ABSTRACT

The concept of digital twin technology is one that is just starting to gain traction in business and, more recently, academics. Industry 4.0 has grown more quickly thanks to conceptual advances, especially in the manufacturing sector. There are many definitions of the Digital Twin, but the best described as the seamless integration of data going either way between a physical and virtual machine. We describe the issues, uses, and supporting technologies for artificial intelligence, the internet of things (IoT), and digital twins. A categorical review of recent studies is produced after a review of publications on digital twins. Manufacturing, healthcare, and smart cities have been categorised by study fields in the review, which also discusses a variety of publications that reflect these domains and the current level of research. The ability to unleash various potentials of virtual product creation has increased with the use and sophistication of simulation models. While simulation software has gotten increasingly powerful, attempts to maximize the potential benefits of product simulations are increasingly centered on the input data that is now accessible. More accurate simulation models that ultimately produce superior products can be created if it is able to incorporate use phase data from real-world applications into the simulation models or use this data to check simulation findings. This project enables the real time simulation and stress analysis of a screw jack in working condition.

Keywords:

Digital Twins, Applications, Enabling Technologies, Industrial Internet of Things (IIoT), Internet of Things (IoT), Machine Learning, Deep Learning, Literature Review

INDEX

CHAPTER 1

INTRODUCTION

1.1 Introduction to Digital Twin Technology

A virtual reproduction of a physical system or process, such as a machine, a structure, or an entire city, is known as a digital twin. It is a digital model that may be used to mimic and examine the functionality and behaviour of the system in the actual world. Digital twins have been around for a while, but recent developments in technology like the Internet of Things (IoT), artificial intelligence, and cloud computing have made it simpler to build and use them.

The physical system's physical and functional properties are captured by the digital twin, which is created to be an exact digital replica of the physical system. It is regularly updated with real-time data from sensors and other sources, which enables it to correctly depict the actual status of the physical system.

Users can learn more about the operation of the physical system and pinpoint areas for improvement by examining the data gathered from the digital twin.

Many different businesses, including manufacturing, healthcare, and transportation, can benefit from the use of digital twins. Digital twins, for instance, can be utilised to streamline production processes, raise product quality, and decrease downtime in the manufacturing industry. To model the human body and provide individualised treatment strategies in healthcare, digital twins can be used. Digital twins can be utilised in the transportation sector to track, improve, and lessen traffic congestion.

The digital twin is an effective technique that can help businesses streamline their processes, cut costs, and increase efficiency.

Fig 1.1 Reference of Digital Twin

1.2 Process of Digital Twin

A virtual representation created to faithfully represent a physical object is called a digital twin. The object being researched, such as a wind turbine, is equipped with a variety of sensors that are connected to key functioning regions. These sensors generate information about a variety of performance characteristics of the physical device, including energy output, temperature, environmental conditions, and more. The processing system then applies this information to the digital copy.

The virtual model can then be used to run simulations, analyse performance problems, and suggest potential modifications in order to get useful insights that can then be applied to the real physical device

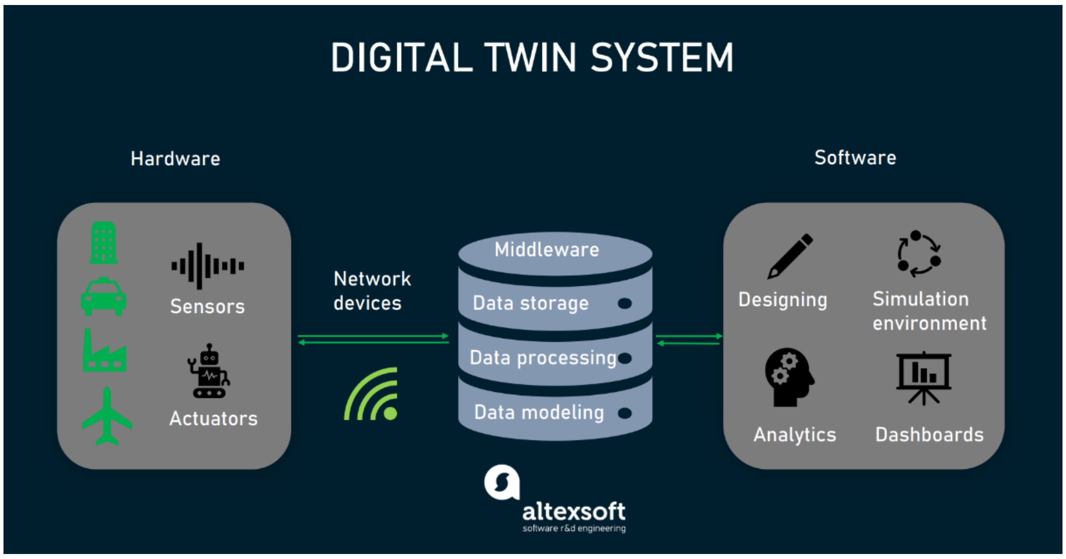


Fig **Error! No text of specified style in document.**.1 Architecture of Digital Twin system.

1.3 Simulations vs. digital twins

Although both simulations and digital twins use digital models to replicate a system's various processes, a digital twin is actually a virtual environment, making it far more rich for research. The primary distinction between digital twin and simulation is one of scale: While a simulation typically studies a single process, a digital twin can run an unlimited number of useful simulations to study multiple processes.

The distinctions do not stop there. For example, simulations rarely benefit from real-time data. However, digital twins are built around a two-way information flow that begins when object sensors provide relevant data to the system processor and continues when the processor's insights are shared back with the origin source user

Digital twins are able to study more issues from far more vantage points than standard simulations can because they have better and constantly updated data related to a wide range of areas, combined with the added computing power that comes with a virtual environment — with greater ultimate potential to improve products and processes

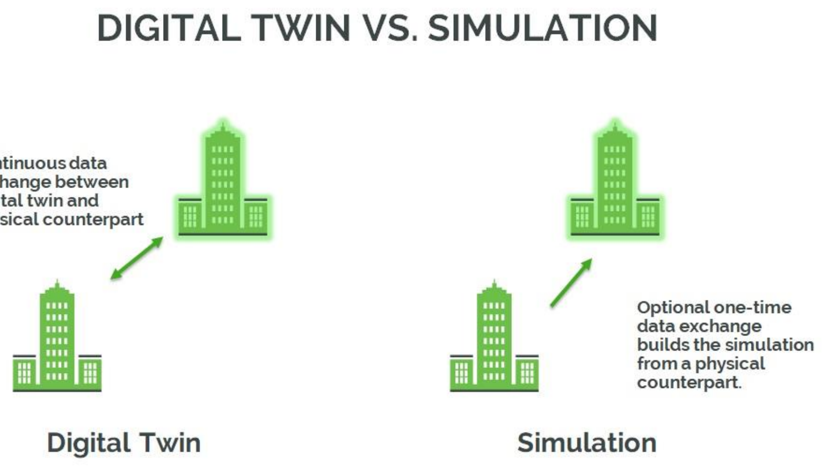

DIGITAL TWIN VS. SIMULATION
Continuous data exchange between digital twin and physical counterpart
Optional one-time data exchange builds the simulation from a physical counterpart.
Digital Twin
Simulation

Fig **Error! No text of specified style in document.**.2 Simulation vs Digital Twin

1.4 Digital twin classifications

Depending on the extent of product magnification, there are many sorts of digital twins. The primary distinction between these twins is their field of application. Several sorts of digital twins frequently coexist within a system or process. Let's look over the many sorts of digital twins to understand the differences and how they are used

- **Parts twins/Component twins**

  Component twins are the fundamental unit of a digital twin, representing the smallest example of a working component. Components twins are roughly the same thing, except they refer to significantly less important components.

- **Twins of assets**

  An asset is formed when two or more components work together to form a unit. Asset twins allow you to investigate the interaction of those components, resulting in a wealth of performance data that can be analysed and transformed into meaningful insights.

- **System twins or Unit twins**

  The next degree of magnification involves system or unit twins, which allow you to see how various assets interact to build a fully functional system. System twins provide visibility into asset interactions and may identify performance improvements.

- **Twins in the process**

  Process twins, the macro level of magnification, show how systems interact to produce an entire manufacturing complex. Are all of those systems synced to run at top efficiency, or will delays in one system have an impact on others? Process twins can assist in determining the specific timing schemes that influence overall effectiveness.

1.5 The advantages and disadvantages of digital twins

- **Improved R&D**

  The utilisation of digital twins allows for more effective product research and creation, with an abundance of data generated concerning expected performance outcomes. This data can lead to insights that can help businesses make necessary product improvements before going into production.

- **Increased effectiveness**

  Even after a new product has entered production, digital twins can assist in mirroring and monitoring production systems in order to achieve and maintain optimal efficiency throughout the manufacturing process.

- **Product obsolescence**

  Digital twins can even assist producers in determining what to do with products that have reached the end of their product lifecycle and require final processing, such as recycling or other measures. They can identify which product materials can be used by employing digital twins

1.6 Sensors

1.6.1 Introduction to sensors

A sensor is a device that detects and responds to some type of input from the physical environment. The input can be light, heat, motion, moisture, pressure or any number of other environmental phenomena. The output is generally a signal that is converted to a human-readable display at the sensor location or transmitted electronically over a network for reading or further processing.

Sensors play a pivotal role in the internet of things (IoT). They make it possible to create an ecosystem for collecting and processing data about a specific environment so it can be monitored, managed and controlled more easily and efficiently. IoT sensors are used in homes, out in the field, in automobiles, on airplanes, in industrial settings and in other environments. Sensors bridge the gap between the physical world and logical world, acting as the eyes and ears for a computing infrastructure that analyses and acts upon the data collected from the sensors.

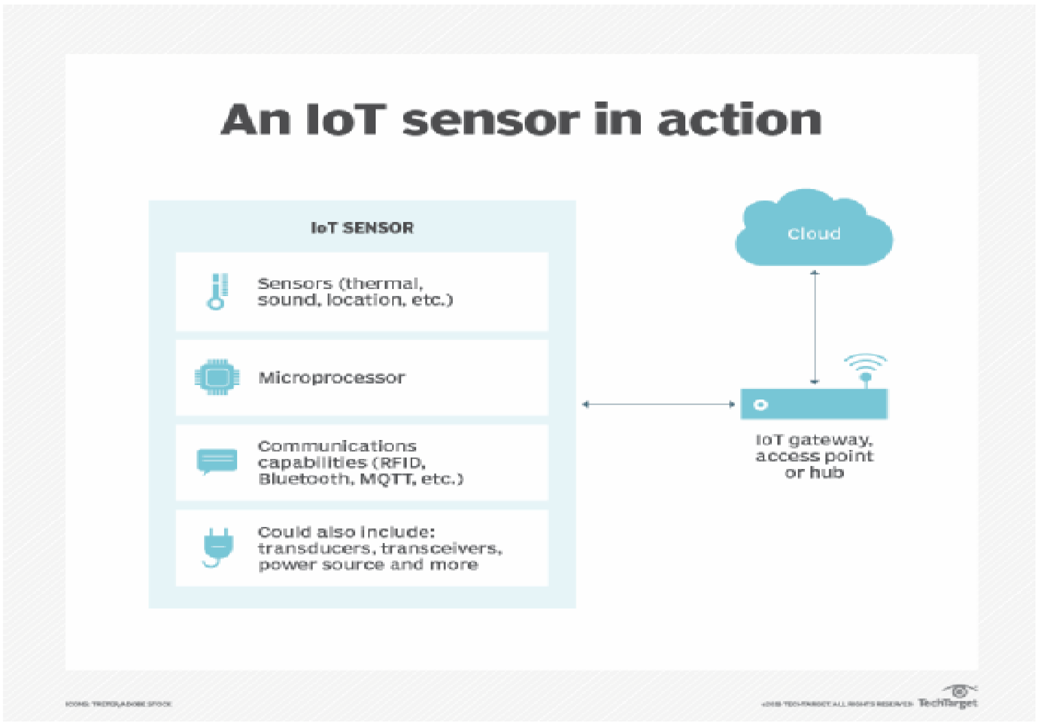

An IoT sensor in action
IoT SENSOR
Sensors (thermal, sound, location, etc.)
Microprocessor
Communications capabilities (RFID, Bluetooth, MQTT, etc.)
Could also include: transducers, transceivers, power source and more
Cloud
IoT gateway, access point or hub
TechTarget

Fig 1.4

1.6.2 How does a Sensor work

A sensor converts the physical action to be measured into an electrical equivalent and processes it so that the electrical signals can be easily sent and further processed. The sensor can output whether an object is present or not present (binary) or what measurement value has been reached (analog or digital).

Components of a sensor

- The sensing section contains the sensor itself which is based on a particular technology. The variety of technologies means you can select a sensor technology which fits your application.
- The processing circuitry converts the physical variable into an electrical variable.
- The signal output contains the electronics connected to a control system.

1.6.3 Sensors Categories based on material

Types of Sensors:

- **Inductive sensors**

  Inductive sensors generate an electromagnetic field. This in turn generates eddy currents in objects made of metal. The sensor detects this change.

  Fig 1.5

- **Capacitive sensors**

  Capacitive sensors generate a capacitive measuring field. An entering object results in a change to the measuring field. The sensor responds to this change.

  Fig 1.6

- **Photoelectric sensors**

  Photoelectric sensors (light curtains) always consist of an emitter and a receiver. There are diffuse, retro-reflective and through-beam types.

  Ultrasonic sensors

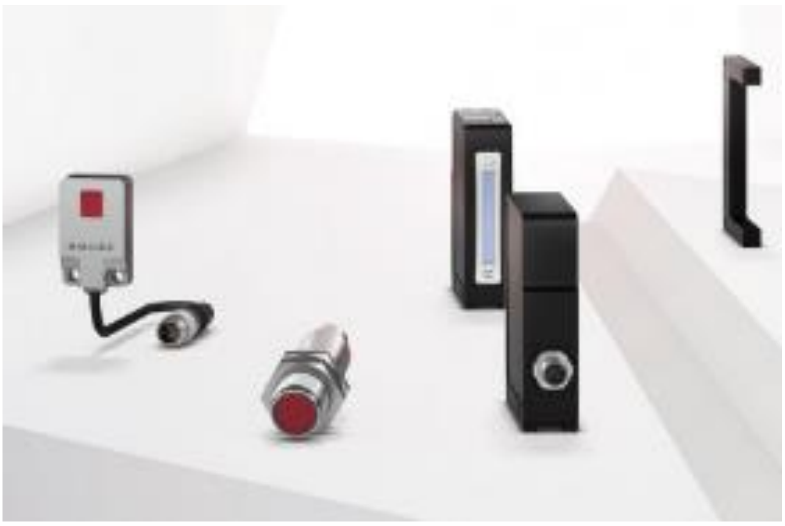
  

  Fig 1.7

- **Ultrasonic sensors**

  Ultrasonic sensors send out a sound pulse in the inaudible range. The echo from the object is processed.

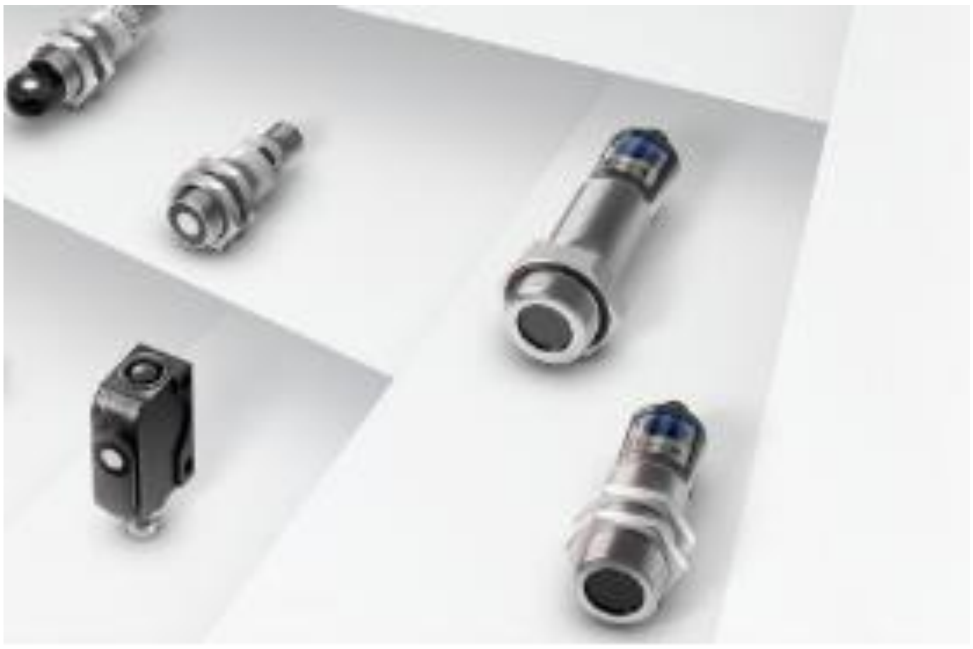
  

  Fig 1.8

1.6.4 Sensor Categories based on Functionality

Numerous categories can be used to group sensors. To categorize them as either active or passive is a typical strategy. An active sensor needs an outside power source to react to environmental input and produce output. For instance, to provide meteorological information on the Earth's atmosphere, sensors used in weather satellites frequently need some form of energy. On the other hand, a passive sensor does not require an external power source to pick up environmental input. It draws its energy from the environment, using things like thermal energy or light. The mercury-based glass thermometer is an excellent example. Temperature changes allow the mercury to expand and contract, affecting the level in the glass.

Some sensor types, including seismic and infrared light sensors, come in both active and passive varieties. Which sort of sensor is ideal for the application often depends on the environment in which it is deployed.Some sensor types, including seismic and infrared light sensors, come in both active and passive varieties. The appropriate sort of sensor for an application is often determined by the environment in which it is installed. Depending on the kind of output the sensors provide, analogness or digitalness can also be used to categorise sensors. Environmental input is transformed into output analogue signals using analogue sensors, which are continuous and variable. An illustration would be the thermocouples found in gas hot water heaters.

Based on the kind of output the sensors provide, another method to categorise sensors is by whether they are analogue or digital. Environmental input is transformed into output analogue signals, which are continuous and variable, via analogue sensors. An excellent illustration of an analogue sensor is the thermocouple, which is used in gas hot water heaters. The thermocouple is continuously heated by the pilot light of the water heater. The thermocouple cools if the pilot light goes out, providing a separate analogue signal that says the gas should be turned off.

Unlike analogue sensors, which transform environmental information into discrete digital signals that are delivered in a binary format, digital sensors convert the environment. (1s and 0s). All industries now use a lot of digital sensors, which have mostly replaced analogue ones.

Sensors are also commonly categorized by the type of environmental factors they monitor. Here are some common examples

- **Accelerometer.** This type of sensor detects changes in gravitational acceleration, making it possible to measure tilt, vibration and, of course, acceleration sensors are used in a wide range of industries, from consumer electronics to professional sports to aerospace and aviation.
- **Chemical**. Chemical sensors identify a particular chemical within a medium. (gas, liquid or solid). Chemical sensors can be used to measure a variety of things, including soil nutrient levels in crop fields, smoke or carbon monoxide levels in enclosed spaces, pH levels in bodies of water, the presence of alcohol on someone's breath, and many other things. For instance, the gasoline-to-oxygen ratio will be monitored by an oxygen sensor in a car's emission control system, typically through a chemical reaction that produces voltage. When the mixture is not optimal, a computer in the engine compartment reads the voltage and changes the ratio.
- **Humidity**. These sensors have the ability to measure the amount of water vapour in the air and calculate relative humidity. Since relative humidity is influenced by air temperature, humidity sensors frequently provide temperature values as well. The sensors are utilised in numerous contexts and industries, including as data centers, meteorology, agriculture, manufacturing, and HVAC (heating, ventilation, and air conditioning). (HVAC).
- **Level**. The level of a physical substance, such as water, gasoline, coolant, grain, fertilizer, or garbage, can be determined using a level sensor. For instance, drivers depend on their gas level sensors to prevent being stranded on the side of the road. Systems for warning of tsunamis also employ level sensors.
- **Motion**. Motion detectors can be used to control lights, cameras, parking gates, water faucets, security systems, automated door openers, and a variety of other systems. They can detect physical movement in a specified area (the field of detection). The sensors usually emit some form of energy, such as microwaves, ultrasonic waves, or laser beams, and are able to detect when the energy flow is obstructed by an object that moves in its path.
- **Optical**. Optical sensors, also known as photosensors, are capable of detecting light waves at many wavelengths, including ultraviolet, visible, and infrared light. Optical

sensors are widely utilised in a variety of systems, including smartphones, robotics, Blu-ray players, home security systems, and medical equipment.

- **Pressure**. These sensors are widely employed in machinery, vehicles, airplanes, HVAC systems, and other situations to measure the pressure of a liquid or gas. They are crucial in meteorology because they measure atmospheric pressure. Pressure sensors can also be used to monitor the flow of gases or liquids, frequently in order to control the flow.
- **Proximity**. The presence of an object or the distance between objects is detected using proximity sensors. In addition to many other situations, proximity monitors are employed in elevators, assembly lines, parking lots, retail establishments, cars, and robotics.
- **Temperature**. These sensors are capable of measuring the temperature of any target media, including air, liquid, and gas. Appliances, machinery, aircraft, autos, computers, greenhouses, farms, thermostats, and a broad variety of other objects and surroundings all employ temperature sensors.

1.6.5 Advantages of Sensors

Deploying sensors and sensing technology has multiple benefits, including predictive and preventive maintenance. They not only ensure that measurement data is transmitted faster, but also increase accuracy, thereby improving process control, and enhancing asset health. A new breed of sensors are capable of wired and wireless transmission, providing a real-time, continuous data feed from assets and processes. This empowers executives with a more holistic view of a process plant. Businesses that leverage sensors are more connected, secure, and agile than ever before. Improved data capture sensitivity, nearly lossless transmission, and ongoing, real-time analysis are some of sensors' main benefits. Processes are active and are carried out as efficiently as possible thanks to real-time feedback and data analytics services.

The development of sensing technology over time has led to the smart and intelligent sensors of today. Smart sensors have electrical circuits, allowing them to take measurements and output values as digital data, in contrast to standard analogue sensors, which lack active components. These sensors incorporate embedded microprocessors and a signal converter on top of a variety of sensing components. The ability to self-test, self-validate, self-adapt, and self-identify are just a few of the innately intelligent tasks that intelligent sensors are capable of performing. They can recognise conditions, manage a variety of conditions, and comprehend process needs.

1.7 Introduction to Visual Studio

Microsoft's Visual Studio Code, sometimes known as VS Code, is a source-code editor for Windows, Linux, and macOS that uses the Electron Framework. Debugging support, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git are among the features. Users can modify the theme, keyboard shortcuts, preferences, and add functionality by installing extensions.

The most widely used development environment tool among 71,010 respondents in the Stack Overflow 2022 development Survey was Visual Studio Code, with 74.48% saying they use it.

Fig 1.9 Visual Studio and Visual Studio code

The word "radar" is an acronym that stands for Radio Detection and Ranging. This technology has been in use since before World War II to track the speed, position, and direction of ships and aircraft.

Radar systems emit radio waves, a type of electromagnetic energy that can be sent into the air. These signals travel at the speed of light. These waves reflect back from objects in the radar's path and are processed by the radar receiver. These returned pulses enable the radar to determine the distance *and position* of objects, relative to the radar system's position.

1.7.1 Features of Visual Studio code

A source-code editor called Visual Studio Code works with many different programming languages, such as C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, and Rust.

Based on the Electron framework, which is used to create Node.js web apps that employ the Blink layout engine, it is used to create mobile applications. The editor component (codenamed "Monaco") used in Azure DevOps is also utilised in Visual Studio Code. (formerly called Visual Studio Online and Visual Studio Team Services). The majority of popular programming languages have minimal support out of the box in Visual Studio Code. This fundamental support consists of configurable snippets, code folding, bracket matching, and syntax highlighting. Along with debugging support for Node.js, Visual Studio Code also comes with IntelliSense for JavaScript, TypeScript, JSON, CSS, and HTML. It enables users to open one or more directories, which may subsequently be saved in workspaces for later usage in place of a project system. As a result, it can function as a language-neutral code editor for any language. It supports a wide variety of programming languages, each with its own set of capabilities. Through the settings, unwanted files and folders can be excluded from the project tree.

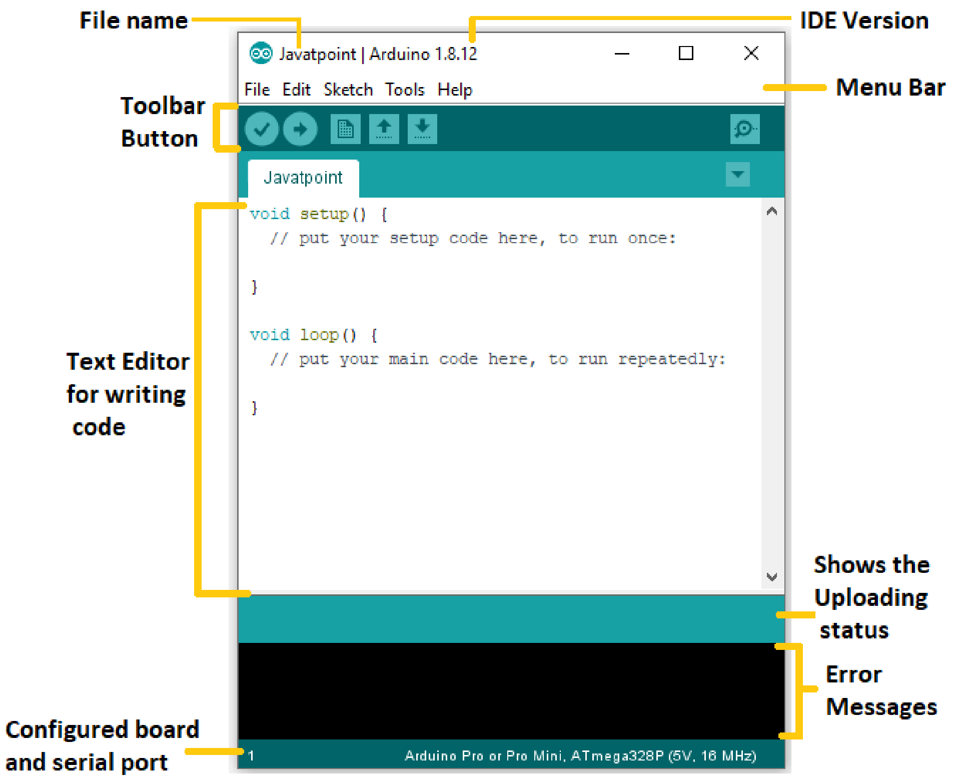
Many aspects of Visual Studio Code can be accessible using the command palette instead of through menus or the user interface. Extensions are add-ons for Visual Studio Code that are accessible through a single repository. This contains language support and editor additions. The capacity to design extensions that add support for new languages, themes, and other features is a noteworthy feature.Visual Studio Code comes with a built-in tool for source control. Users can access version control settings and see changes made to the current project by using a dedicated tab in the menu bar. Any compatible version control system must be linked to Visual Studio Code in order to use the feature. (Git, Apache Subversion, Perforce, etc.). This enables users to push and pull requests from within the Visual Studio Code software, as well as create repositories.As a result of its numerous FTP extensions, Visual Studio Code can be used as a cost-free substitute for web development. Without having to download any additional software, code may be synced between the editor and the server.

1.8 Introduction to Arduino

Arduino IDE is an open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as **Windows, Mac OS X, and Linux**. It supports the programming languages C and C++. Here, IDE stands for **Integrated Development Environment**.

The program or code written in the Arduino IDE is often called as sketching. We need to connect the Genuino and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ino.'

The Arduino IDE will appear as:


File name
IDE Version
Javatpoint | Arduino 1.8.12
File Edit Sketch Tools Help
Menu Bar
Toolbar Button
Javatpoint
Text Editor for writing code
Shows the Uploading status
Error Messages
Configured board and serial port
1
Arduino Pro or Pro Mini, ATmega328P (5V, 16 MHz)

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Fig 1.10

Extensions are add-ons for Visual Studio Code that are accessible through a single repository. This contains language support and editor additions. The capacity to design extensions that add support for new languages, themes, and other features is a noteworthy feature.

1.8.1 Arduino UNO Board

A microcontroller board called Arduino UNO is based on the ATmega328P. It contains 6 analogue inputs, a 16 MHz ceramic resonator, 14 digital input/output pins (six of which can be used as PWM outputs), a USB port, a power jack, an ICSP header, and a reset button. It comes with everything needed to support the microcontroller; to get started, just plug in a USB cable, an AC-to-DC adapter, or a battery. You can experiment with your UNO without being overly concerned that you'll make a mistake; in the worst case, you can replace the chip for a few dollars and start over.

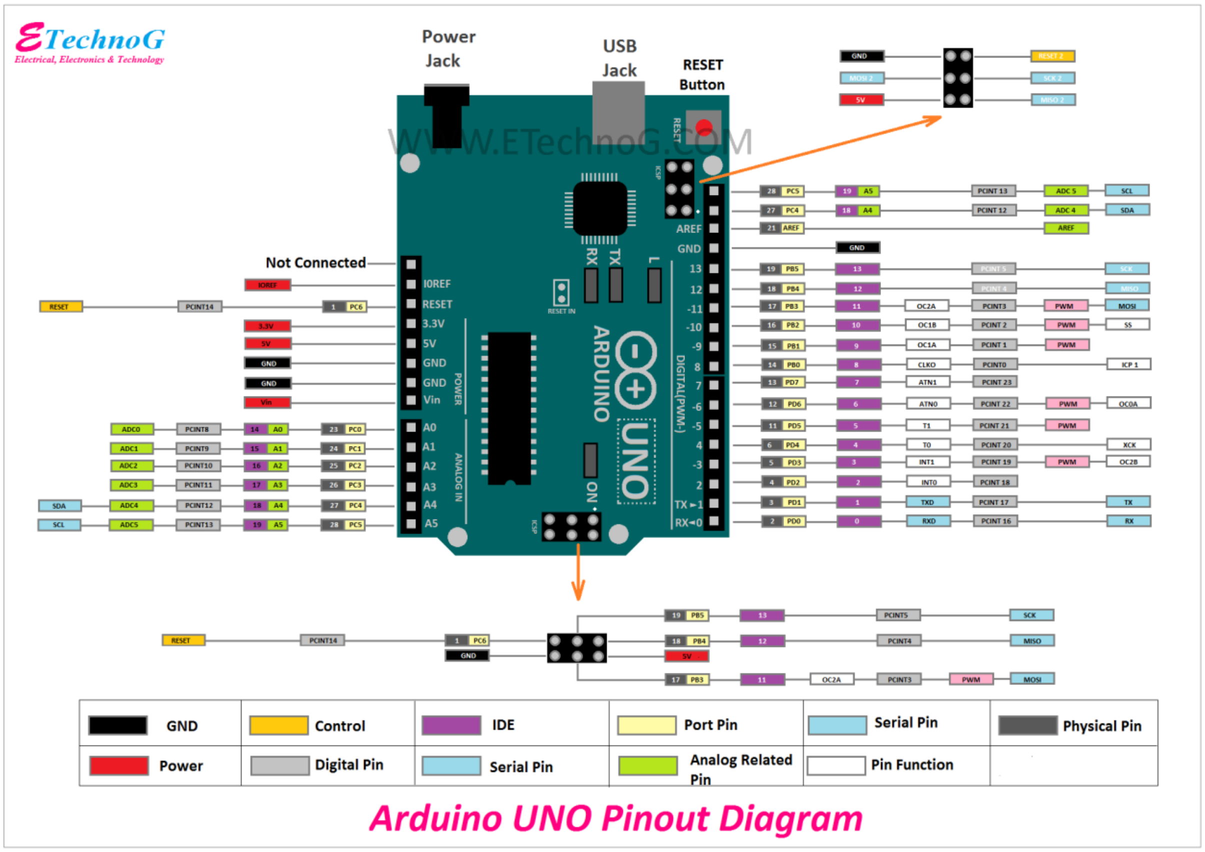


Fig 1.11

This is a description of what you will see when you look at the board from the top down; areas of the board that you might interact with during regular use are highlighted.

In a clockwise direction from the top centre:

Orange Analog Reference pin

(Light Green) Digital Ground

Green digital pins 2–13

Dark green Digital Pins 0-1/Serial In/Out - TX/RX If serial communication is also being used (for example, via Serial.begin), then these pins cannot be used for digital input and output (digitalRead and digitalWrite).

Reset Button - S1 (dark blue)

Blue-green In-Circuit Serial Programmer

Pins 0–5 for analogue in (light blue)

Ground and power pins (grounds: light orange, power: orange)

(9–12 VDC) External Power Supply In - X1 (pink)

Toggles Place the jumper on the two pins that are closest to the selected source for external and USB power. - Purple SV1

USB (used to power the board, to upload sketches to the board, and for serial communication between the board and the computer) (yellow)

1.8.2 Microcontroller

ATmega328P microcontrollers (found on modern circuit boards)

14 digital I/O pins, 6 of which are used to produce PWM.

Pins for analogue input: 6 (DIP) or 8 (SMD).

DC I/O pin current: 40 mA

32 KB Flash Memory; 2 KB SRAM

(Used on the majority of Arduino Diecimila and early Duemilanove) EEPROM: 1 KB

ATmega168

14 digital I/O pins, 6 of which are used to produce PWM.

Pins for analogue input: 6 (DIP) or 8 (SMD).

DC I/O pin current: 40 mA

16 KB Flash Memory

1 KB SRAM

512-byte

ATmega8 EEPROM (used on some older boards)

14 digital I/O pins, 3 of which are used to produce PWM.

6 DC Analog Input Pins I/O pin current: 40 mA

8 KB of flash memory

1 KB SRAM

1.9 Java Script

Along with HTML and CSS, the programming language JavaScript, sometimes known as JS, is one of the foundational elements of the World Wide Web. 98% of websites will utilize JavaScript on the client side by 2022 to control webpage behaviour, frequently integrating third-party libraries. A dedicated JavaScript engine is available in every major web browser and is used to run the code on users' devices.

JavaScript is an ECMAScript-compliant high-level, frequently just-in-time compiled language. It features first-class functions, prototype-based object orientation, and dynamic typing. It supports event-driven, functional, and imperative programming paradigms and is a multi-paradigm. For working with text, dates, regular expressions, shared data structures, and the Document Object Model (DOM), it includes application programming interfaces (APIs).

Fig 1.12

There are no input/output (I/O) features like networking, storage, or graphics capabilities in the ECMAScript standard. In reality, JavaScript I/O APIs are offered by the web browser or another runtime system.Originally only used in web browsers, JavaScript engines are now essential parts of some servers and a wide range of applications. Node.js is the most widely used runtime system for this application. Even while Java and JavaScript share the same name, syntax, and standard libraries, the two programming languages are separate and have very different designs.

For their client-side scripting, more than 80% of websites employ a third-party JavaScript library or web framework. Over 75% of websites utilize jQuery, making it by far the most popular client-side library. Twitter currently uses the React library, which was developed by Facebook for its website and then made available as open source. Similar to this, Google's Angular framework, which it developed for its websites like YouTube and Gmail, is now an open-source project that other people utilize. On the other hand, websites that solely rely on ordinary JavaScript capabilities and don't use any libraries or frameworks are referred to as "Vanilla JS" websites.

1.9.1 Features of Java Script

Structured and organized

Most of the structured programming syntax from C, including if statements, while loops, switch statements, do while loops, etc., is supported by JavaScript. Scoping is one example of a partial exception; prior to ECMAScript 2015, block scoping was only available in JavaScript and was added with the let and const keywords. Expressions and statements are distinct in JavaScript, as they are in C. Automatic semicolon insertion, which allows semicolons (which terminate statements) to be omitted, is one syntactic distinction from C.

Weakly Typed

Because JavaScript is loosely typed, depending on the operation employed, some types are implicitly cast.

Values are cast to strings like the following:

- Strings are left as-is
- Numbers are converted to their string representation
- Arrays have their elements cast to strings after which they are joined by commas (,)
- Other objects are converted to the string [object Object] where Object is the name of the constructor of the object

Values are cast to numbers by casting to strings and then casting the strings to numbers. These processes can be modified by defining to string and value Of functions on the prototype for string and number casting respectively.

JavaScript has received criticism for the way it implements these conversions as the complexity of the rules can be mistaken for inconsistency. For example, when adding a number to a string, the number will be cast to a string before performing concatenation, but when subtracting a number from a string, the string is cast to a number before performing subtraction.

Typing

JavaScript is dynamically typed like most other scripting languages. A type is associated with a value rather than an expression. For example, a variable initially bound to a number may be reassigned to a string. JavaScript supports various ways to test the type of objects, including duck typing.

Run-time evaluation

JavaScript includes an `eval` function that can execute statements provided as strings at run-time.

Object-orientation (prototype-based)

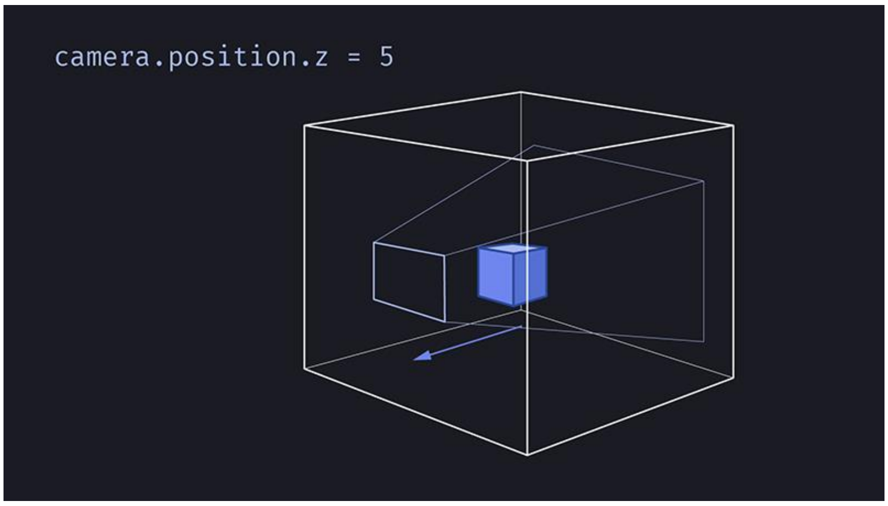
Prototypal inheritance in JavaScript is described by Douglas Crockford as:

You make prototype objects, and then ... make new instances. Objects are mutable in JavaScript, so we can augment the new instances, giving them new fields and methods. These can then act as prototypes for even newer objects. We don't need classes to make lots of similar objects... Objects inherit from objects.

In JavaScript, an object is an associative array, augmented with a prototype (see below); each key provides the name for an object property, and there are two syntactical ways to specify such a name: dot notation (`obj.x = 10`) and bracket notation (`obj['x'] = 10`). A property may be added, rebound, or deleted at run-time. Most properties of an object (and any property that belongs to an object's prototype inheritance chain) can be enumerated using a `for...in` loop.

1.10 Three.js

Three.js is a JavaScript-based WebGL engine that enables browser-based GPU-powered gaming and other graphics-intensive applications. Many capabilities and APIs are available in the three.js library for generating 3D scenes in your browser.Without the use of specialized browser plugins, Three.js enables the development of 3D animations that are GPU-accelerated and integrated into websites using JavaScript. The development of WebGL, a low-level graphics API designed exclusively for the web, has made this possible. Complex 3D computer animations can be created for browser display without the work needed for a traditional standalone application or a plugin thanks to high-level tools like Three.js or GLGE, SceneJS, PhiloGL, and many others.



```
camera.position.z = 5
```

Fig 1.13

CHAPTER 2

LITERATURE REVIEW

2. Literature Review

- **Michael G. Kapteyn, Karen E. Willcox : Design of Digital Twin Sensing Strategies Via Predictive Modeling and Interpretable Machine Learning**
  The methodology for sensor placement and dynamic sensor scheduling decisions for digital twins is developed in this work. Predictive models are used to train the best classification trees that represent the map from observed data to estimated digital twin states. The assimilation of digital twin data is framed as a classification problem. The resulting classification trees produce an understandable mathematical representation that can be queried to help with sensor location and sensor scheduling decisions, in addition to giving a rapid digital twin updating capability. The suggested method is illustrated using a structural digital twin of an unmanned aerial vehicle with a 12 foot wingspan. By modelling situations with predictive reduced-order models of the vehicle in a variety of structural states, training data are produced offline.
- **Xiaonan Lai, Xiwang He, Shuo Wang : Building a Lightweight Digital Twin of a Crane Boom for Structural Safety Monitoring Based on a Multifidelity Surrogate Model**
  This study analyzed the Unrecognized wear and tear from overloading and tiredness at critical locations is one of the major dangers in construction that might cause structural failure in the crane boom. Consequently, the stability of the crane boom To make sure the crane boom is operating within the intended load capacity, it should be continuously monitored. To increase the real-time monitoring and prediction accuracy of the structural safety of a crane boom, we created a lightweight digital twin using the multifidelity surrogate (MFS) model in this work. The use of digital twin technology for online monitoring and analysis of buildings, machinery, and even human bodies has great promise because it allows for real-time mapping between the physical and digital worlds. Using sensor data and the MFS model, the lightweight digital twin may dynamically mirror the crane boom postures and predict its structural performance in real time.
- **Xiaonan Lai, Xiwang He, Yong Pang : A Scalable Digital Twin Framework Based on a Novel Adaptive Ensemble Surrogate Model**
  This work deals with ensembling offers reliable approximation in addition to reduces the number of ISMs participating in the ensemble by multicriterion model screening, which also lowers the additional cost brought on by the ensemble. Moreover, A node

rearrangement approach that offers scalability for the creation of a digital model is proposed and is based on the features of the finite element method. In other words, the distribution and number of nodes can be altered to acquire the information at strategic locations while simultaneously lowering the computational cost by reducing the number of nodes. The effectiveness of the suggested ensemble and node rearrangement strategy is tested using numerical experiments. To demonstrate the viability of creating scalable digital twins, a telehandler is utilised as an example.

- **Ricard Sanchıs, Salvador Cardona, Jordi Martınez : Determination of the Vertical Vibration of a Ballasted Railway Track to Be Used in the Experimental Detection of Wheel Flats in Metropolitan Railways**
  This paper presents a mathematical model used to obtain the vertical vibration of a ballasted railway track when a wheel is passing at a certain speed over a fixed location of the rail. The aim of this simulation is to compare calculated root-mean-square (RMS) values of the vertical vibration velocity with measured RMS values. This comparison is the basis for a proposed time domain methodology for detecting potential wheel flats or any other singular defect on the wheel rolling bands of metropolitan trains. In order to reach this goal, a wheel–rail contact model is proposed; this model is described by the track vertical impulse response and the vertical impulse response of the wheel with the primary suspension, both linked through a Hertz nonlinear stiffness.
- **Aidan Fuller, , Zhong Fan, Charles Day : Digital Twin: Enabling Technologies, Challenges and Open Research**
  The paper provides an assessment of the enabling technologies, challenges and open research for Digital Twins There are many definitions of the Digital Twin, but the best described as the seamless integration of data going either way between a physical and virtual machine. The problems uses, and technology that makes artificial intelligence and the Internet of Things (IoT) possible as well as Digital Twins are shown. A categorical review of recent studies is produced after a review of publications on digital twins. Manufacturing, healthcare, and smart cities have been classed by study topics, and the review discusses a variety of publications that reflect these areas and the current level of research in each of these fields.

CHAPTER 3

METHODOLOGY

3. Methodology

There are two ways to deploy digital twin technology in this scenario. They are:

1. Processing applications while utilising an Arduino.

2. Using three.js, Fusion 360 and webapp.

3.1 In Processing application while using an Arduino

In this method an Arduino UNO board, MPU 650 Sensor, processing application and Arduino Application are required.

Step by step procedure:

- Initially Arduino application is to be downloaded in the windows / mac based on the availability.
- MPU 650 Sensor has to be connected with Arduino board using the jumper wires as shown in the below figure.

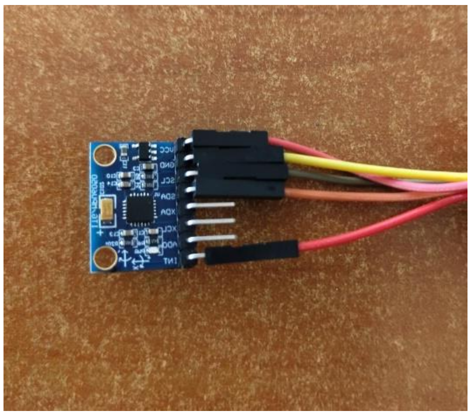


Fig 3.1 MPU 6050 Sensor

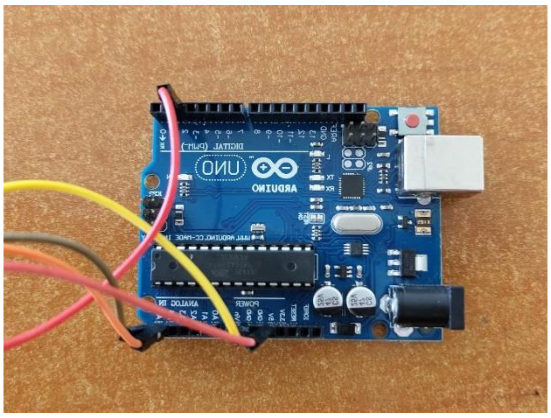


Fig 3.2 Arduino UNO Board

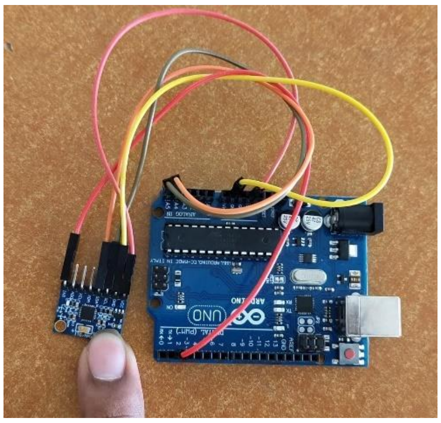


Fig 3.3

- Now the entire Sensor – Arduino fusion is to be connected to the computer using Micro USB Cable A-B as shown in the below figure.

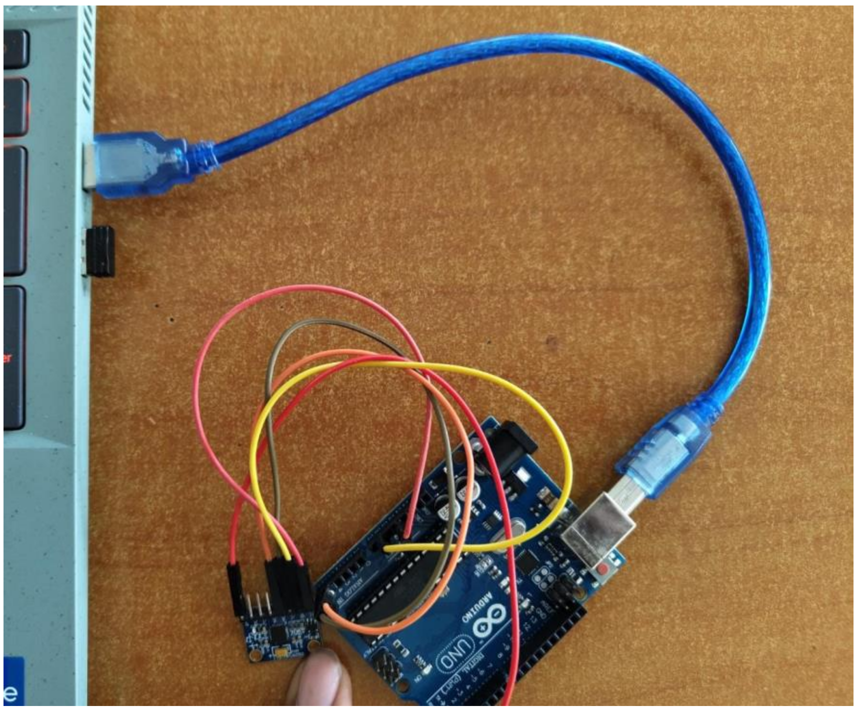


Fig 3.4(a) Connection to the system

- Make sure that the LED bulbs of sensor and Arduino board are blinking / Switched on as shown in Fig 3.4 so that it is ensured that all the components are in proper working condition.

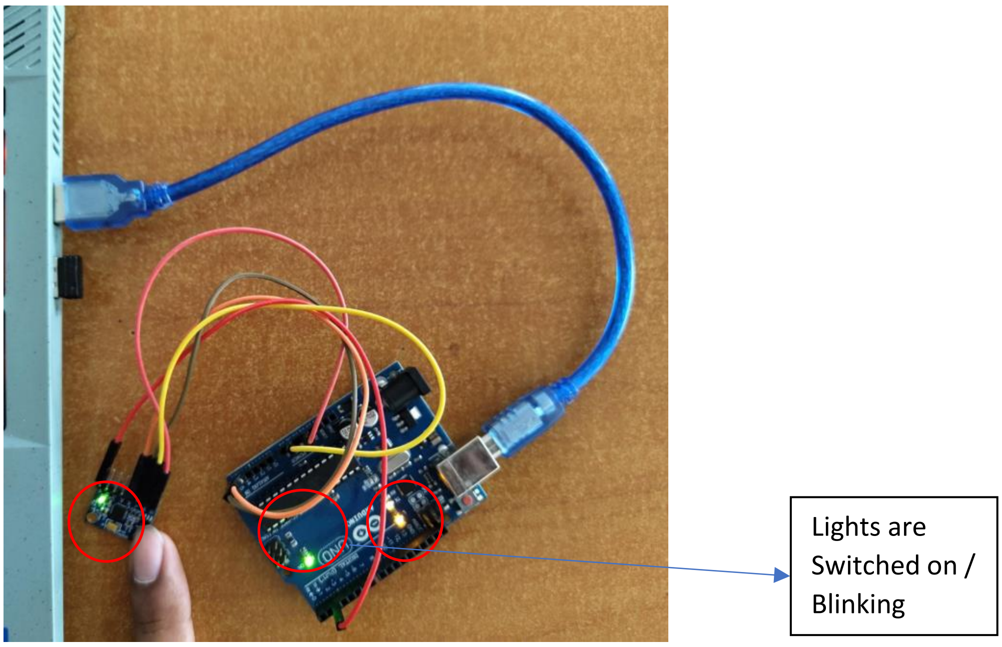
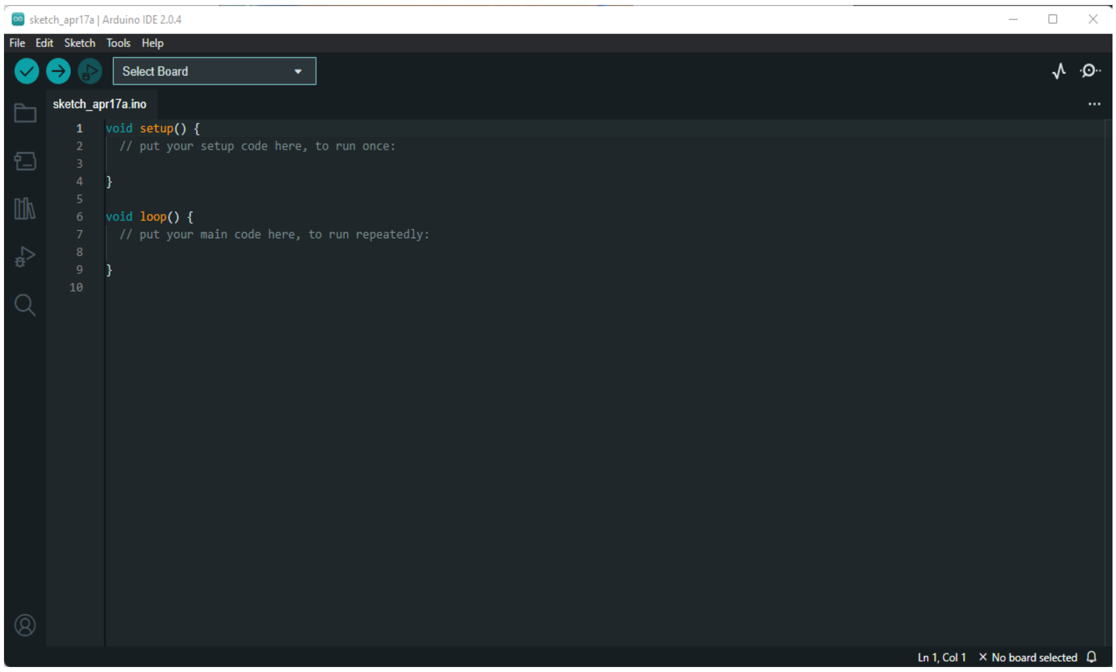


Fig 3.4(b)

- Immediately after the setup is done, Arduino Application is to be opened which will look as shown in Fig 3.5


sketch_apr17a | Arduino IDE 2.0.4
File Edit Sketch Tools Help
Select Board
sketch_apr17a.ino
Ln 1, Col 1
No board selected

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Fig 3.5 Arduino IDE

- Select the board by clicking on “Select Board” Dropdown. Choose the Required board based on the ports available and port that is mentioned in the code. The available ports are shown in the Fig 3.6

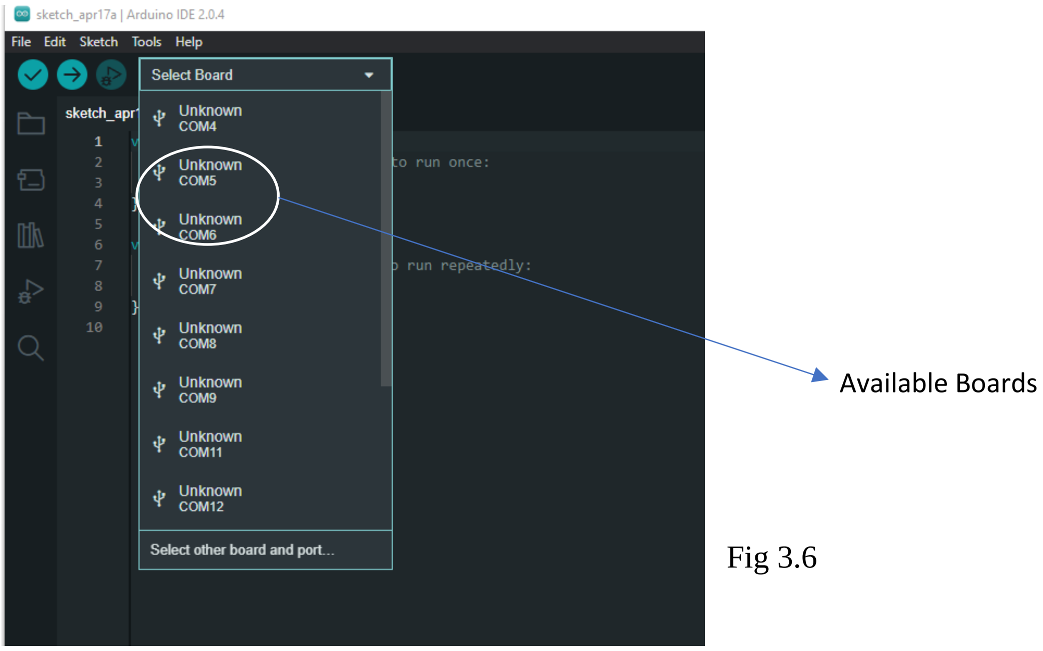


Fig 3.6

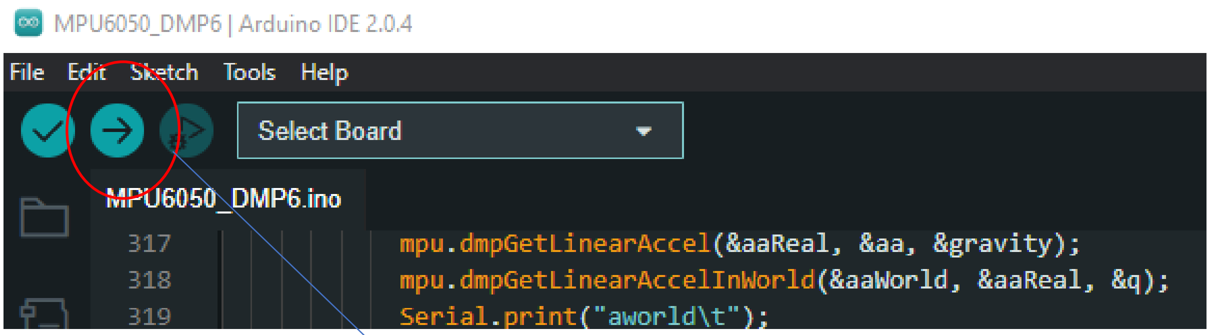
- Based on the Port selected and Sensor configurations the code need to be developed in the Arduino IDE.
- Arduino code developed to integrate MPU 6050 sensor and Arduino UNO Board is given below

```
===============================================
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 * ========================================================================= */

/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 * ========================================================================= */



// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
```

```
//#define OUTPUT_READABLE_QUATERNION

#define OUTPUT_READABLE_YAWPITCHROLL


// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT



#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

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

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };



// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}



// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation
difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately


    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection
failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

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

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

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}



// ================================================================
// ===                      MAIN PROGRAM LOOP                   ===
// ================================================================

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;
    // read a packet from FIFO
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
        #endif

        #ifdef OUTPUT_READABLE_EULER
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);
            Serial.print("euler\t");
            Serial.print(euler[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(euler[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(euler[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            Serial.print("areal\t");
            Serial.print(aaReal.x);
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
```

```
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

- In order to integrate this code into Arduino UNO Board, this code has to be compiled and executed using the arrow symbol shown in the Fig 3.7



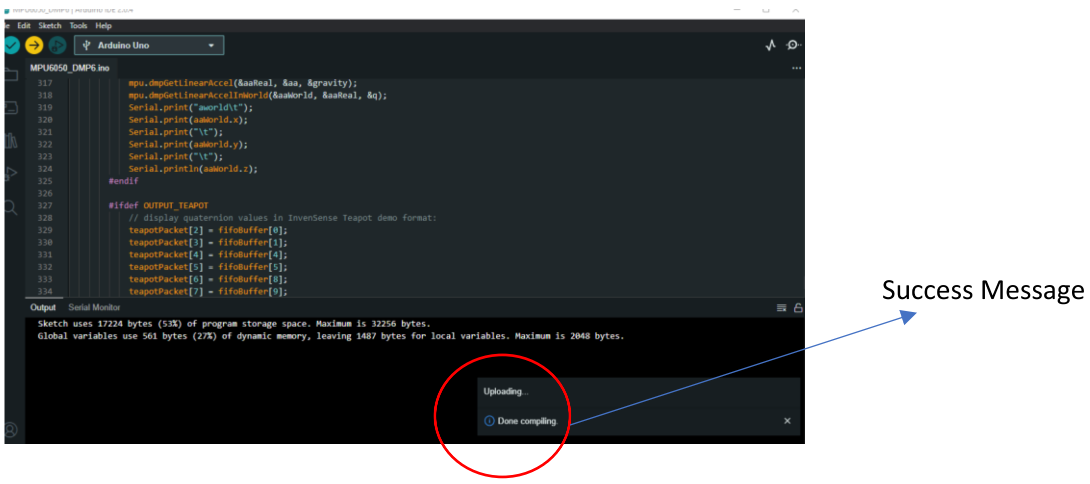
```
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
```

Fig 3.7

Compile button

- Wait until the compilation is done and receive A success message in the terminal as shown in Fig 3.8

Fig 3.8



```
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
```

```
Sketch uses 17224 bytes (53%) of program storage space. Maximum is 32256 bytes.
Global variables use 561 bytes (27%) of dynamic memory, leaving 1487 bytes for local variables. Maximum is 2048 bytes.
```

Processing Application

- Download Java’s Processing application from chrome and install it in the computer.
- Open the processing app and need to develop a code to integrate Arduino with processing application.
- The processing application's user interface will resemble that in Fig. 3.9

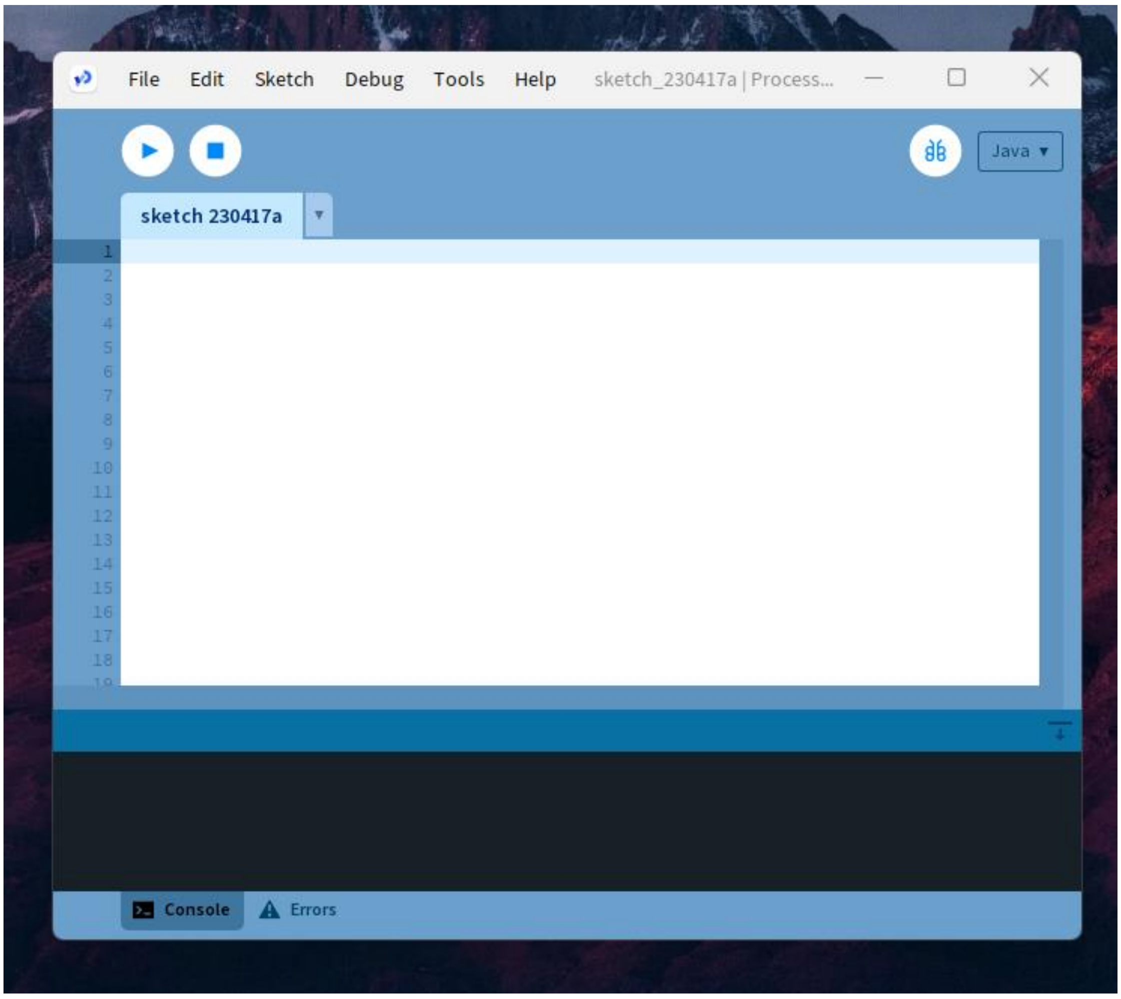


Fig 3.9

- Now a new extension called ‘TeaPot’ is to be downloaded which is a high-level JavaScript library for making SVG plots.
- Path to Open the TeaPot file from Libraries is :

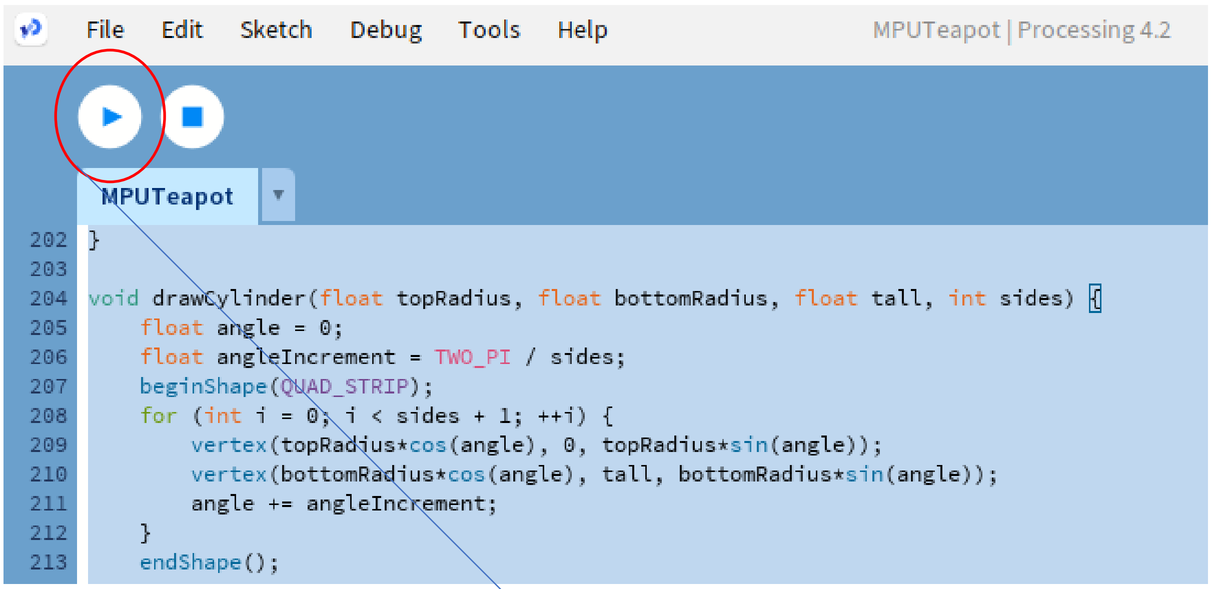
C:\Users\CHANDRA MOULI\Documents\Processing\MPUTeapot

- The code developed is given below.

```
import processing.serial.*;
import processing.opengl.*;
import toxi.geom.*;
import toxi.processing.*;

// NOTE: requires ToxicLibs to be installed in order to run properly.
// 1. Download from http://toxiclibs.org/downloads
// 2. Extract into [userdir]/Processing/libraries
//    (location may be different on Mac/Linux)
// 3. Run and bask in awesomeness

ToxiclibsSupport gfx;
```

```
Serial port;                         // The serial port
char[] teapotPacket = new char[14];  // InvenSense Teapot packet
int serialCount = 0;                 // current packet byte position
int synced = 0;
int interval = 0;

float[] q = new float[4];
Quaternion quat = new Quaternion(1, 0, 0, 0);

float[] gravity = new float[3];
float[] euler = new float[3];
float[] ypr = new float[3];

void setup() {
    // 300px square viewport using OpenGL rendering
    size(300, 300, OPENGL);
    gfx = new ToxiclibsSupport(this);

    // setup lights and antialiasing
    lights();
    smooth();

    // display serial port list for debugging/clarity
    println(Serial.list());

    // get the first available port (use EITHER this OR the specific port code below)
    //String portName = Serial.list()[0];

    // get a specific serial port (use EITHER this OR the first-available code above)
    String portName = "COM10";

    // open the serial port
    port = new Serial(this, portName, 115200);

    // send single character to trigger DMP init/start
    // (expected by MPU6050_DMP6 example Arduino sketch)
    port.write('r');
}

void draw() {
    if (millis() - interval > 1000) {
        // resend single character to trigger DMP init/start
        // in case the MPU is halted/reset while applet is running
        port.write('r');
        interval = millis();
    }

    // black background
    background(0);

    // translate everything to the middle of the viewport
    pushMatrix();
    translate(width / 2, height / 2);

    // 3-step rotation from yaw/pitch/roll angles (gimbal lock!)
    // ...and other weirdness I haven't figured out yet
    //rotateY(-ypr[0]);
    //rotateZ(-ypr[1]);
    //rotateX(-ypr[2]);

    // toxiclibs direct angle/axis rotation from quaternion (NO gimbal lock!)
    // (axis order [1, 3, 2] and inversion [-1, +1, +1] is a consequence of
    // different coordinate system orientation assumptions between Processing
    // and InvenSense DMP)
    float[] axis = quat.toAxisAngle();
    rotate(axis[0], -axis[1], axis[3], axis[2]);

    // draw main body in red
    fill(255, 0, 0, 200);
    box(10, 10, 200);
    // draw front-facing tip in blue
    fill(0, 0, 255, 200);
    pushMatrix();
    translate(0, 0, -120);
    rotateX(PI/2);
    drawCylinder(0, 20, 20, 8);
    popMatrix();

    // draw wings and tail fin in green
    fill(0, 255, 0, 200);
    beginShape(TRIANGLES);
    vertex(-100,  2, 30); vertex(0,  2, -80); vertex(100,  2, 30);  // wing top layer
    vertex(-100, -2, 30); vertex(0, -2, -80); vertex(100, -2, 30);  // wing bottom layer
    vertex(-2, 0, 98); vertex(-2, -30, 98); vertex(-2, 0, 70);  // tail left layer
    vertex( 2, 0, 98); vertex( 2, -30, 98); vertex( 2, 0, 70);  // tail right layer
    endShape();
    beginShape(QUADS);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -80);
    vertex( 100, 2, 30); vertex( 100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -80);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(100, -2,  30); vertex(100, 2,  30);
    vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2, -30, 98); vertex(-2, -30, 98);
    vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
    vertex(-2, -30, 98); vertex(2, -30, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
    endShape();

    popMatrix();
}

void serialEvent(Serial port) {
    interval = millis();
    while (port.available() > 0) {
        int ch = port.read();

        if (synced == 0 && ch != '$') return;   // initial synchronization - also used to resync/realign if needed
        synced = 1;
        print ((char)ch);

        if ((serialCount == 1 && ch != 2)
            || (serialCount == 12 && ch != '\r')
            || (serialCount == 13 && ch != '\n'))  {
            serialCount = 0;
            synced = 0;
            return;
        }

        if (serialCount > 0 || ch == '$') {
            teapotPacket[serialCount++] = (char)ch;
            if (serialCount == 14) {
                serialCount = 0; // restart packet byte position

                // get quaternion from data packet
                q[0] = ((teapotPacket[2] << 8) | teapotPacket[3]) / 16384.0f;
                q[1] = ((teapotPacket[4] << 8) | teapotPacket[5]) / 16384.0f;
                q[2] = ((teapotPacket[6] << 8) | teapotPacket[7]) / 16384.0f;
                q[3] = ((teapotPacket[8] << 8) | teapotPacket[9]) / 16384.0f;
                for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4 + q[i];

                // set our toxilibs quaternion to new data
                quat.set(q[0], q[1], q[2], q[3]);

                /*
                // below calculations unnecessary for orientation only using toxilibs

                // calculate gravity vector
                gravity[0] = 2 * (q[1]*q[3] - q[0]*q[2]);
                gravity[1] = 2 * (q[0]*q[1] + q[2]*q[3]);
                gravity[2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] + q[3]*q[3];

                // calculate Euler angles
                euler[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
                euler[1] = -asin(2*q[1]*q[3] + 2*q[0]*q[2]);
                euler[2] = atan2(2*q[2]*q[3] - 2*q[0]*q[1], 2*q[0]*q[0] + 2*q[3]*q[3] - 1);

                // calculate yaw/pitch/roll angles
                ypr[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
                ypr[1] = atan(gravity[0] / sqrt(gravity[1]*gravity[1] + gravity[2]*gravity[2]));
                ypr[2] = atan(gravity[1] / sqrt(gravity[0]*gravity[0] + gravity[2]*gravity[2]));

                // output various components for debugging
                //println("q:\t" + round(q[0]*100.0f)/100.0f + "\t" + round(q[1]*100.0f)/100.0f + "\t" + round(q[2]*100.0f)/100.0f +
"\t" + round(q[3]*100.0f)/100.0f);
                //println("euler:\t" + euler[0]*180.0f/PI + "\t" + euler[1]*180.0f/PI + "\t" + euler[2]*180.0f/PI);
                //println("ypr:\t" + ypr[0]*180.0f/PI + "\t" + ypr[1]*180.0f/PI + "\t" + ypr[2]*180.0f/PI);
                */
            }
        }
    }
}

void drawCylinder(float topRadius, float bottomRadius, float tall, int sides) {
    float angle = 0;
    float angleIncrement = TWO_PI / sides;
    beginShape(QUAD_STRIP);
    for (int i = 0; i < sides + 1; ++i) {
        vertex(topRadius*cos(angle), 0, topRadius*sin(angle));
        vertex(bottomRadius*cos(angle), tall, bottomRadius*sin(angle));
        angle += angleIncrement;
    }
    endShape();

    // If it is not a cone, draw the circular top cap
    if (topRadius != 0) {
        angle = 0;
        beginShape(TRIANGLE_FAN);

        // Center point
        vertex(0, 0, 0);
        for (int i = 0; i < sides + 1; i++) {
            vertex(topRadius * cos(angle), 0, topRadius * sin(angle));
            angle += angleIncrement;
        }
        endShape();
    }

    // If it is not a cone, draw the circular bottom cap
    if (bottomRadius != 0) {
        angle = 0;
        beginShape(TRIANGLE_FAN);

        // Center point
        vertex(0, tall, 0);
        for (int i = 0; i < sides + 1; i++) {
            vertex(bottomRadius * cos(angle), tall, bottomRadius * sin(angle));
            angle += angleIncrement;
        }
        endShape();
    }
}
```

- Hence all the required codes are developed and execution part is left.
- Execute the file using execute button in processing application.



```
}

void drawCylinder(float topRadius, float bottomRadius, float tall, int sides) {
    float angle = 0;
    float angleIncrement = TWO_PI / sides;
    beginShape(QUAD_STRIP);
    for (int i = 0; i < sides + 1; ++i) {
        vertex(topRadius*cos(angle), 0, topRadius*sin(angle));
        vertex(bottomRadius*cos(angle), tall, bottomRadius*sin(angle));
        angle += angleIncrement;
    }
    endShape();
```

Fig 3.10

Execute Button

- When execute button is clicked a blank screen with the 3d model will be opened which is running live as shown in the Fig 3.11

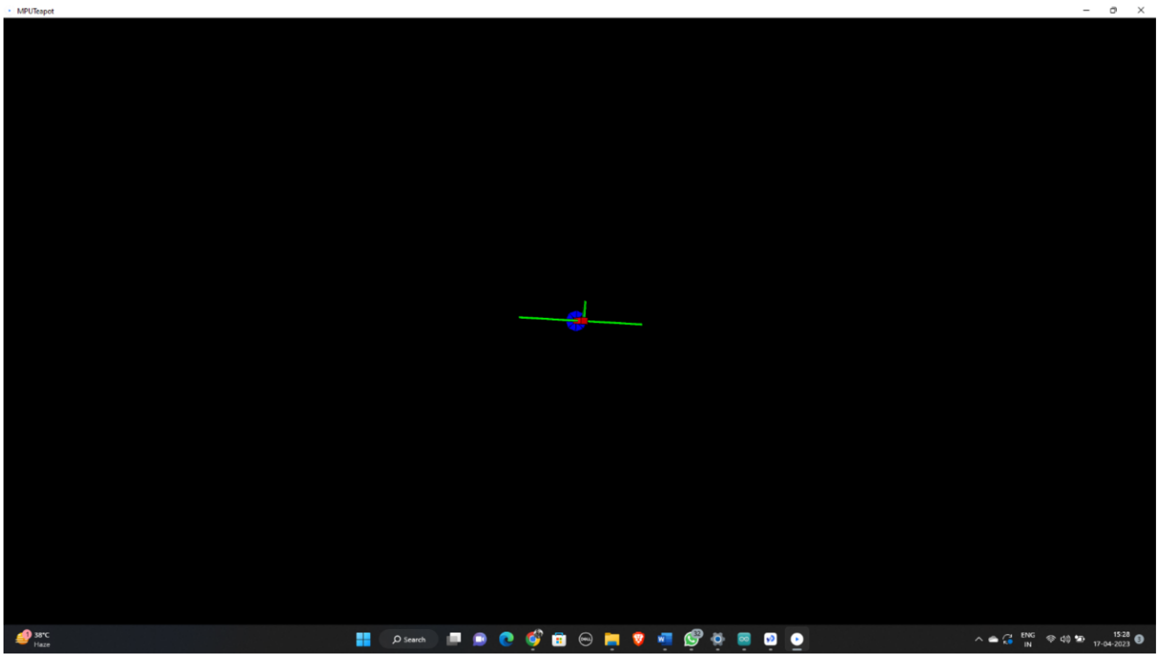


Fig 3.11

- Hence the 3d model is integrated with the sensor equipped to the physical object and will be resembled as a twin to the physical object.

3.2 Development of Digital Twin of a screw jack using three.js, Fusion 360 and webapp.

CHAPTER 4

RESULTS & DISCUSSIONS

4.1 Results of Digital twin model using Arduino and Processing application.

This is the final 3d model of a physical object in virtual/ Digital format. Which exactly possess the structural and functional features of physical object.

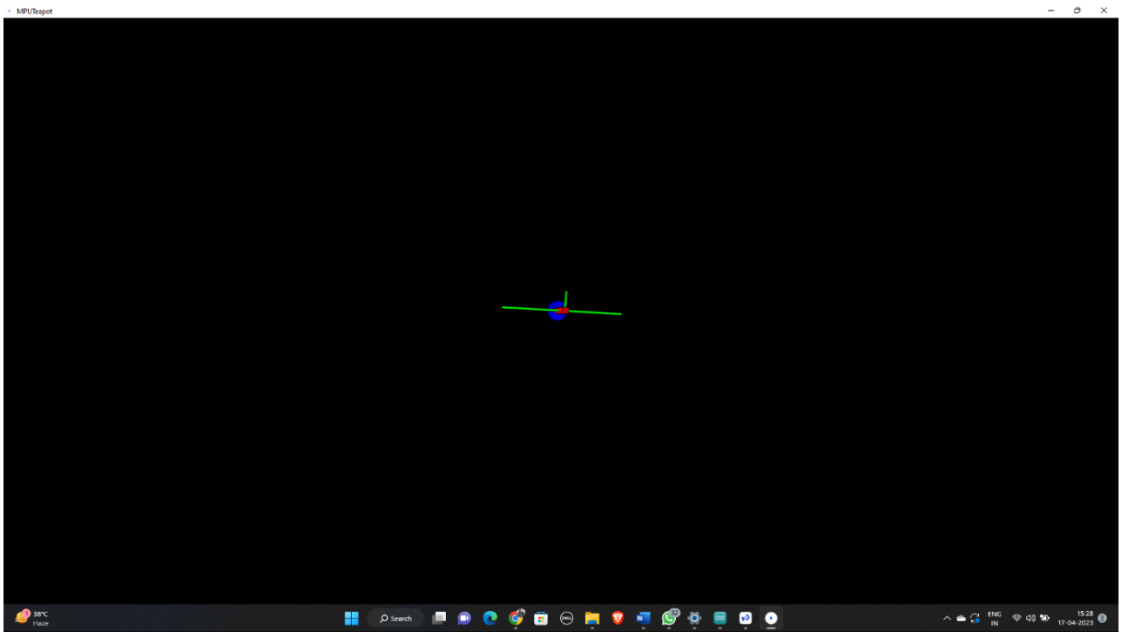


Fig 4.1 : Digital twin of physical object in virtual space.

Following figures Fig 4.2, Fig 4.3, Fig 4.4 and Fig 4.5 Depict the variation or transformation of 3d model based on the movement of physical object.

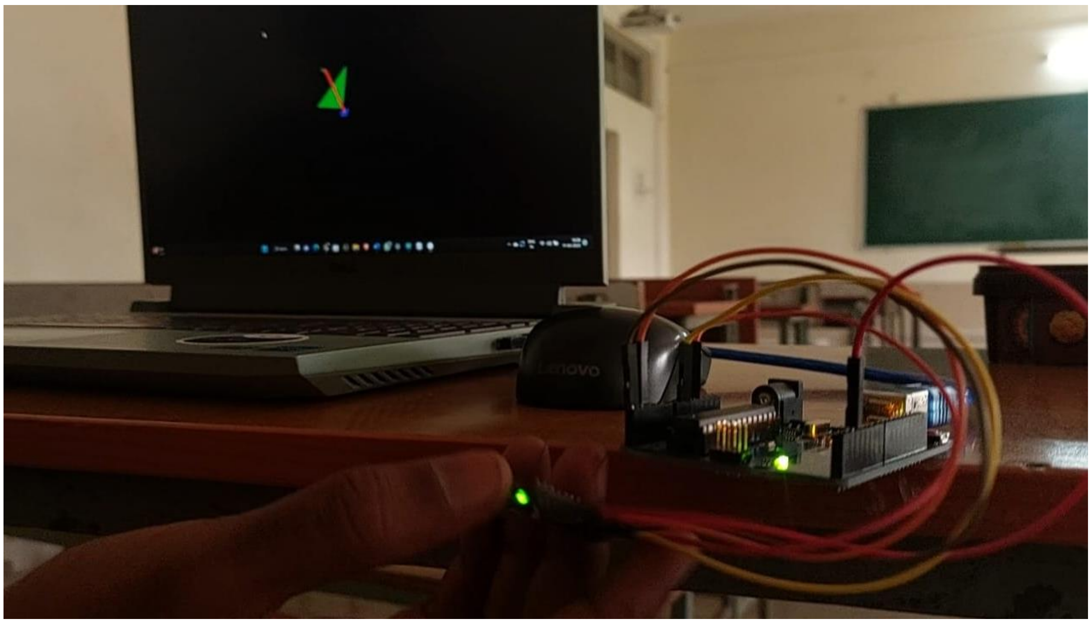


Fig 4.2

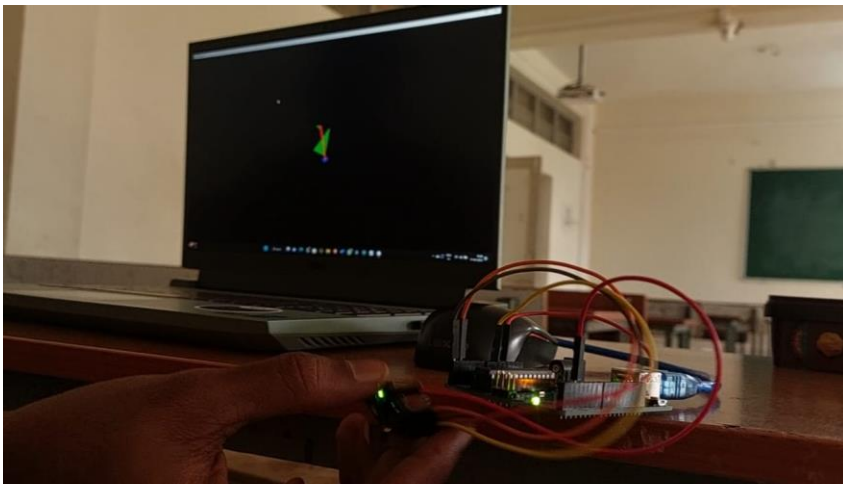


Fig 4.3

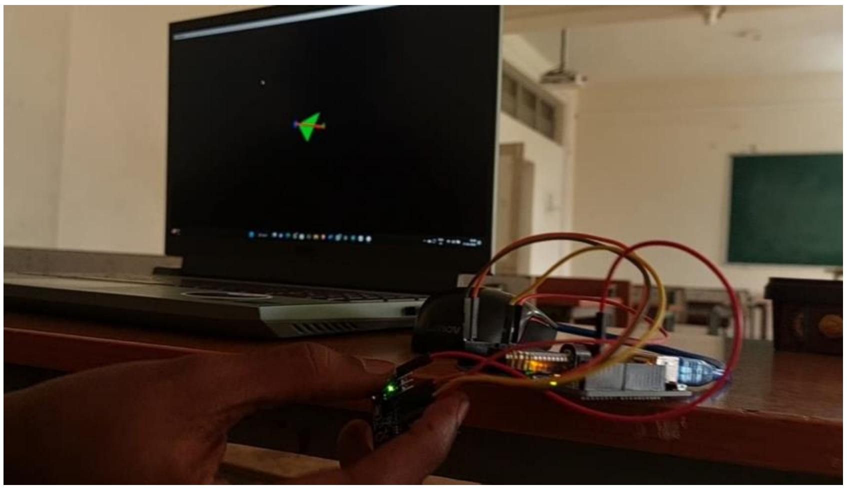


Fig 4.4

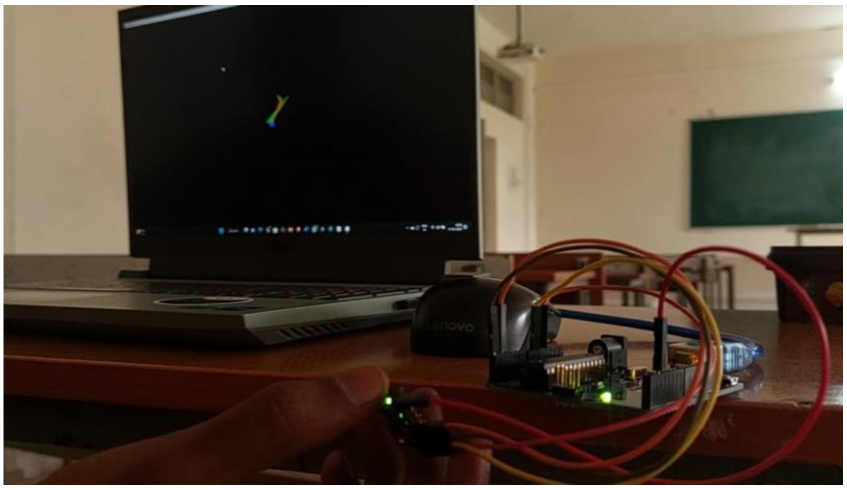


Fig 4.5

4.2 Results of Digital Twin of a screw jack developed in fusion 360 and implemented in webapp using three.js

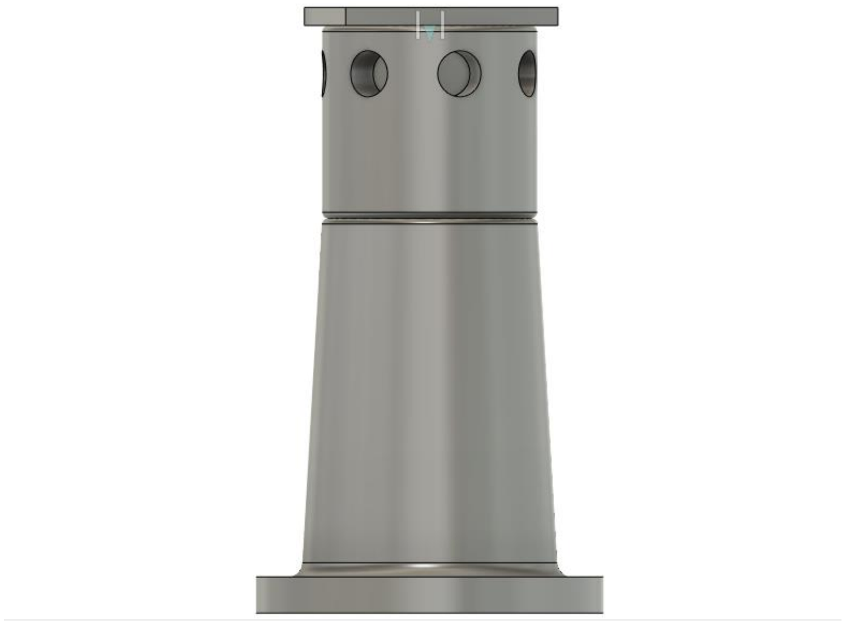


Fig 4.6 Front view of Screw jack developed in fusion 360

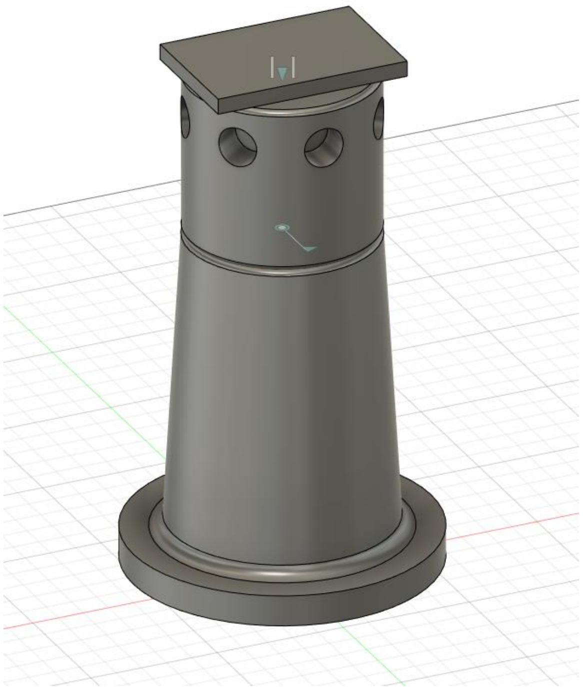


Fig 4.7 Front view of Screw jack developed in fusion 360

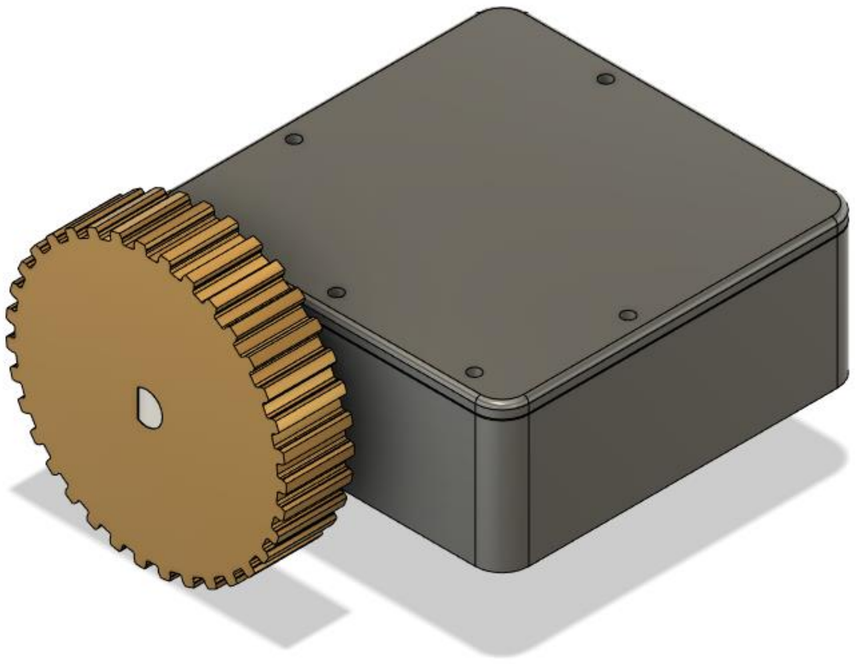


Fig 4.8 Sensor Box

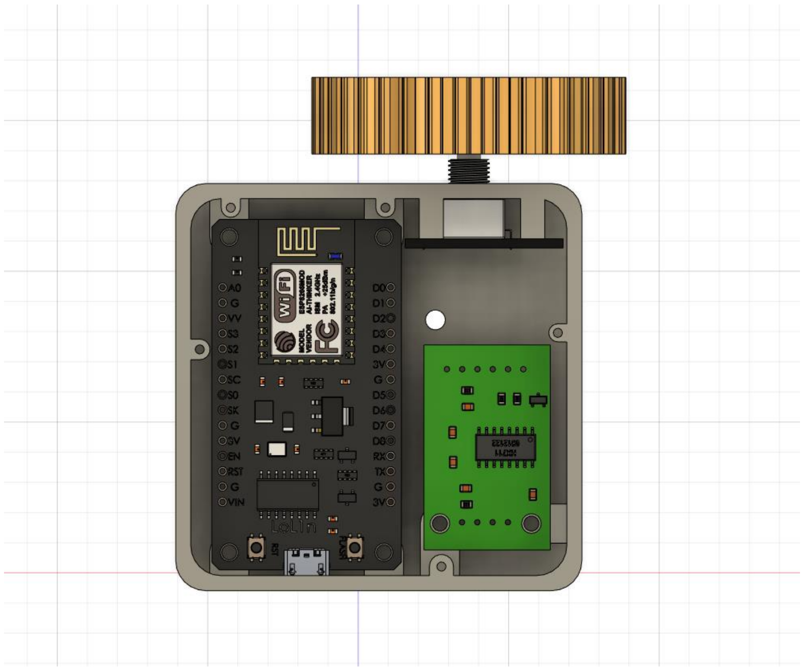


Fig 4.9 cross sectional view of sensor box consisting of M274 encoder, 5kg load cell hx711 module

CHAPTER 5

CONCLUSION

5. CONCLUSION

Digital twin can be created in numerous ways but the accuracy of the twin created will vary from method to method. Through the 1st method using ‘Processing application’ the model obtained was not so accurate. It is not recommended to create digital twin for complex structures like mechanical components through this method of processing application.

Whereas it is highly recommended to use web application method using three.js to create a digital twin of any object. Fusion 360 is a very sophisticated and highly reliable 3d modelling software which is used to create 3d models of all types ranging from minute nut and bolt to complex mechanical component like screw jack. Digital twin technology has proven to be a valuable tool in solving mechanical problems related to real-time analysis and detecting failures of mechanical components.

By creating a virtual replica of a physical system or component, engineers can analyze its behaviour and predict its performance under different conditions. Real-time analysis is essential in mechanical systems to ensure optimal performance and prevent failures. Digital twin technology enables engineers to monitor and analyze the behaviour of mechanical systems in real-time, allowing for early detection of potential problems. This early detection can lead to faster and more efficient maintenance, reducing downtime and improving overall system reliability. In addition, digital twin technology can be used to simulate different scenarios and test the performance of mechanical systems under various conditions. This allows engineers to optimize the design and operation of mechanical systems, leading to better performance, increased efficiency, and reduced costs. Overall, digital twin technology has significant potential in solving mechanical problems related to real-time analysis and detecting failures of mechanical components. As the technology continues to evolve and improve, it is likely to become an even more valuable tool for engineers working in the mechanical engineering field.

6. REFERENCES

- https://amses-journal.springeropen.com/articles/10.1186/s40323-020-00147-4
- https://digitaltwin1.org/articles/1-12
- https://www.researchgate.net/publication/337019778_Digital_Twin_Enabling_Technologies_Challenges_and_Open_Research
- https://typeset.io/papers/digital-twin-enabling-technologies-challenges-and-open-2aylj47ira
- https://www.mdpi.com/journal/applsci/special_issues/Digital_Twins_Industry
- https://www.electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module