

A Project Report on

IOT BASED SMART CHECK WEIGHER DATA SYSTEM



Department Of Electronics Engineering
Finolex Academy of Management and Technology, Ratnagiri – 415639
2021-22

A Project Report on

IOT BASED SMART CHECK WEIGHER DATA SYSTEM

Prepared by

Bhave Omkar Vinay – N-18-0303

Teredesai Rajesh Sudhakar – ND-19-0349

Sawant Parth Rajesh – N-18-0273

Guide

Prof. V. V. Nimbalkar

Co – Guide

Prof. S. R. Nalage



Department Of Electronics Engineering
Finolex Academy of Management and Technology, Ratnagiri – 415639
2021-22

CERTIFICATE

This is to verify the following student have prepared and presented a report on **“IOT BASED SMART CHECK WEIGHER DATA SYSTEM”** as per partial fulfilment of the term work requirements for the project prescribed by University of Mumbai for Bachelor of Engineering, Year 2021-22.

Prepared by

Bhave Omkar Vinay – N-18-0303

Teredesai Rajesh Sudhakar – ND-19-0349

Sawant Parth Rajesh – N-18-0273

Prof. V. V. Nimbalkar
(Project Guide)

Prof. S. R. Nalage
(Project Co-Guide)

External Examiner

Internal Examiner

Prof. V. V. Nimbalkar
(Hod, Electronics)

Dr. Kaushal Prasad
Principal

Date:

Place: FAMT, Ratnagiri.

ACKNOWLEDGEMENT

We acknowledge with gratitude those who have imparted their valuable time, energy and intellectual towards the beautification of our project titled, **“IOT BASED SMART CHECK WEIGHER DATA SYSTEM”**.

We are grateful to thank our co-guide, **Prof. S. R. Nalage** and our guide, **Prof. V. V. Nimbalkar** at the Electronics Engineering Department at FAMT, Ratnagiri, and **Mr. S. Ravindran**, HOD Instrumentation at Finolex Industries Ltd. Ratnagiri for their kind interest, support, advice, constant encouragement and guidance throughout the project work.

We are thankful to **Finolex Industries** for the sponsorship and technical assistance.

Furthermore, we would like to thank all those who have associated and helped directly and indirectly in our project.

DECLARATION

We declare that this written submission represents our ideas in our own words. In addition, we affirm that we adhered to all the standards of academic honesty, integrity, and regulations of **Finolex Industries** and have not falsified any information, data, or fact in our submission. We understand that any violation of the above will cause disciplinary action by the Institute and can also evoke penal action from the sources which have just not been referred or from whom proper permission has not been taken when needed.

Bhave Omkar Vinay

Teredesai Rajesh Sudhakar

Sawant Parth Rajesh

Date:

Abstract

Currently, a company loads random sample bags and weighs them as a palate out of 60 bags. Thus, the dispatch palate is not of accurate weight. We are developing and implementing an IoT-based solution to this problem.

The software and hardware required to implement the solution is explain furthermore, the operation is described.

Contents

1	Introduction	2
1.1	Background of the project	2
1.2	Problem Statement and our Solution	3
1.3	Project Objective	4
1.4	Significance of the Project	4
2	Project Idea	6
2.1	Block Diagram	6
2.2	Description of Block Diagram	6
2.3	Theory of Weight Scales	6
2.4	Circuit Diagram	7
3	Hardware Components	9
3.1	ESP8266	9
3.2	DAT-400	11
3.2.1	Introduction	11
3.2.2	Frame Format	12
3.3	Protocols Used	13
3.3.1	RS232	13
3.3.2	RS485	13
3.4	Front Panel of DAT-400	14
3.4.1	Display	14
3.4.2	Led indicators	14
3.4.3	Keyboard Functions	14
4	Software Requirements	17
4.1	Arduino IDE	17
4.2	Google App Scripts	18
4.2.1	General information	18
4.2.2	Benefits	19
4.2.3	Limitations	19
4.3	MIT App inventor	19
5	Codes	22
5.1	Arduino Code	22
5.1.1	Initialization	22
5.1.2	Setup Code	23
5.1.3	Loop Code	24
5.1.4	Google sheets initialization	24
5.1.5	Google Sheets Data-Upload	25
5.2	Google Appscript Code	26
5.2.1	Google Appscript initialization	26
5.2.2	Google Appscript Read Code	27
5.3	MIT app inventor code	27
5.3.1	App Inventor Initialization	27
5.3.2	App Inventor Formating	28

5.3.3	App Inventor Read Function	28
6	Results	30
6.1	Arduino Code Result	30
6.2	Google app script code result	31
6.3	MIT app inventor result	32
7	Conclusion	34
8	Future Work	36

List of Figures

2.1	Block Diagram	6
2.2	Circuit Diagram	7
3.1	ESP8266	9
3.2	Actual image of the Die of ESP chip	10
3.3	Different versions of ESP	10
3.4	Hardware in the Industry	11
3.5	Connection of DAT-400 to 4 load-cells	11
3.6	Frame Format	12
3.7	Maximum Frequency to baud rate	12
3.8	RS232 used by DAT-400	13
3.9	RS485 used by DAT-400	13
3.10	Front Panel of DAT-400	14
3.11	Function during the weight display	14
4.1	Arduino IDE	17
4.2	Google App Scripts Image	18
4.3	MIT App inventor Screen	20
5.1	Initialization of Arduino Code	22
5.2	Setup of Arduino Code	23
5.3	Setup (contd) of Arduino Code	23
5.4	Loop routine of Arduino Code	24
5.5	Send data sub-routine of Arduino Code	24
5.6	Send data sub-routine upload part of Arduino Code	25
5.7	Gsheet Code	26
5.8	Gsheet code for the android app	27
5.9	Initialization of android app	27
5.10	Reception of data on android app	28
5.11	Reading data on android app	28
6.1	Result of Arduino Code	30
6.2	Result of Arduino Code on Google sheets	31
6.3	Result of the MIT app inventor code	32

Chapter 1
INTRODUCTION

Chapter 1

Introduction

1.1 Background of the project

In the area of mass production, products are weighed using load cell based dynamic weighing systems. A load cell is an uncontrollable weighing device and the value of weight, for the passing product, is estimated by filtering the electrical signal from a load cell. Improvement in filtering increases the speed of weighing and enhances the measurement accuracy. Since this approach is based on the accurate model of the system in question, the exact model of the load cell based dynamic weighing system has been derived and presented. For one particular value of the weight, the parameters of the model are time-varying due to the product coming onto the weigh-table and due to the product length. Changing the measurement from one value of the weight to another causes even greater changes in the values of the model parameters and introduces a nonlinearity in the system. Therefore an adaptivity approach has been considered and a solution proposed. The simulation and experimental results are presented and compared.

Currently, most of the commodities packed with specific weights based on the customer needs. These jobs handled by the logistic groups by paying lesser amount to the farmers. Weighing scales that are used analog or manual scales. These scales are robust, which are made of iron but there is high possibility of manipulations can be done at any point of time while weighing the commodities. Even the commodities must be of the best quality with respect to different aspects like moisture percentage, size of the seeds and colour of the seeds and so on. As discussed, traditional weighing scales used in the older days from past few decades are volumetric measurement, which later moved on to the balance weighing scale, and analog weighing scale called the Kata machine (for weighing bags, big bowls and so on). Currently, digital weighing machines entered the market. Following are four different types of weighing machines used for different purposes. Spring Scale, also called as Newton Meter, used to weigh boxes and bags, can weigh loads between 0 to 50 kg. Floor Scales used to weigh the trucks with load and heavy animals like elephants, loaded trucks and so on. These scales generally weigh loads between 500 kg to 10000 kg.

Platform Scales used in small-scale industries where to measure the raw materials and products on daily basis. These materials weigh loads between 1000 kg and 5000 kg. Bench Scales used to weigh the high cost metals such as gold, platinum and so on. These weighing machines weigh in grams and milligrams and are 9574 Special Issue-international Journal of Pure and Applied Mathematics very accurate in the calculation. Sometimes these machines used for weighing new-born babies in the hospitals. For setting up an Intelligent Filing and Weighing, system will need huge investment/maintenance cost for the small and medium farmers. Automatic Weighing Solution provided for all varieties like Nuts, Grains and Powders, which weighed and bagged. However, it needs space for the entire infrastructure set up and capital investment will be huge. Farmers or specific business people do not require this automatic weighing machine solution for season based crops/business.

As people are getting smarter, so are the things. The Internet of Things (IoT) is a system of connecting mechanical and digital machines, animals or people with interrelated computing devices to provide an ability to collect, process and transfer data over a network without requiring human interaction. While one thinks of smart cities, we should also think of advanced industries. A checkweigher is an automatic or manual machine

for checking the weight of packaged commodities. It is normally found at the offgoing end of a production process and is used to ensure that the weight of a pack of the commodity is within specified limits. Any packs that are outside the tolerance are taken out of line automatically.

1.2 Problem Statement and our Solution

Currently, a company loads two sample bags and weighs them as a palate out of 60 bags per bag weighing 25kg. Thus, the dispatch palate is not of accurate weight, which is why we are developing and implementing an Io T- based solution to this problem. Every bag will be measured using a load cell, and its net weight will be serially transmitted via RS232, and its net weight is then extracted from the frame format. A comparison follows between the extracted weight and the ideal weight. The bag is forwarded for packing if the weight matches; simultaneously, the weight is displayed and recorded in an excel spreadsheet. The bag will not be processed if its weight is wrong. Stacks of sixty accurately weighed bags are packed by repeating this process. Such a stack is referred to as a palette. Initially, we will be implementing the idea using Python, and at a later stage, we will use ESP8266 hardware.

1.3 Project Objective

The main objective of our project is to fulfil the requirement of the industry. The weigh-scale system available in the industry is manual and needs the workers to check the accuracy of the weight and our project goal is to automate this task of checking weight and continuously monitor the weight through an application.

1.4 Significance of the Project

The main advantage of this project is to achieve the automation required to reduce the manpower that is being used just to monitor the weight scales and the time it takes to report that data to the supervisor and so on, by making sure that the right information is sent to the right person of responsibility and at the right time. We have used the current system in the industry and did some of our own modifications to it making it more accurate.

Chapter 2
PROJECT IDEA

Chapter 2

Project Idea

2.1 Block Diagram



Figure 2.1: Block Diagram

2.2 Description of Block Diagram

The block diagram represents the hardware components with the software part included in it. We have used ESP8266 as the processing unit which will fetch the data and will upload that data to the google sheets using its WiFi capabilities and will also be able to run the code of precisely fetching the net weight from the actual frame format (which will be in the later chapters). We have used the LCD display for only the hardware representation of the project.

2.3 Theory of Weight Scales

Balances, which weigh an object by matching it against one or more reference weights, have a delicate touch and are still used in laboratories. Scales use somewhat different physical principles and mechanical components to measure weight and other forces (weight is simply the force on an object due to gravity). Spring scales, for example, measure weight using Hooke's law, which relates force (weight) to the stretching or compression of a spring made from a given material. Just as a roadside carjack might lift a car via mechanical advantage – the leverage of a handle or the inclined plane of a screw – while a mechanic's hoist might use hydraulic pressure, different scales weigh objects using a variety of operational principles, like hydraulics, pneumatics or bending beams. Whatever makes them stretch, compress or sway, however, most modern scales share one component

in common: a load cell. Scales come in all shapes, sizes and configurations, but the basic component doing the measuring is nearly always a load cell.

A load cell is a kind of transducer, a device that converts one form of energy into another. Through load cells, digital scales change mechanical energy – the smooshing or stretching caused by a sitting or hanging load – into an electrical effect. The widely used strain gauge (you'll also see it as strain gage), for example, reads compression or tension as tiny changes in electrical resistance in a Wheatstone bridge. Let's break that down using a compression strain gauge as an example. Compression occurs when an applied force reduces an object's volume, but it can also refer to a more general decrease in size along one or more dimensions. As it happens, squishing an electrically conductive material changes its electrical resistance, because longer and narrower wires are more resistant than shorter, wider ones. Think of it like water pushing through a pipe: The longer and narrower the pipe, the harder it is to force water through it. Various materials experience different resistance changes under deformation, a quality known as gauge factor. Gauge factor can also alter in response to temperature. Consequently, constantan alloy (55 percent copper and 45 percent nickel), which performs well at room temperature, has established itself as the go-to material for strain measurements.

To pick up the change in resistance caused by weight compression, one or more strain gauges are placed within a Wheatstone bridge. A Wheatstone bridge is an electrical circuit that can detect an unknown electrical resistance by balancing it against known resistances elsewhere in the circuit. In a sense, it's like a balance scale for electrical resistance: The "weight" (resistance) on one side tells you the unknown "weight" (resistance) on the other. A given bridge can contain 1-4 strain gauges. When multiple gauges are used, they're arranged in opposing directions to improve sensitivity and to mitigate temperature effects. Because the resistance change in a strain circuit can be minuscule, the signal often requires amplification. This part of processing and digital read out are handled by the use of Microprocessor Control Unit (MCU). Based on design necessity, it performs other necessary operations. It is the maintenance of zero adjustment, statistical data, and interface to a PC system etc.

2.4 Circuit Diagram

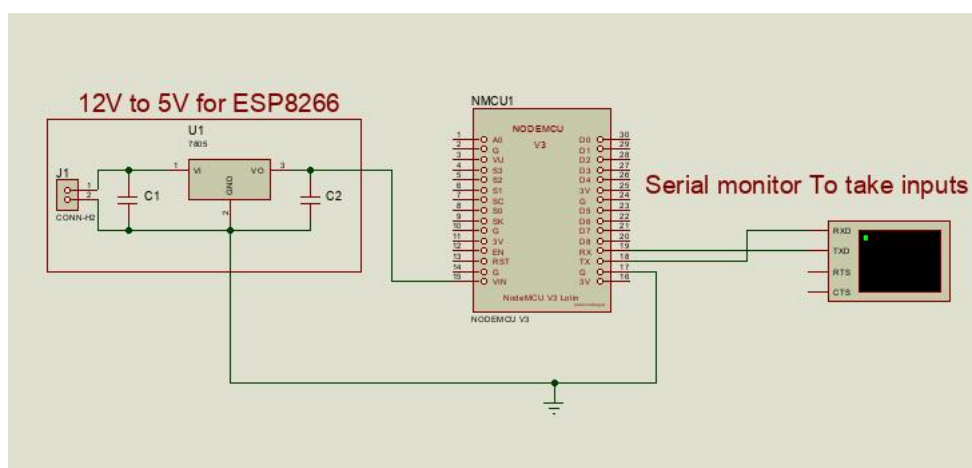


Figure 2.2: Circuit Diagram

7805 is a three terminal linear voltage regulator IC with a fixed output voltage of 5V which is useful in a wide range of applications. Currently, the 7805 Voltage Regulator IC is manufactured by Texas Instruments, ON Semiconductor, STMicroelectronics, Diodes incorporated, Infineon Technologies, etc. We have used it to convert our 12V power input to 5V power output for our ESP8266 microcontroller because it has the threshold of 5V. ESP8266 is our main processing unit which takes in the data serially and transmits it to the google sheets and also sends the data to android app. For the sake of simulation we have used the virtual terminal as a means of sending the frame format manually.

Chapter 3
HARDWARE COMPONENT

Chapter 3

Hardware Components

3.1 ESP8266



Figure 3.1: ESP8266

ESP8266 is Wi-Fi enabled system on chip (SoC) module developed by Espressif system. It is mostly used for development of IoT (Internet of Things) embedded applications. ESP8266 comes with capabilities of

- 1) 2.4 GHz Wi-Fi (802.11 b/g/n, supporting WPA/WPA2),
- 2) general-purpose input/output (16 GPIO),
- 3) Inter-Integrated Circuit (I2C) serial communication protocol,
- 4) analog-to-digital conversion (10-bit ADC)
- 5) Serial Peripheral Interface (SPI) serial communication protocol,
- 6) I2S (Inter-IC Sound) interfaces with DMA(Direct Memory Access),
- 7) UART (on dedicated pins, plus a transmit-only UART can be enabled on GPIO2), and pulse-width modulation (PWM).

It employs a 32-bit RISC CPU based on the Tensilica Xtensa L106 running at 80 MHz (or overclocked to 160 MHz). It has a 64 KB boot ROM, 64 KB instruction RAM and 96 KB data RAM. External flash memory can be accessed through SPI. ESP8266 module is low cost standalone wireless transceiver that can be used for end-point IoT developments.

To communicate with the ESP8266 module, microcontroller needs to use set of AT commands. Microcontroller communicates with ESP8266-01 module using UART having specified Baud rate. There are many third-party manufacturers that produce different modules based on this chip. So, the module comes with different pin availability options like,

ESP-01 comes with 8 pins (2 GPIO pins) – PCB trace antenna.

ESP-02 comes with 8 pins, (3 GPIO pins) – U-FL antenna connector.

ESP-03 comes with 14 pins, (7 GPIO pins) – Ceramic antenna.

ESP-04 comes with 14 pins, (7 GPIO pins) – No ant.

3V3: - 3.3 V Power Pin.

GND: - Ground Pin.

RST: - Active Low Reset Pin.

EN: - Active High Enable Pin.

TX: - Serial Transmit Pin of UART.

RX: - Serial Receive Pin of UART.

GPIO0 and GPIO2: - General Purpose I/O Pins.

These pins decide what mode (boot or normal) the module starts up in. It also decides whether the TX/RX pins are used for Programming the module or for serial I/O purpose. To program the module using UART, Connect GPIO0 to ground and GPIO2 to VCC or leave it open. To use UART for normal Serial I/O leave both the pins open (neither VCC nor Ground).

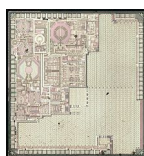


Figure 3.2: Actual image of the Die of ESP chip

Name	Active pins	Pitch	Form factor	LEDs	Antenna	Shielded	Dimensions (mm)	Notes
ESP-WROOM-02 ^[16]	18	1.5 mm	2×9 castellated	No	PCB trace	Yes	18 × 20	FCC ID 2AC7Z-ESPWROOM02.
ESP-WROOM-02D ^[17]	18	1.5 mm	2×9 castellated	No	PCB trace	Yes	18 × 20	FCC ID 2AC7Z-ESPWROOM02D. Revision of ESP-WROOM-02 compatible with both 150-mil and 208-mil flash memory chips.
ESP-WROOM-02U ^[17]	18	1.5 mm	2×9 castellated	No	U.FL socket	Yes	18 × 20	Differs from ESP-WROOM-02D in that includes an U.FL compatible antenna socket connector.
ESP-WROOM-S2 ^[18]	20	1.5 mm	2×10 castellated	No	PCB trace	Yes	16 × 23	FCC ID 2AC7Z-ESPWROOMS2.

Figure 3.3: Different versions of ESP

Vendors have consequently created a multitude of compact printed circuit board modules based around the ESP8266 chip. Some of these modules have specific identifiers, including monikers such as "ESP-WROOM-02" and "ESP-01" through "ESP-14"; while other modules might be ill-labeled and merely referred to by a general description — e.g., "ESP8266 Wireless Transceiver." ESP8266-based modules have demonstrated themselves as a capable, low-cost, networkable foundation for facilitating end-point IoT developments. Espressif's official modules are presently ESP-WROOM-02 and ESP-WROOM-S2. The Ai-Thinker modules are succinctly labeled ESP-01 through ESP-14. (Note: many people refer to the Ai-Thinker modules with the unofficial monikers of "ESP8266-01" through "ESP8266-14" for clarity.)

ESP8266 based development boards/modules often incorporate a surface-mount PCB module, a on-board USB-to-serial bridge, and breakout to 0.1 inch pitch connections. For example, the NodeMCU Development Kits use Ai-Thinker modules, the Adafruit Feather HUZZAH uses an Ai-Thinker ESP-12S module with a SiLabs CP2104 USB-to-serial bridge chip, and the WEMOS D1 Mini version 2.3 uses an Ai-Thinker ESP-12S module with a WinChipHead CH340G USB-to-serial bridge chip. Other development boards don't use an intermediary module and instead directly incorporate the chip itself on-board — for example, WEMOS D1 Mini Pro uses ESP8266EX and WEMOS D1 Mini Lite uses ESP8285.

3.2 DAT-400

3.2.1 Introduction



Figure 3.4: Hardware in the Industry

The DAT 400 is composed of a motherboard, on which you can add the options available; the mother-board is housed in a plastic enclosure by a 35mm DIN rail. The transmitter DAT 400 uses screw terminal boards, pitch 5.08 mm, for the electrical connection. The load cell cable must be shielded and channeled away from tension cables to prevent electromagnetic interference. The instrument is powered through the terminals 8 and 9. The power cord must be channeled separately from other cables. The supply voltage is electrically isolated. Power supply voltage: 24 Vdc/ plus or minus 15 percent max. 5W The cell/s cable must not be channeled with other cables, but must follow its own path. The instrument can be connected up to a maximum of 8 load cells of 350 ohm in parallel. The supply voltage of the cells is 5 Vdc and is protected by temporary short circuit. The measuring range of the instrument involves the use of load cells with a sensitivity of up to 3.5 mV / V. The cable of the load cells must be connected to terminals 13-18. In the case of 4-wire load cell cable, jumper the terminals 13 to 16 and 14 to 15.

Connect the cell cable shield to the terminal 9. In the case of the usage of two or more load cells, use special junction boxes (CEM4/C or CSG4/C). Below please find their connection.

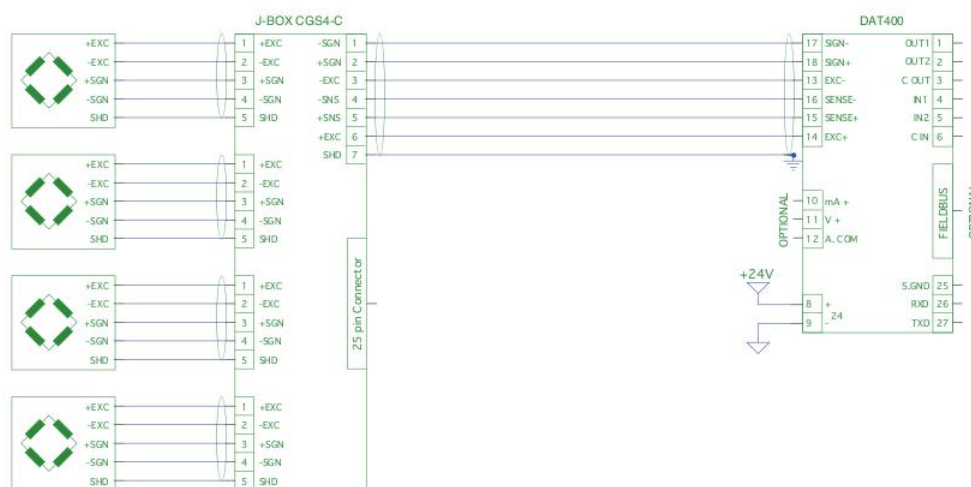


Figure 3.5: Connection of DAT-400 to 4 load-cells

3.2.2 Frame Format

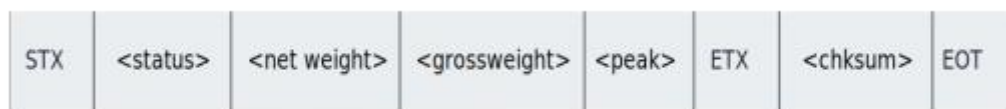


Figure 3.6: Frame Format

Where, STX (start of text) = 0x02h ETX (end of text) = 0x03h EOT (end of transmission) = 0x04. status = an ASCII character that can take the following values:

“S” = stable weight

“M” = weight that is not stable (moving)

“O” = weight greater than the maximum capacity

“E” = weight that cannot be detected.

net weight = field consisting of 6 ASCII characters of net weight.

gross weight = field consisting of 6 ASCII characters of gross weight.

peak = field consisting of 6 ASCII characters of peak.

checksum = 2 ASCII control characters calculated

(considering the characters between STX and ETX excluded.)

The control value is obtained by executing the operation of XOR (or exclusive) of the 8-bit ASCII codes of the characters considered. The result is a character that is expressed in hexadecimal with 2 digits that can take values from “0” to “9” and “A” to “F”. checksum is the ASCII encoding of the two hexadecimal digits. From the above Frame format we need only the net weight and gross weight byte to be shared to our google sheet.

The double quotes enclose constant characters (observe upper and lower case); the and symbols contain variable numeric fields. addr = Serial communication address of the instrument; it is the ASCII character obtained by adding 80h to the number of address (i.e. address 1: Addr = 80h + 01h = 81h). csum = checksum of the string data. It is calculated by performing the exclusive OR (XOR) of all characters from Addr to ETX excluded the latter; the result of the XOR is decomposed into 2 characters by considering separately the upper 4 bits (first character) and lower 4 bits (second character); the 2 characters obtained are then ASCII encoded (example: XOR = 5Dh; csum = “5Dh” namely 35h and 44h).

ETX (end of text) = 0x03h,

EOT (end of transmission) = 0x04h,

ACK (acknowledgment) = 0x06h,

NAK (No acknowledgment) = 0x15h.

If the request is made cyclically, the weight is acquired with a maximum frequency of:

Frequency	Baud Rate
200Hz	115200
50Hz	38400
35Hz	19200
25Hz	9600
8Hz	2400

Figure 3.7: Maximum Frequency to baud rate

3.3 Protocols Used

3.3.1 RS232

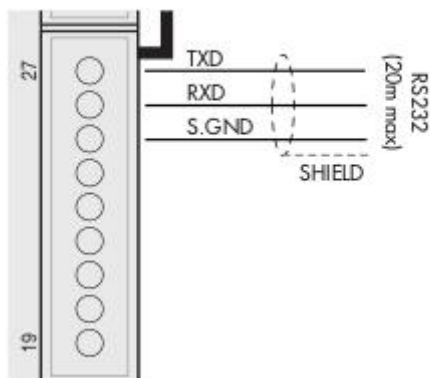


Figure 3.8: RS232 used by DAT-400

The RS232 serial port is always present and handles various protocols. To achieve the serial connection, use a shielded cable, making sure to connect the shield to one of the two ends: to pin 25 if connected on the side of the instrument, to the ground if it is connected on the other side. The cable must not be channeled with power cables; the maximum length is 15 meters (EIA RS-232-C), beyond which you should take the optional RS485 interface.

3.3.2 RS485

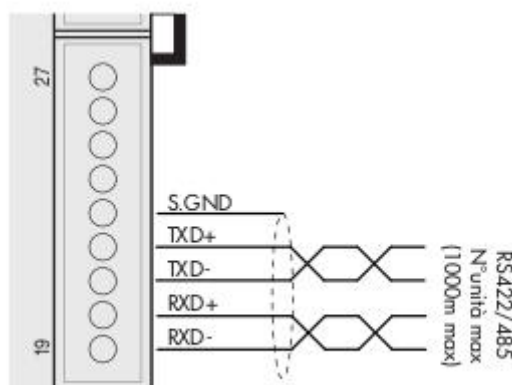


Figure 3.9: RS485 used by DAT-400

The serial port RS485 (2-wire) is present in the model DAT 400/ RS485. To achieve the serial connection, use a suitable shielded cable, making sure to connect the shield to one of the two ends: to pin 23 if connected on the side of the instrument, to the ground if it is connected on the opposite side. The cable should not be channeled with the power cables.

3.4 Front Panel of DAT-400

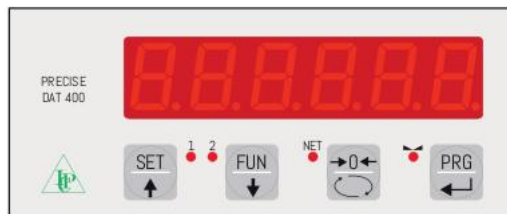


Figure 3.10: Front Panel of DAT-400

The DAT 400 has a bright 6-digit display, 4 status LEDs and four keys. In this operating mode the display shows the weight and the LEDs indicate the status of weight and the setpoints. The set-up parameters are easily accessed and modified through the use of the three front buttons used to select, edit, confirm and save the new settings.

3.4.1 Display

On the 6-digit display, it's usually shown the scale weight. According to the various programming procedures, the display is used for programming of the parameters to be stored in the memory, or the messages that indicate the type of operation being carried out and help therefore the Operator in the management and programming of the instrument.

3.4.2 Led indicators

Below the display there are 4 LED indicators:

- 1 State of the logic output 1 (ON = closed contact OFF = open contact)
- 2 State of the logic output 2 (ON = closed contact OFF = open contact)
- NET The displayed value is the net weight
- 0 IT indicates the condition of stable weight.

3.4.3 Keyboard Functions

KEY	FUNCTIONS DURING THE WEIGHT DISPLAY
	Access to the menu for the programming of the setpoints
	Select the display view (gross weight, net weight). (Long press) Selection of the weight/peak display
	Resetting the displayed value (gross weight, net weight or peak). (Press and hold for 5 sec.) Calibration of zero, to be executed only if its function is enabled in the PARAM menu (see item "0 ALL").
	Sending the weight string on the serial line. (Long press) Access to the quick set-up menu.
+	(Press for 3 sec) Access to the setup menu.
+	(Press for 3 sec) It accesses the keypad lock/unlock menu and auto-off function of the display.

Figure 3.11: Function during the weight display

The instrument is programmed and controlled through the keyboard which has 4 keys, with double functions. The selection of one of the key functions is established automatically by the instrument according to the operation in progress. In general, the management of the programming menus is done by using the SET and FUN keys to scroll through the items; the PRG key is used to enter its sub-menu or programmable parameter, while the 0 button allows you exiting the menu or returning to the top level.

Chapter 4
SOFTWARE REQUIREMENTS

Chapter 4

Software Requirements

4.1 Arduino IDE

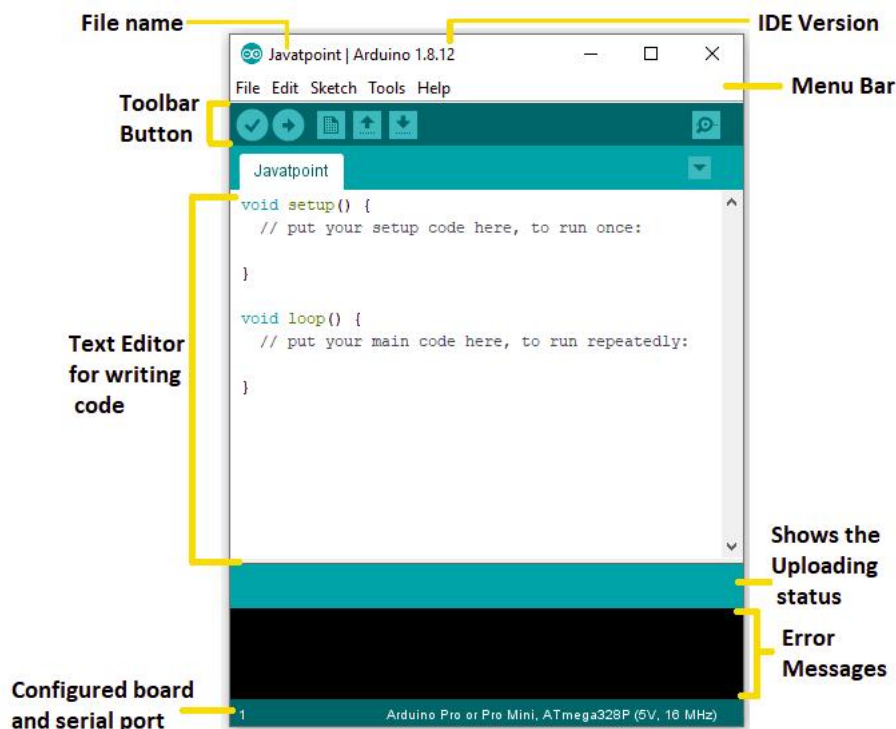


Figure 4.1: Arduino IDE

The 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. It is an official Arduino software, making code compilation too easy that even a common person with no prior technical knowledge can get their feet wet with the learning process. It is available for all operating systems i.e. MAC, Windows, Linux and runs on the Java Platform that comes with inbuilt functions and commands that play a vital role in debugging, editing and compiling the code. A range of Arduino modules available including Arduino Uno, Arduino Mega, Arduino Leonardo, Arduino Micro and many more. Each of them contains a microcontroller on the board that is actually programmed and accepts the information in the form of code. The main code, also known as a sketch, created on the IDE platform will ultimately generate a Hex File which is then transferred and uploaded in the controller on the board. The IDE environment mainly

contains two basic parts: Editor and Compiler where former is used for writing the required code and later is used for compiling and uploading the code into the given Arduino Module. This environment supports both C and C++ languages. 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.'

4.2 Google App Scripts

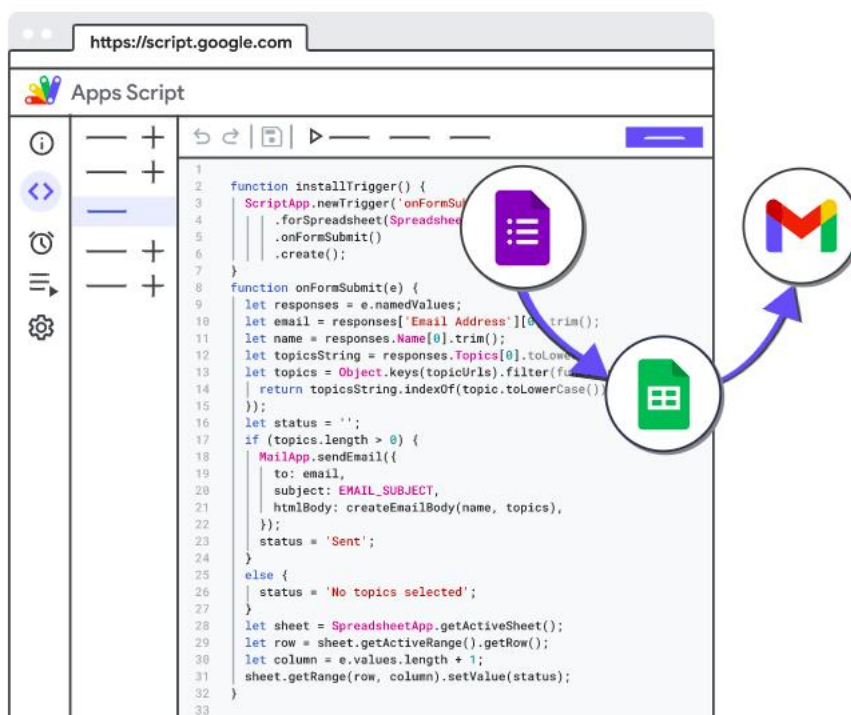


Figure 4.2: Google App Scripts Image

4.2.1 General information

Google Apps Script is a scripting platform developed by Google for light-weight application development in the Google Workspace platform. Google Apps Script was initially developed by Mike Harm as a side project while working as a developer on Google Sheets. Google Apps Script was first publicly announced in May 2009 when a beta testing program was announced by Jonathan Rochelle, then Product Manager for Google Docs. In August 2009 Google Apps Script was subsequently made available to all Google Apps Premier and Education Edition customers. It is based on JavaScript 1.6, but also includes some portions of 1.7 and 1.8 and a subset of the ECMAScript 5 API. Apps Script projects run server-side on Google's infrastructure. According to Google, Apps Script "provides easy ways to automate tasks across Google products and third party services." Apps Script is also the tool that powers the add-ons for Google Docs, Sheets and Slides.

Google Apps Scripts is incredibly powerful and enables complex systems to be built on top of Google services. It can be a great choice when you need to quickly prototype an idea or design a solution that's customizable by non-technical users. A great way to make an accessible solution is to build on top of products that users are already familiar with.

Google Sheets will be used as a flat-file database to store the blog posts. A flat-file database stores data in plain-text in a single table. In contrast, a relational database captures relationships across tables and enforces

the structure of those relationships to minimize duplication and maximize data integrity. Although more limited, a flat-file structure is easy to get started and is suitable for our use case of a small blog.

4.2.2 Benefits

- Based on JavaScript 1.6 and a selection of JavaScript 1.7 and 1.8.
- Cloud based debugger for debugging App Scripts in the web browser.
- It can be used to create simple tools for an organization's internal consumption.
- It can be used to perform simple system administration tasks.
- Community-based support model.

4.2.3 Limitations

- 1) Processing limitations – As a cloud-based service, Apps Script limits the time that a user's script may run, as well as limiting access to Google services.
- 2) Currently Google Apps Store does not allow direct connection to internal (behind-the-firewall) corporate databases, which is key to building business apps, however, via use of the JDBC service, this can be overcome, if connections are allowed from Google servers to the internal database server. Similarly, lack of other connectivity, such as LDAP connectivity, limits the level to which GAS can be used in the enterprise.
- 3) Due to the cloud nature of Apps Script, functions related to date and time will produce results that seem to be incorrect due to the data crossing time zones. Using Date/Time objects and functions without very precise declaration and thorough testing may result in inaccurate results.

4.3 MIT App inventor

MIT App Inventor is a web application integrated development environment originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT). It allows newcomers to computer programming to create application software(apps) for two operating systems (OS): Android, and iOS, which, as of 8 July 2019, is in final beta testing. It is free and open-source software released under dual licensing: a Creative Commons Attribution ShareAlike 3.0 Unported license, and an Apache License 2.0 for the source code.

It uses a graphical user interface (GUI) very similar to the programming languages Scratch (programming language) and the StarLogo, which allows users to drag and drop visual objects to create an application that can run on Android devices, while a App-Inventor Companion (The program that allows the app to run and debug on) that works on iOS running devices are still under development. In creating App Inventor, Google drew upon significant prior research in educational computing, and work done within Google on online development environments.

App Inventor and the other projects are based on and informed by constructionist learning theories, which emphasize that programming can be a vehicle for engaging powerful ideas through active learning. As such, it is part of an ongoing movement in computers and education that began with the work of Seymour Papert and the MIT Logo Group in the 1960s, and has also manifested itself with Mitchel Resnick's work on Lego Mindstorms and StarLogo. App Inventor also supports the use of cloud data via an experimental Firebase. Firebase Realtime Database component

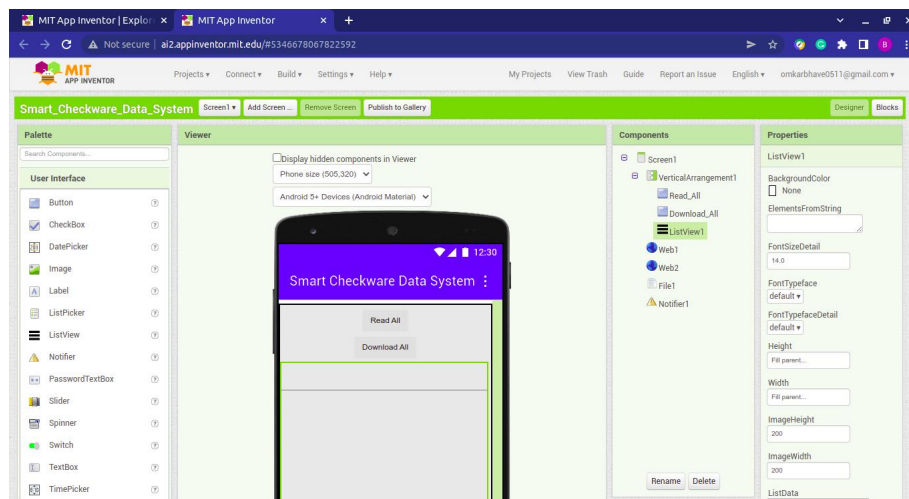


Figure 4.3: MIT App inventor Screen

Chapter 5
CODE

Chapter 5

Codes

5.1 Arduino Code

5.1.1 Initialization

```
//-----  
#include <ESP8266WiFi.h>  
#include <WiFiClientSecure.h>  
#include <SoftwareSerial.h>  
SoftwareSerial MySerial(5, 4); //Red wire D2 & Black wire D1  
//-----  
#define ON_Board_LED 2 //--> Defining an On Board LED, used for indicators when the process of connecting to a wifi router  
//-----SSID and Password of your WiFi router.  
const char* ssid = "*****"; //--> Your wifi name or SSID.  
const char* password = ""; //--> Your wifi password.  
//-----  
  
//-----Host & httpsPort  
const char* host = "script.google.com";  
const int httpsPort = 443;  
//-----  
  
//-----  
int h, d, n;  
String url;  
String t;  
String stringData;  
char someDataFeed[10];  
char keyword[] = "S";  
bool Myflag = false;  
//-----
```

Figure 5.1: Initialization of Arduino Code

The above code initializes the Arduino IDE to include various important header files that may be important for the code to function. We also declare some variables which will be used in the later part of the code. The ESP8266WiFi library provides a wide collection of C++ methods (functions) and properties to configure and operate an ESP8266 module in station and / or soft access point mode. They are described in the following chapters. An access point (AP) is a device that provides access to a Wi-Fi network to other devices (stations) and connects them to a wired network. The ESP8266 can provide similar functionality, except it does not have interface to a wired network. Such mode of operation is called soft access point (soft-AP). The maximum number of stations that can simultaneously be connected to the soft-AP can be set from 0 to 8, but defaults to 4.

5.1.2 Setup Code

```

WiFiClientSecure client; //--> Create a WiFiClientSecure object.

String GAS_ID = "AKfycbwUHWs_qAo6A_Jzl6XnSKlDvcltj88WF-wtEay7sDB7uZsdbyxPD7V-mcdju-_9hLE"; //--> spreadsheet script ID

//===== void setup
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600); // initialize UART with baud rate of 9600 bps
  MySerial.begin(9600);
  delay(500);

  WiFi.begin(ssid, password); //--> Connect to your WiFi router
  Serial.println("");

  pinMode(ON_Board_LED, OUTPUT); //--> On Board LED port Direction output
  digitalWrite(ON_Board_LED, HIGH); //--> Turn off Led On Board

  //-----Wait for connection
  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    //-----Make the On Board Flashing LED on the process of connecting to the wifi router.
    digitalWrite(ON_Board_LED, LOW);
    delay(250);
    digitalWrite(ON_Board_LED, HIGH);
    delay(250);
  }
}

```

Figure 5.2: Setup of Arduino Code

Here we have set up our ESP8266 with its communication Baud rate (9600) and we have also initialized wifi and serial communication. We have also given a loop to always look if the wifi is connected to the given wifi credentials and will stop working if those were incorrect. The Arduino/ESP32 and ESP8266 hardware has built-in support for serial communication on pins 0 and 1 (which also goes to the computer via the USB connection). The native serial support happens via a piece of hardware (built into the chip) called a UART. The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial"). It is possible to have multiple software serial ports with speeds up to 115200 bps.

```

//-----
digitalWrite(ON_Board_LED, HIGH); //-->
//-----
Serial.println("");
Serial.print("Successfully connected to : ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
Serial.println();
//-----

client.setInsecure();
}

```

Figure 5.3: Setup (contd) of Arduino Code

Here the setup routine is continued forward with the important steps that is to give the serial monitor the required information to debug if there are any errors.

5.1.3 Loop Code

```
//===== void loop
void loop() {
  stringData = MySerial.readString();
  stringData.toCharArray(someDataFeed, 10);
  char *pointerToFoundData = strstr(someDataFeed, keyword); //go find keyword
  if (pointerToFoundData != NULL) { //found it
    int positionInString = pointerToFoundData - someDataFeed;
    Myflag = true;
    //Serial.print("Keyword Starts at ");
    //Serial.println(positionInString);

    //now strip out keyword and junk
    char goodData[5];
    strncpy(goodData, &someDataFeed[positionInString + strlen(keyword)], sizeof(goodData));
    goodData[5] = NULL;
    Serial.println(goodData);
    t = String(goodData);
  }
  //else {
  //  Serial.println("NO KEYWORD");
  //}
  if (Myflag == true)
  {
    sendData(t); //---> Calls the sendData Subroutine
    Myflag = false;
  }
}
```

Figure 5.4: Loop routine of Arduino Code

Here the code is repeated unless and until the hardware is stopped or resetted manually. Here we are taking in the transmitted frame format and then it is sliced to only give out the net weight making it easy for the ESP8266 to transmit the data to the sheets.

5.1.4 Google sheets initialization

```
//=====
// Subroutine for sending data to Google Sheets
void sendData(String t) {
  Serial.println("=====");
  Serial.print("connecting to ");
  Serial.println(host);
  Serial.print(n);

  //-----Connect to Google host
  if (!client.connect(host, httpsPort)) {
    Serial.println("connection failed");
    return;
  }
  //-----

  //-----Processing data and sending data

  String string_Weight = String(t);
  //String string_GrossWeight = String(hum, DEC);
  url = "/macros/s/" + GAS_ID + "/exec?weight=" + string_Weight;
}
```

Figure 5.5: Send data sub-routine of Arduino Code

Here we have initialized the google sheets code by making the ESP8266 hit or post to the link everytime it receives some sort of data, this ensures that the data is sent to the sheets as fast as possible.

5.1.5 Google Sheets Data-Upload

```

Serial.print("requesting URL: ");
Serial.println(url);

client.print(String("GET ") + url + " HTTP/1.1\r\n" +
             "Host: " + host + "\r\n" +
             "User-Agent: BuildFailureDetectorESP8266\r\n" +
             "Connection: close\r\n\r\n");

Serial.println("request sent");
//-----
//-----Checking whether the data was sent successfully or not
while (client.connected()) {
  String line = client.readStringUntil('\n');
  if (line == "\r") {
    Serial.println("headers received");
    break;
  }
}
String line = client.readStringUntil('\n');
if (line.startsWith("{\"state\":\"success\"}") {
  Serial.println("esp8266/Arduino CI successfull!");
} else {
  Serial.println("esp8266/Arduino CI has failed");
}
Serial.print("reply was : ");
Serial.println(line);
Serial.println("closing connection");
Serial.println("=====");
Serial.println();

```

Figure 5.6: Send data sub-routine upload part of Arduino Code

Here we send a post request to the google appscript link which updates the data sent by our ESP8266 to the google sheets. If the wifi doesn't have enough speed or its slow the data will not be sent to the Google sheets and on the serial monitor we will get the error that the packet or request is cancelled suggesting that the internet speed might be slow.

5.2 Google Appscript Code

5.2.1 Google Appscript initialization

```
function doGet(e) {
  Logger.log( JSON.stringify(e) );
  var result = 'Ok';
  if (e.parameter == 'undefined') {
    result = 'No Parameters';
  }
  else {
    var sheet_id = '1a6DXmI-82zBF7oVdPcPiEgZRCXXCtTfIAvwX30j2U2w'; // Spreadsheet ID
    var sheet = SpreadsheetApp.openById(sheet_id).getActiveSheet();
    var newRow = sheet.getLastRow() + 1;
    var rowData = [];
    var Curr_Date = new Date();
    rowData[0] = Curr_Date; // Date in column A
    var Curr_Time = Utilities.formatDate(Curr_Date, "Asia/Kolkata", 'HH:mm:ss');
    rowData[1] = Curr_Time; // Time in column B
    for (var param in e.parameter) {
      Logger.log('In for loop, param=' + param);
      var value = stripQuotes(e.parameter[param]);
      Logger.log(param + ':' + e.parameter[param]);
      switch (param) {
        case 'weight':
          rowData[2] = value; // Weight in column C
          result = 'Weight Written on column C';
          break;
        case 'grossweight':
          rowData[3] = value; // Grossweight in column D
          result += ' GrossWeight Written on column D';
          break;
      }
    }
  }
}
```

Figure 5.7: Gsheet Code

The above code gives an idea of using the GScript (Short form of google scripts), here we have taken variables and have linked the appscript to the google sheets with the sheetid variable. This part of code sets up the sheet to start receiving data from the ESP8266.

5.2.2 Google Appscript Read Code

```

    Logger.log(JSON.stringify(rowData));
    var newRange = sheet.getRange(newRow, 1, 1, rowData.length);
    newRange.setValues([rowData]);
  }
  return ManageSheet(e);
  return ContentService.createTextOutput(result);
}
function stripQuotes( value ) {
  return value.replace(/^[\'"]|['"]$/g, "");
}
function doPost(e) {
  return ManageSheet(e);
}
function ManageSheet(e) {
  //READ ALL RECORDS
  if ( e.parameter.func == "ReadAll" ) {
    var ss = SpreadsheetApp.getActive();
    var sh = ss.getSheets()[0];
    var rg = sh.getDataRange().getValues();
    var outString = '';
    for(var row=0 ; row<rg.length ; ++row){
      outString += rg[row].join(',') + '\n';
    }
  }
}

```

Figure 5.8: Gsheet code for the android app

The next part of init method of the gsheet code is to receive the data transmitted by the ESP8266 module and to format it with the given headers (net weight and gross weight), this code does the part of formatting it and making it look neat. We have given a ReadAll method where in it send the data to an android app made with MIT app inventor which read the data and can also be downloaded as a csv file (comma separated values).

5.3 MIT app inventor code

5.3.1 App Inventor Initialization

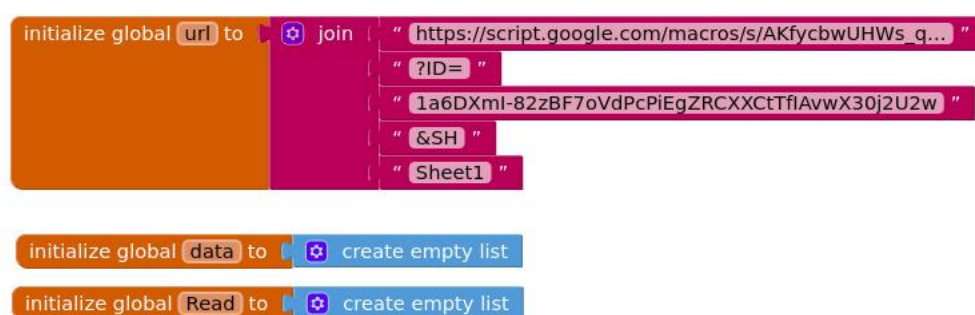


Figure 5.9: Initialization of android app

Here we have given the google sheets url to perform the GET function for the android app to get the data from the sheets.

5.3.2 App Inventor Formatting

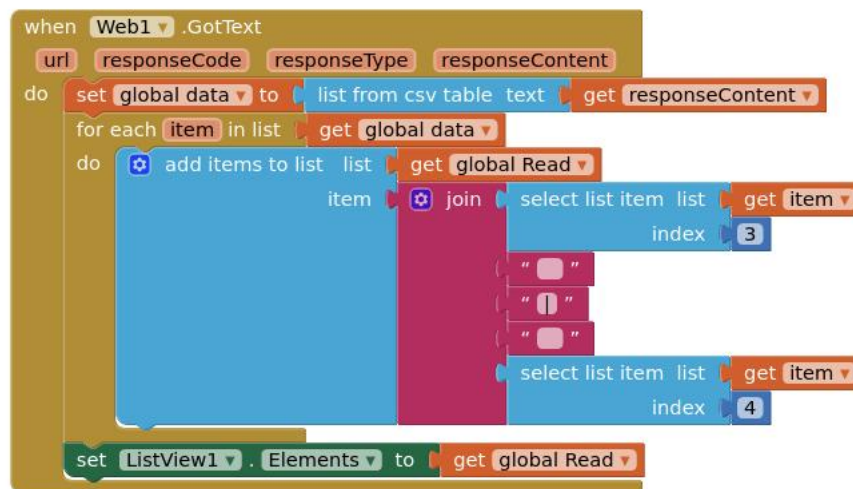


Figure 5.10: Reception of data on android app

This part does the formatting the for the android app to view the data in a formatted way where only net weight and gross weight is shown.

5.3.3 App Inventor Read Function

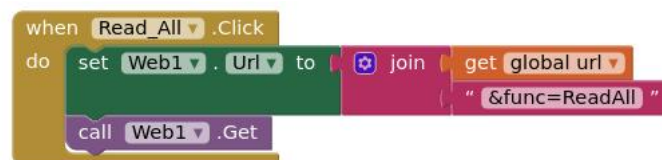


Figure 5.11: Reading data on android app

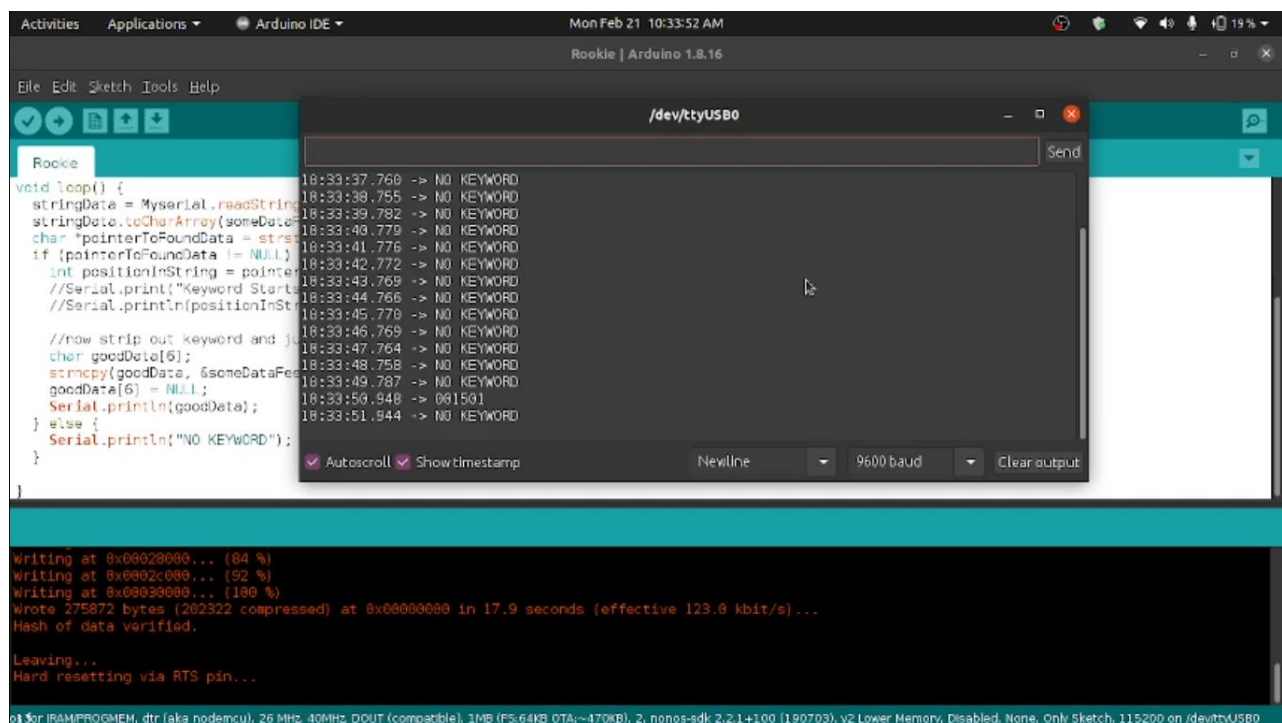
This where the read all button as well as the function comes wherein whenever we click on the "Read all" button on the app it shows the data of the sheets on it in a simplified format. Same for the Download is performed whenever we hit the "Download All" button we request the sheets to send us the csv file of the sheet and store it on the device for further use.

Chapter 6
RESULT

Chapter 6

Results

6.1 Arduino Code Result



The screenshot shows the Arduino IDE interface. The top menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. The toolbar contains icons for opening, saving, and running sketches. The main editor area displays the following C++ code:

```
void loop() {  
  stringData = MySerial.readString();  
  stringData.toCharArray(someData,  
  char *pointerToFoundData = stringData;  
  if (pointerToFoundData != NULL) {  
    int positionInString = pointerToFoundData->find("15.01");  
    //Serial.print("Keyword Starts at ");  
    //Serial.println(positionInString);  
    //now strip out keyword and just the value  
    char goodData[6];  
    strncpy(goodData, &someData[pointerToFoundData->find("15.01")], 6);  
    goodData[6] = NULL;  
    Serial.println(goodData);  
  } else {  
    Serial.println("NO KEYWORD");  
  }  
}
```

The serial monitor window, titled '/dev/ttyUSB0', shows the following output:

```
18:33:37.760 -> NO KEYWORD  
18:33:38.755 -> NO KEYWORD  
18:33:39.782 -> NO KEYWORD  
18:33:40.779 -> NO KEYWORD  
18:33:41.776 -> NO KEYWORD  
18:33:42.772 -> NO KEYWORD  
18:33:43.769 -> NO KEYWORD  
18:33:44.766 -> NO KEYWORD  
18:33:45.770 -> NO KEYWORD  
18:33:46.769 -> NO KEYWORD  
18:33:47.764 -> NO KEYWORD  
18:33:48.758 -> NO KEYWORD  
18:33:49.787 -> NO KEYWORD  
18:33:50.948 -> 001501  
18:33:51.944 -> NO KEYWORD
```

The bottom status bar indicates the board is 'Rookie | Arduino 1.8.16' and the upload progress is 'Writing at 0x00020000... (84 %)'.

Figure 6.1: Result of Arduino Code

When we placed 10kg + 5kg weights on load cell platform, a string was sent by DAT 400 device to arduino. Arduino code extracted the weight and this extracted weight(15.01) is displayed on serial monitor whose screenshot is shown in figure 6.1.

6.2 Google app script code result

Time	Date	Weight	Gross Weight
3/20/2022	6:47:25	15	15
3/21/2022	7:47:25	15	15
3/22/2022	8:47:25	15	15
3/23/2022	9:47:25	15	15
3/24/2022	10:47:25	15	15
3/24/2022	11:14:14	15	15
3/24/2022	11:41:03	15	15
3/24/2022	12:07:52	15	15
3/24/2022	12:34:41	15	15

Figure 6.2: Result of Arduino Code on Google sheets

The above image (or Screenshot) is taken when the machine was plugged in to the actual hardware in the industry, and the readings were taken by placing 10kg and 5 kg weights. Our code works as intended i.e extracts the weight and updates the google sheet runtime. The above figure 6.2 shows the measured weight and hence the sheet gets updated runtime. whenever there is any loss of data i.e no internet connectivity or improper placement of bags, ESP8266 will not receive start of the string character 'S' and hence "No Keyword" string is displayed on serial monitor and in Google sheets.

6.3 MIT app inventor result

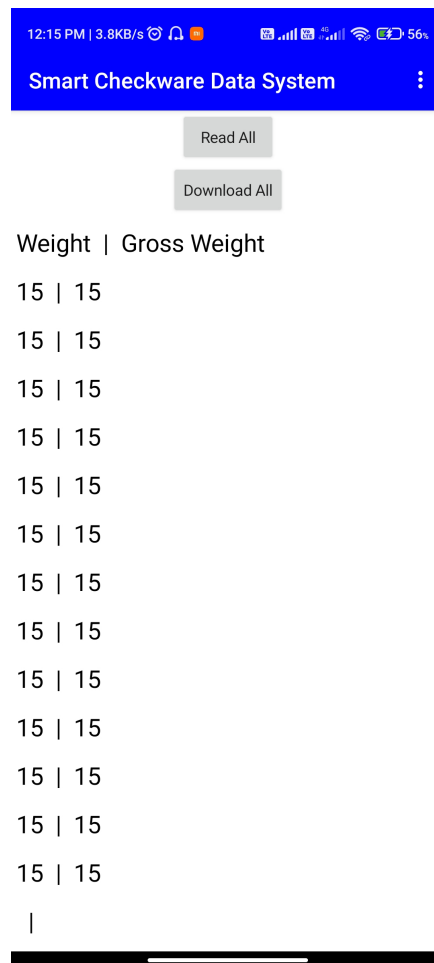


Figure 6.3: Result of the MIT app inventor code

This was made with MIT app inventor. This app displays the google sheet which is updated by the ESP8266 on receiving the data from weighing controller (DAT-400). Whenever weighing controller sends a frame format it is then sliced by the arduino code to get only the net weight and the gross weight whenever the weight is stable and is transmitted to the google sheets. When the user pushes the Read All button on the app it gives this result to the user as shown in the figure 6.3

Chapter 8
CONCLUSION

Chapter 7

Conclusion

We have thoroughly tested the functionality of the hardware and app with real weights and also actual industrial bags wherein we had placed 10 bags of 25 kg each on the load cell platform, which updates the Google sheet with ten entries of 25 Kg and also the android app showed ten entries of 25 kg bags. Thus we tested the functionality of the hardware and the software and it functions as required.

Chapter 8
FUTURE WORK

Chapter 8

Future Work

In the future we will be using PIC16f industrial micro-controller and a custom made android app (made on Android Studio) to monitor and download the data on the phone. This project was designed to directly be implemented in the industrial level, so next goal will be to actually build a hardware to be used on the industrial level and to get it working optimally and in every condition. This is the reason why we are planning to go for PIC16F microcontroller series for the next industrial implementation of the project. Further modifications can be made by using industrial micro-controllers (i.e., PIC16F).

Book References

- [1] ArshdeepBahga and Vijay Madisetti, “Internet of Things: A Hands-on Approach, Universities Press.
- [2] David Hanes ,Gonzalo salgueiro“IoT Fundamentals Networking Technologies,Protocols and Use Cases for Internet of Things”, Cisco Press, Kindle 2017 Edition
- [3] Yashavant Kanetkar , Shrirang Korde :Paperback “21 Internet of Things (IOT) Experiments”
- [4] Andrew Minter ,”Analytics for the Internet of Things(IoT)”,Kindle Edition
- [5] Curtis D. Johnson, Process Control Instrumentation Technology, 7th edition, PHI
- [6] Head-First Python, 2nd edition.
- [7] MicroPython for the Internet of Things: A Beginner’s Guide to Programming with Python on Microcontrollers by Charles Bell.
- [8] Building Arduino Projects for the Internet of Things
- [9] The Internet of things: do-it-yourself projects with Arduino, Raspberry Pi, and BeagleBone Black
- [10] Optimal Digital Control and Filtering for Dynamic Weighing Systems By Mirsad Halimic and Migdat Hodzic