_noon = round(time_left_noon, 2)
        time_left_2 = round(time_left_2, 2)
        time_left_3 = round(time_left_3, 2)

        print(elapsed_time)
        if elapsed_time >= raise_halfway_time and not raised_to_middle:
# Raise ball to halfway
            forward_to_middle()
            raised_to_middle = True

        elif elapsed_time  < raise_top_time and raised_to_middle:
            display_text(f"Waiting for noon. Time left:
{time_left_noon}")

        elif elapsed_time  >= raise_top_time and not raised_to_top:
            forward_to_top()
            raised_to_top = True

        elif elapsed_time < drop_time and raised_to_top:
            display_text(f"Top. Time left: {time_left_2}")

        elif elapsed_time >= drop_time and not dropped:
            backwards()
            dropped = True

        elif elapsed_time < restart_time and dropped:
            display_text("Bottom")

        elif elapsed_time >= restart_time and not restarted:    # Restart
the sequence
            start_time = time.time()
            raised_to_middle = False
            raised_to_top = False
            dropped = False
            restarted = True


def destroy():
    pwm.stop()

if __name__ == '__main__':
    setup()
    try:
        sequence()
    except KeyboardInterrupt:
        destroy()
        GPIO.cleanup()
    pygame.quit()  # Clean up pygame resources.
```

demo_full_speed.py

```
import RPi.GPIO as GPIO
import time
import sys

# Pin definitions.
AIN1Pin = 6
AIN2Pin = 5

Topsensor_pin = 16
Midsensor_pin = 20
Bottomsensor_pin = 21


# Set the times in seconds
raise_halfway_time = 10
raise_top_time = 40
drop_time = 80
restart_time = 120

#The time measurements:
#12.3
#12.9
#25.6

#timeouts
raise_halfway_timeout = 13
raise_top_timeout = 13.6
drop_timeout = 26.3


# define a function to exit the program
def exit_program():
    GPIO.output(AIN1Pin, GPIO.LOW)
    GPIO.output(AIN2Pin, GPIO.LOW)
    sys.exit()

def setup():
    # Setup GPIO.
    GPIO.setmode(GPIO.BCM)

    GPIO.setup(AIN1Pin, GPIO.OUT)
    GPIO.setup(AIN2Pin, GPIO.OUT)
    GPIO.setup(Topsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Midsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Bottomsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)


def forward_to_middle():
    start_time_1 = time.time()

    Midsensor_status = GPIO.input(Midsensor_pin)
    while Midsensor_status == 1:
        # Going forwards (going up) to the middle.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)

        Midsensor_status = GPIO.input(Midsensor_pin)

        # Check if the elapsed time has surpassed the timeout.
        elapsed_time = time.time() - start_time_1
        if elapsed_time > raise_halfway_timeout:
            print("Timeout reached. Shutting down.")
            # Call a function to shutdown the system.
            exit_program()

    #stop
    GPIO.output(AIN1Pin, GPIO.LOW)

def forward_to_top():
    start_time_2 = time.time()
    Topsensor_status = GPIO.input(Topsensor_pin)
    while Topsensor_status == 1:
        # Going forwards (going up) to the top.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)


        Topsensor_status = GPIO.input(Topsensor_pin)

        # Check if the elapsed time has surpassed the timeout.
        elapsed_time = time.time() - start_time_2
        if elapsed_time > raise_top_timeout:
            print("Timeout reached. Shutting down.")
            # Call a function to shutdown the system.
            exit_program()

    #stop
    GPIO.output(AIN1Pin, GPIO.LOW)

def backwards():
    start_time_3 = time.time()
    Bottomsensor_status = GPIO.input(Bottomsensor_pin)
    while Bottomsensor_status == 1:
        # Going backwards (going down).
        GPIO.output(AIN1Pin, GPIO.LOW)
        GPIO.output(AIN2Pin, GPIO.HIGH)

        Bottomsensor_status = GPIO.input(Bottomsensor_pin)

        # Check if the elapsed time has surpassed the timeout.
        elapsed_time = time.time() - start_time_3
        if elapsed_time > drop_timeout:
            print("Timeout reached. Shutting down.")
            # Call a function to shutdown the system.
            exit_program()

    GPIO.output(AIN2Pin, GPIO.LOW)

def sequence():
    GPIO.output(AIN1Pin, GPIO.LOW)
    GPIO.output(AIN2Pin, GPIO.LOW)
    #Flags
    raised_to_middle = False
    raised_to_top = False
    dropped = False
    restarted = False


    # Assume that the ball starts at the bottom position.
    start_time = time.time()
    while True:
        elapsed_time = time.time() - start_time
        print("Elapsed Time:", elapsed_time)
        #print("Raised to Middle:", raised_to_middle)
        #print("Raised to Top:", raised_to_top)
        #print("Dropped:", dropped)
        #print("Restarted:", restarted)
        #print(elapsed_time)
        if elapsed_time >= raise_halfway_time and not raised_to_middle:
# Raise ball to halfway
            print("Triggering forward_to_middle()")
            forward_to_middle()
            raised_to_middle = True

        if elapsed_time  >= raise_top_time and not raised_to_top:
            print("Triggering forward_to_top()")

            forward_to_top()
            raised_to_top = True

        if elapsed_time >= drop_time and not dropped:
            print("Triggering backwards()")
            backwards()
            dropped = True

        if elapsed_time >= restart_time and not restarted:   # Restart
the sequence
            print("Triggering restart")
            start_time = time.time()
            raised_to_middle = False
            raised_to_top = False
            dropped = False
            restarted = True


if __name__ == '__main__':
    setup()
    try:
        sequence()
    except KeyboardInterrupt:
        GPIO.cleanup()
```

motor_full_speed.py

```
import RPi.GPIO as GPIO


# Pin definitions
AIN1Pin = 6
AIN2Pin = 5

GPIO.setwarnings(False)

def setup():
    # Set up GPIO
    GPIO.setmode(GPIO.BCM)

    # Set up the pins
    GPIO.setup(AIN1Pin, GPIO.OUT)
    GPIO.setup(AIN2Pin, GPIO.OUT)


def forwards():
    # Going forwards (going up)

    GPIO.output(AIN1Pin, GPIO.HIGH)
    GPIO.output(AIN2Pin, GPIO.LOW)

def backwards():
    # Going backwards (going down)
    GPIO.output(AIN1Pin, GPIO.LOW)
    GPIO.output(AIN2Pin, GPIO.HIGH)


def sequence():
    while True:
        backwards()

def destroy():
    # Clean up the GPIO pins
    GPIO.cleanup()

if __name__ == '__main__':
    # Set up the GPIO pins
    setup()
    try:
        # Run the sequence
        sequence()
    except KeyboardInterrupt:
        # Clean up the GPIO pins on interrupt
        destroy()
```

motor_half_speed.py

```
import RPi.GPIO as GPIO


# Pin definitions
pwmPin_1 = 6
pwmPin_2 = 5

GPIO.setwarnings(False)

# Duty cycle and frequency
duty = 50
frequency = 50

def setup():
    # Set up GPIO
    GPIO.setmode(GPIO.BCM)

    # Set up the pins
    GPIO.setup(pwmPin_1, GPIO.OUT)
    GPIO.setup(pwmPin_2, GPIO.OUT)

    # Set up the PWM channel
    global pwm_1
    global pwm_2
    pwm_1 = GPIO.PWM(pwmPin_1, frequency)
    pwm_2 = GPIO.PWM(pwmPin_2, frequency)
def forwards():
    # Going forwards (going up)

    pwm_1.start(duty)
    pwm_2.start(0)
def backwards():
    # Going backwards (going down)
    pwm_2.start(duty)
    pwm_1.start(0)
def sequence():
    while True:
        forwards()

def destroy():
    # Clean up the GPIO pins
    GPIO.cleanup()

if __name__ == '__main__':
    # Set up the GPIO pins
    setup()
    try:
        # Run the sequence
        sequence()
    except KeyboardInterrupt:
        # Clean up the GPIO pins on interrupt
        destroy()
```

testing.py

```
import RPi.GPIO as GPIO


# Pin definitions.
pwmPin = 26
AIN1Pin = 5
AIN2Pin = 6

Topsensor_pin = 16
Midsensor_pin = 20
Bottomsensor_pin = 21

duty = 50
frequency = 50

def setup():
    # Setup GPIO.
    GPIO.setmode(GPIO.BCM)

    GPIO.setup(pwmPin, GPIO.OUT)
    GPIO.setup(AIN1Pin, GPIO.OUT)
    GPIO.setup(AIN2Pin, GPIO.OUT)
    GPIO.setup(Topsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Midsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Bottomsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

    global pwm
    pwm = GPIO.PWM(pwmPin, frequency)

def forward():
    Topsensor_status = GPIO.input(Topsensor_pin)
    while Topsensor_status == 1:
        # Going forwards (going up) to the top.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)
        pwm.start(duty)

        Topsensor_status = GPIO.input(Topsensor_pin)


        # Check for the middle sensor trigger
        if GPIO.input(Midsensor_pin) == 0:
            print("Middle sensor triggered")

    pwm.ChangeDutyCycle(0)  # The ball is already on top.

def backwards():
    Bottomsensor_status = GPIO.input(Bottomsensor_pin)
    while Bottomsensor_status == 1:
        # Going backwards (going down).
        GPIO.output(AIN1Pin, GPIO.LOW)
        GPIO.output(AIN2Pin, GPIO.HIGH)
        pwm.start(duty)
        Bottomsensor_status = GPIO.input(Bottomsensor_pin)

        # Check for the middle sensor trigger
        if GPIO.input(Midsensor_pin) == 0:
            print("Middle sensor triggered")

    pwm.ChangeDutyCycle(0)

def sequence():
    # Assume that the ball starts at the bottom position.
    while True:
        forward()
        backwards()

def destroy():
    pwm.stop()

if __name__ == '__main__':
    setup()
    try:
        sequence()
    except KeyboardInterrupt:
        destroy()
        GPIO.cleanup()
```

top_to_bottom_full_speed.py

```
import RPi.GPIO as GPIO


# Pin definitions.
AIN1Pin = 6
AIN2Pin = 5

Topsensor_pin = 16
Midsensor_pin = 20
Bottomsensor_pin = 21


def setup():
    # Setup GPIO.
    GPIO.setmode(GPIO.BCM)

    GPIO.setup(AIN1Pin, GPIO.OUT)
    GPIO.setup(AIN2Pin, GPIO.OUT)
    GPIO.setup(Topsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Midsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Bottomsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)



def forward():
    Topsensor_status = GPIO.input(Topsensor_pin)
    while Topsensor_status == 1:
        # Going forwards (going up) to the top.
        GPIO.output(AIN1Pin, GPIO.HIGH)
        GPIO.output(AIN2Pin, GPIO.LOW)


        Topsensor_status = GPIO.input(Topsensor_pin)


        # Check for the middle sensor trigger
        if GPIO.input(Midsensor_pin) == 0:
            print("Middle sensor triggered")


def backwards():
    Bottomsensor_status = GPIO.input(Bottomsensor_pin)
    while Bottomsensor_status == 1:
        # Going backwards (going down).
        GPIO.output(AIN1Pin, GPIO.LOW)
        GPIO.output(AIN2Pin, GPIO.HIGH)

        Bottomsensor_status = GPIO.input(Bottomsensor_pin)

        # Check for the middle sensor trigger
        if GPIO.input(Midsensor_pin) == 0:
            print("Middle sensor triggered")


def sequence():
    # Assume that the ball starts at the bottom position.
    while True:
        forward()
        backwards()



if __name__ == '__main__':
    setup()
    try:
        sequence()
    except KeyboardInterrupt:
        GPIO.cleanup()
```

top_to_bottom_half_speed.py

```
import RPi.GPIO as GPIO


# Pin definitions.
pwmPin_1 = 6
```

```
pwmPin_2 = 5

Topsensor_pin = 16
Midsensor_pin = 20
Bottomsensor_pin = 21

duty = 50
frequency = 50

def setup():
    # Setup GPIO.
    GPIO.setmode(GPIO.BCM)

    GPIO.setup(pwmPin_1, GPIO.OUT)
    GPIO.setup(pwmPin_2, GPIO.OUT)
    GPIO.setup(Topsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Midsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(Bottomsensor_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

    # Set up the PWM channel
    global pwm_1
    global pwm_2
    pwm_1 = GPIO.PWM(pwmPin_1, frequency)
    pwm_2 = GPIO.PWM(pwmPin_2, frequency)

def forward():
    Topsensor_status = GPIO.input(Topsensor_pin)
    while Topsensor_status == 1:
        # Going forwards (going up) to the top.
        pwm_1.start(duty)
        pwm_2.start(0)

        Topsensor_status = GPIO.input(Topsensor_pin)


        # Check for the middle sensor trigger
        if GPIO.input(Midsensor_pin) == 0:
            print("Middle sensor triggered")

    pwm_1.ChangeDutyCycle(0)  # The ball is already on top.
    pwm_2.ChangeDutyCycle(0)  # The ball is already on top.
def backwards():
    Bottomsensor_status = GPIO.input(Bottomsensor_pin)
    while Bottomsensor_status == 1:
        # Going backwards (going down).
        pwm_2.start(duty)
        pwm_1.start(0)

        Bottomsensor_status = GPIO.input(Bottomsensor_pin)

        # Check for the middle sensor trigger
        if GPIO.input(Midsensor_pin) == 0:
            print("Middle sensor triggered")

    pwm_1.ChangeDutyCycle(0)
    pwm_2.ChangeDutyCycle(0)

def sequence():
    # Assume that the ball starts at the bottom position.
    while True:
        forward()
        backwards()

def destroy():
    pwm_1.stop()
    pwm_2.stop()

if __name__ == '__main__':
    setup()
    try:
        sequence()
    except KeyboardInterrupt:
        destroy()
        GPIO.cleanup()
```