A PROJECT REPORT ON

SMART AMBULANCE SERVICE

A Project Work Submitted to

Department of Electronics and Communication Engineering

Central Institute of Technology, Kokrajhar

Under the supervision and guidance of

DR. BORNALI BORA PATOWARY

(Assistant Professor, Dept. of ECE)

Submitted by –

Chittaranjan Mallik Gau-C-17/L-299

Rajdeep Baruah Gau-C-17/042

Priyanka Bayan Gau-C-17/018

Koustav Dutta Gau-C-17/151

केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार

CENTRAL INSTITUTE OF TECHNOLOGY KOKRAJHAR

Deemed to be University, MHRD, Govt. of India

Kokrajhar, BTAD, Assam 783370

www.cit.ac.in

केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार

CENTRAL INSTITUTE OF TECHNOLOGY KOKRAJHAR

Deemed to be University, MHRD, Govt. of India

Kokrajhar, BTAD, Assam 783370

www.cit.ac.in

CERTIFICATE OF APPROVAL

This is to certify that the work embodied in the project entitled, "SMART AMBULANCE SERVICE" submitted Chittaranjan Mallik, Rajdeep Baruah, Priyanka Bayan, Koustav Dutta to the Department of ELECTRONICS AND COMMUNICATION ENGINEERING of Central Institute of Technology, Kokrajhar is carried out under my direct supervisions and guidance.

The project work has been prepared as per the regulations of Central Institute of Technology and qualifies to be accepted as a Major Project-II, a part of requirements of 8th semester curriculum of Bachelor of Technology in Electronics and Communication Engineering.

_____

Supervisor

DR. BORNALI BORA PATOWARY

(Assistant Professor, Dept. of ECE)

केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार

CENTRAL INSTITUTE OF TECHNOLOGY KOKRAJHAR

Deemed to be University, MHRD, Govt. of India

Kokrajhar, BTAD, Assam 783370

www.cit.ac.in

CERTIFICATE OF APPROVAL

This is to certify that the work embodied in the project entitled "SMART AMBULANCE SERVICE" submitted Chittaranjan Mallik, Rajdeep Baruah, Priyanka Bayan, Koustav Dutta to the Department of ELECTRONICS AND COMMUNICATION ENGINEERING of Central Institute of Technology, Kokrajhar is carried out under my direct supervisions and guidance.

The project work has been prepared as per the regulations of Central Institute of Technology and qualifies to be accepted as a Major Project-II, a part of requirements of 8th semester curriculum of Bachelor of Technology in Electronics and Communication Engineering.

_____

PROF. HARADHAN CHEL

HOD, Department of ECE

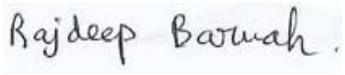
(Assistant Professor, Dept. of ECE)

DECLARATION

We hereby declare that the project work entitled "SMART AMBULANCE SERVICE" is a authenticated work carried out by us under the guidance of Dr. Bornali Bora Patowary for the requirements of Major Project-II, as per of the 8th Semester curriculum of Bachelor of Technology in Electronics and Communication Engineering and this work has not been submitted for similar purpose anywhere else, except to the Department of Electronics and Communication Engineering, Central Institute of Technology, Kokrajhar.

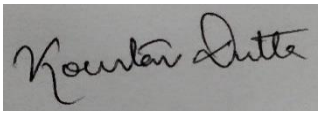
Chittaranjan Mallik

Gau-C-17/L-299



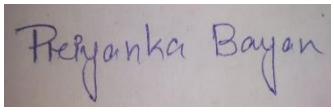
Rajdeep Baruah

Gau-C-17/042



Koustav Dutta

Gau-C-17/151



Priyanka Bayan

Gau-C-17/018

ACKNOWLEDGEMENT

We sincerely express our heartiest gratitude to our project guide Dr. Bornali Bora Patowary (Assistant Prof., Dept. of Electronics and Communication Engineering, Central Institute of Technology, Kokrajhar) for her valuable guidance and vital support throughout the completion of the project.

We would also like to thank Mr. Jeet Dutta (Assistant Professor Dept. of Instrumentation Engineering), Dr. Gunajit Sarma (Assistant Professor Dept. of Humanities and Social Science) for technical and moral support during completion of the project.

Finally, we would like to express our deepest gratitude to our parents for supporting us both morally and financially and encouraging us to give our full dedication to the project work.

ABSTRACT

The world is advancing towards a brighter future and road networking system is playing a vital role in it. At the same time mortality rate due to road traffic accidents is the nightmare which is increasing in an alarming rate. Many people have lost their lives during an accident due to lack of communication. Keeping this in mind, we came up with this idea of introducing a device which can actually help many people during their emergency.

We have developed a system which can detect an accident and its geo location, and can also measure it with the help of an impact detector and accelerometer/ gyroscope. In case of an accident, a buzzer will buzz for 40-50 seconds. If it is a minor accident the person can reset the device in the meantime. But if it is a major accident and the device is not reset within 40-50 seconds the device will collect GPS coordinates and will notify the nearest Hospital/Ambulance service/Police Station/fire brigade and their family members. If the user is not using any vehicle but some emergency occurs then he/she can also call for help with the help of mobile application.

The product has some sensor to detect the accident by using GPS we acquire location coordinates and send data to server where we store all the data and monitor real time database. With the help of GEOQUERY algorithm we can find the nearby services available and notify via an app to track the accident location.

In the present scenario, the victim has to manually contact and describe the situation and location (if the victim knows) otherwise victim have to take help from other people if available. But after using our device and application, the victim doesn't have to contact and describe manually because the whole system is automatic.

For a person who is new to a place, tourists, travelers, who find hard to find emergency services like ambulance, police and fire station of a new place they have to either Google it or take help from others but by using our application's hotkeys the victim can easily contact nearby emergency services irrespective of his/her location.

It will create a very positive impact in our society because everyone wants their loved ones to be safe. Time is a very important factor in case of an emergency. After an accident every second of time is very precious for the victim and it is all about life and death. With the help of this device we can save so many lives that are lost in the road accidents due to poor emergency services. By using this system, the victim will be assured that he will get help in minimum time for sure.

CONTENTS

Chapters **Page**

CHAPTER 1: Introduction

As far as mortality rate is considered, a major percent of deaths occurs as a result of accidents. In this project the overall system to avoid collision at the initial stage has been developed. Moreover if unavoidable accidents occur, the automatic Smart Rescue system and Smart Lifesaver equipment will ensure that the lives of victims are saved to a greater extent.

For this project we have used many hardware components and software applications which will be discussed in details in the coming chapters.

CHAPTER 2: Hardware

2.1 Arduino NANO

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. The Arduino Nano is the preferred board for many projects requiring a small and easy to use microcontroller board. [1][

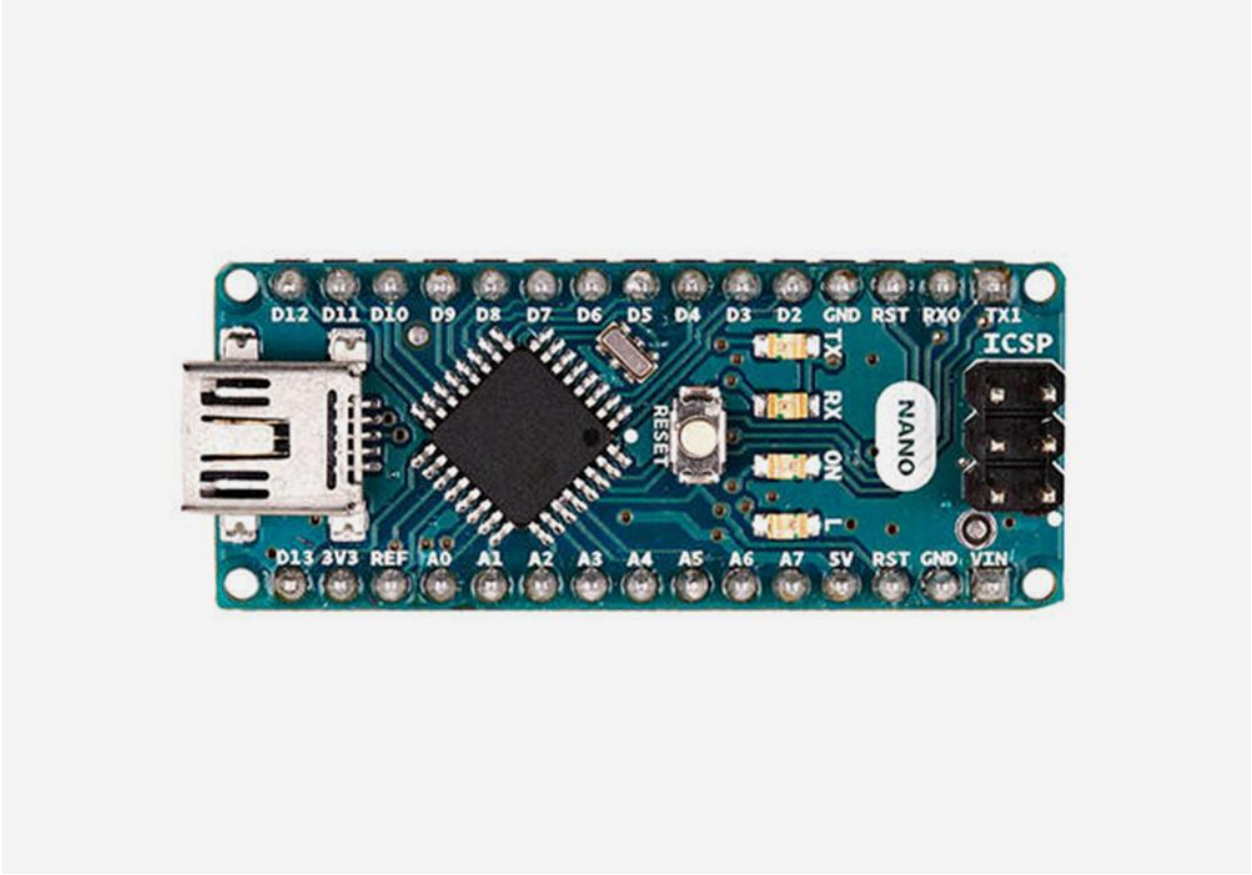


Fig 1. Arduino Nano [2][16/06/2021]

Working Principle of Arduino Nano

It comes with an operating voltage of 5V; however, the input voltage can vary from 7 to 12V. Arduino Nano Pinout contains 14 digital pins, 8 analog Pins, 2 Reset Pins & 6 Power Pins. Each of these digital and analog pins is assigned with multiple functions but their main function is to be configured as input or output. They act as input pins when they are interfaced with sensors, but if you are driving some load then use them as output. Functions like pinMode() and digitalWrite() are used to control the operations of digital pins while analogRead() is used to control analog pins. The analog pins come with a total resolution of 10 bits which measure the value from zero to 5V. Arduino Nano comes with a crystal oscillator of frequency 16 MHz. It is used to produce a clock of precise frequency using constant voltage. [3]

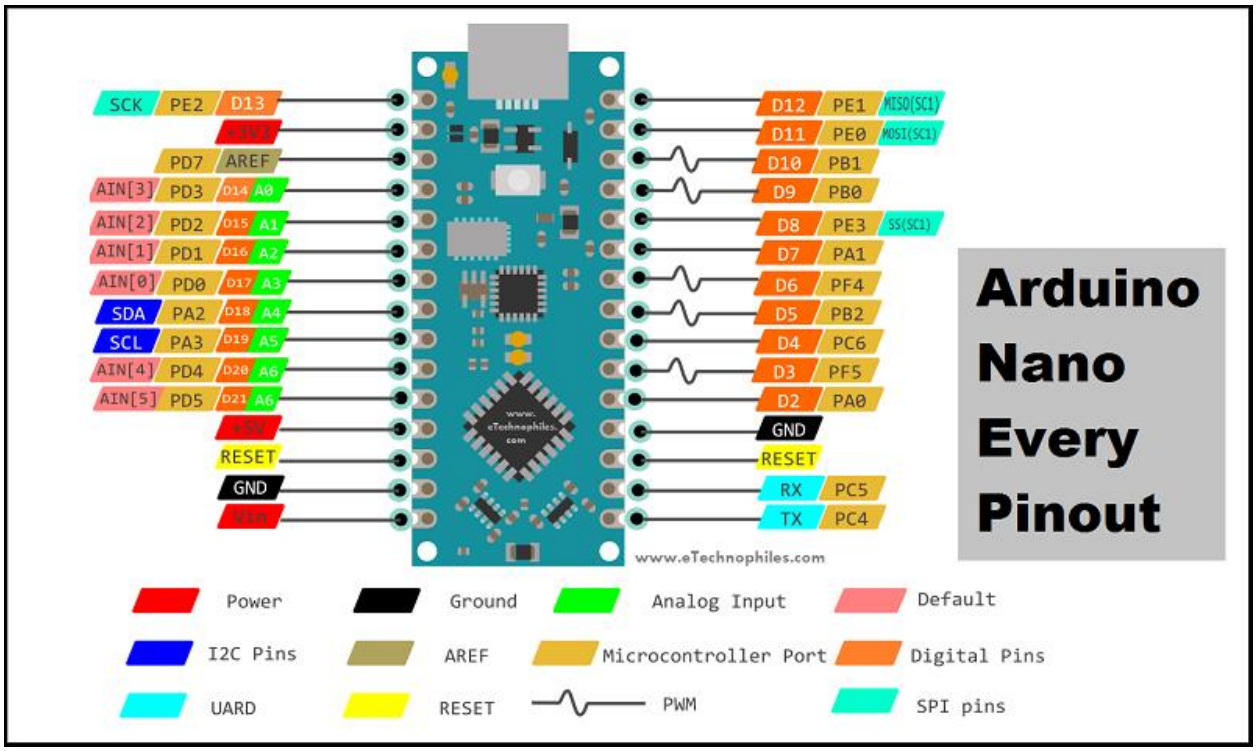


Fig 2.Arduino Nano Pinouts [4][16/06/2021]

2.2 NodeMCU

The NodeMCU ESP8266 development board comes with the ESP-12E module containing ESP8266 chip having Tensilica Xtensa 32-bit LX106 RISC microprocessor. This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs. It has a high processing power with in-built Wi-Fi / Bluetooth and Deep Sleep Operating feature which make it ideal for IoT projects. It is an open-source platform; its hardware design is open for edit/modify/build. [5]

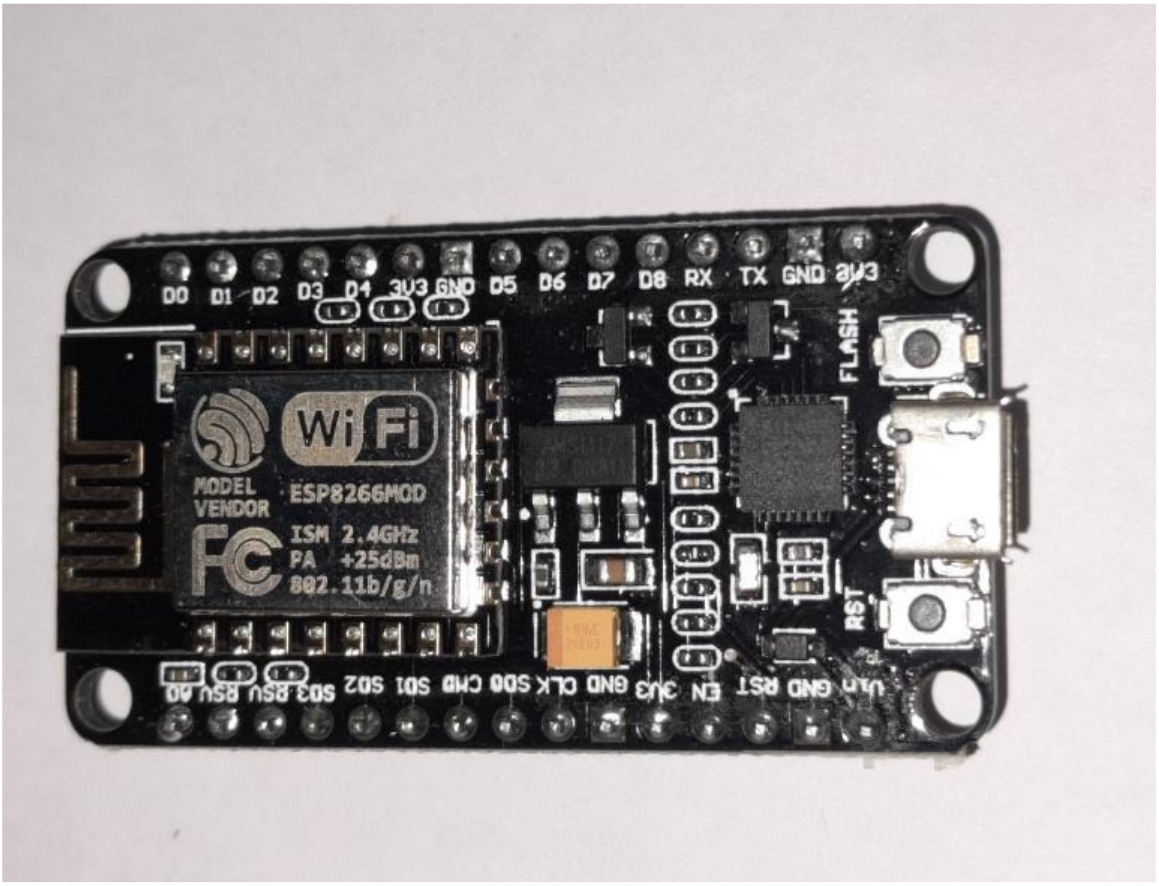


Fig 3. NodeMCU Development Board/kit v1.0 (Version2) [6][16/06/2021]

NodeMCU Dev Kit/board consists of ESP8266 WIFI enabled chip. The ESP8266 is a low-cost Wi-Fi chip developed by Espressif Systems with TCP/IP protocol. NodeMCU uses an on-module flash-based SPIFFS (Serial Peripheral Interface Flash File System) file system. NodeMCU is implemented in C. NodeMCU is an Arduino like device. Its main component is ESP8266. It has Programmable pins. It has built in Wi-Fi. It can get power through micro-USB port. Its cost is low. It can be programmed through multiple programming environments.

The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability produced by manufacturer Espressif Systems in Shanghai, China. ESP8266 offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor. [7]

When ESP8266 hosts the application, and when it is the only application processor in the device, it is able to boot up directly from an external flash. It has integrated cache to improve the performance of the system in such applications, and to minimize the memory requirements [8]

Fig 4. Antenna and Wi-Fi chip of a Node MCU ESP8266 [16/06/2021]

Power Requirement for the NodeMCU

As the operating voltage range of ESP8266 is 3V to 3.6V, the board comes with a LDO voltage regulator to keep the voltage steady at 3.3V. It can reliably supply up to 600mA, which should be more than enough when ESP8266 pulls as much as 80mA during RF transmissions. The output of the regulator is also broken out to one of the sides of the board and labeled as 3V3. This pin can be used to supply power to external components. Power to the ESP8266 NodeMCU is supplied via the on-board Micro USB connector. Alternatively, if you have a regulated 5V voltage source, the VIN pin can be used to directly supply the ESP8266 and its peripherals.

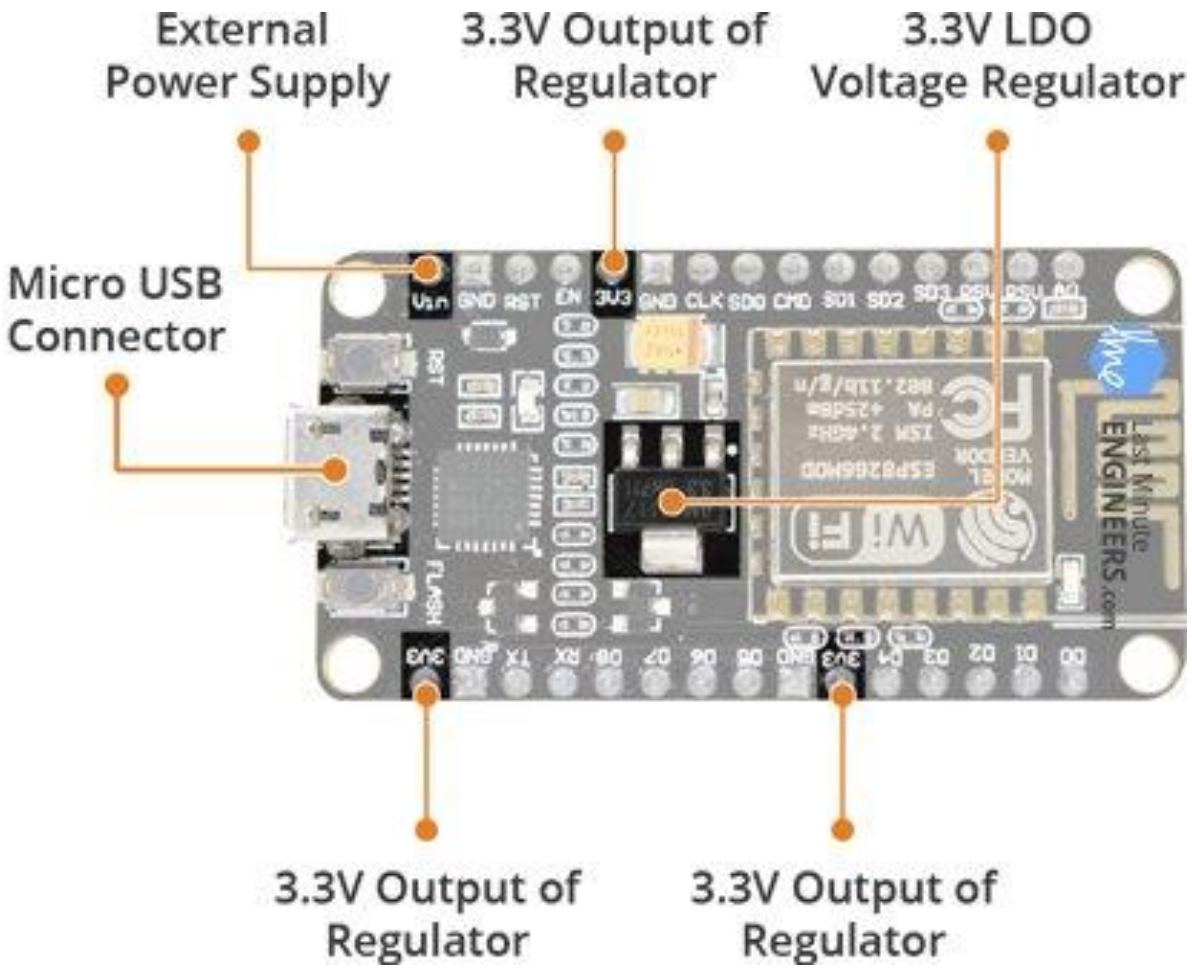


Fig 5. Power input and output point of the NodeMCU[9][16/06/2021]

Peripherals and I/O of the NODEMCU

The ESP8266 NodeMCU has total 17 GPIO pins broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

- ADC channel – A 10-bit ADC channel.
- UART interface – UART interface is used to load code serially.
- PWM outputs – PWM pins for dimming LEDs or controlling motors.
- SPI, I2C & I2S interface – SPI and I2C interface to hook up all sorts of sensors and peripherals.
- I2S interface – I2S interface if you want to add sound to your project.

ESP8266 has pin multiplexing feature (Multiple peripherals multiplexed on a single GPIO pin). Meaning a single GPIO pin can act as PWM/UART/SPI. [9]

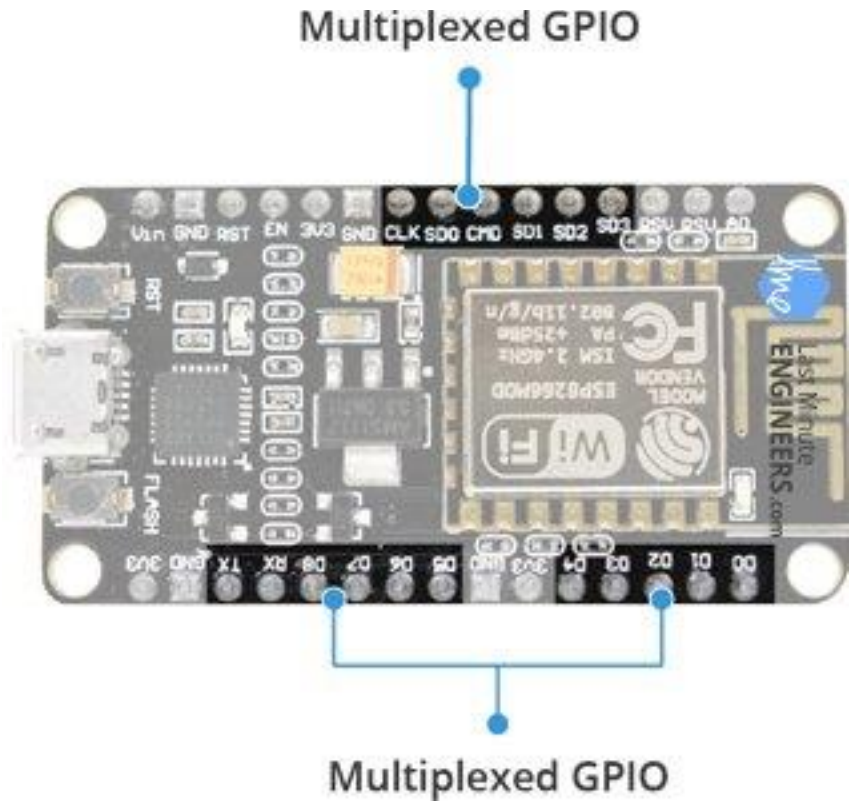


Fig 6. Multiplexed GPIO pins of a NodeMCU [9][16/06/2021]

On-board Switches & LED Indicator of a NODEMCU

The ESP8266 NodeMCU features two buttons. One marked as RST located on the top left corner is the Reset button, used of course to reset the ESP8266 chip. The other FLASH button on the bottom left corner is the download button used while upgrading firmware. The board also has a LED indicator which is user programmable and is connected to the D0 pin of the board.

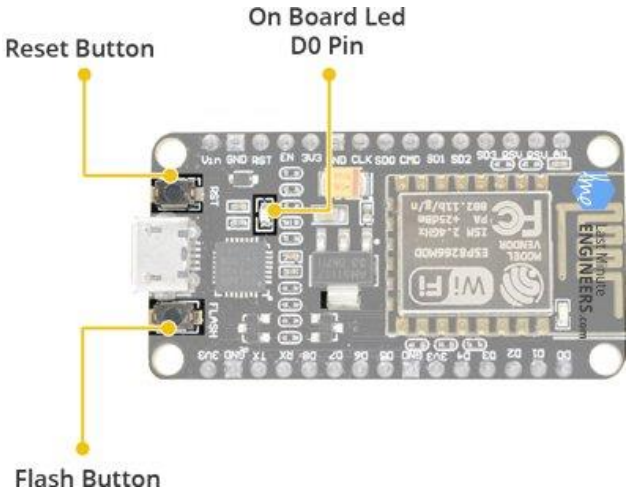


Fig 7. On-board Switches & LED Indicator of a NodeMCU [9] [16/06/2021]

Serial Communication

The board includes CP2102 USB-to-UART Bridge Controller from Silicon Labs, which converts USB signal to serial and allows your computer to program and communicate with the ESP8266 chip.

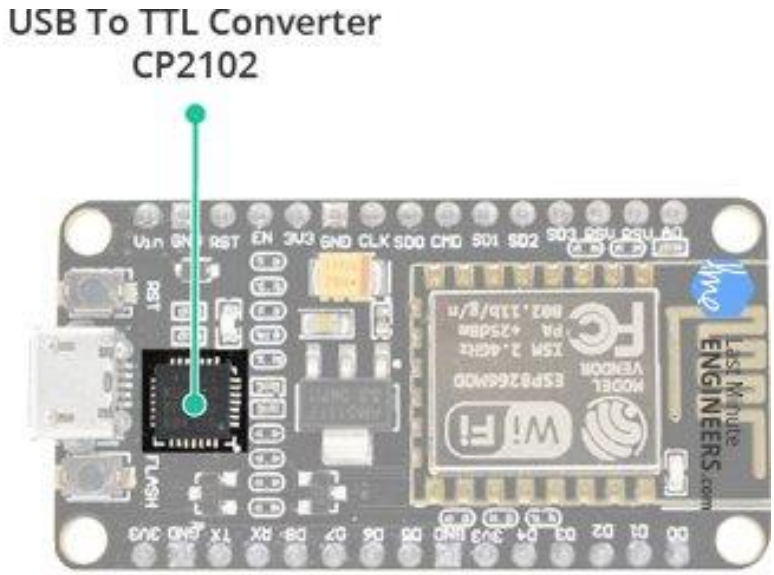


Fig 8. USB to TTL converter on a NODEMCU [9] [16/06/2021]

ESP8266 NodeMCU Pinout

The ESP8266 NodeMCU has total 30 pins that interface it to the outside world. The connections are as follows:

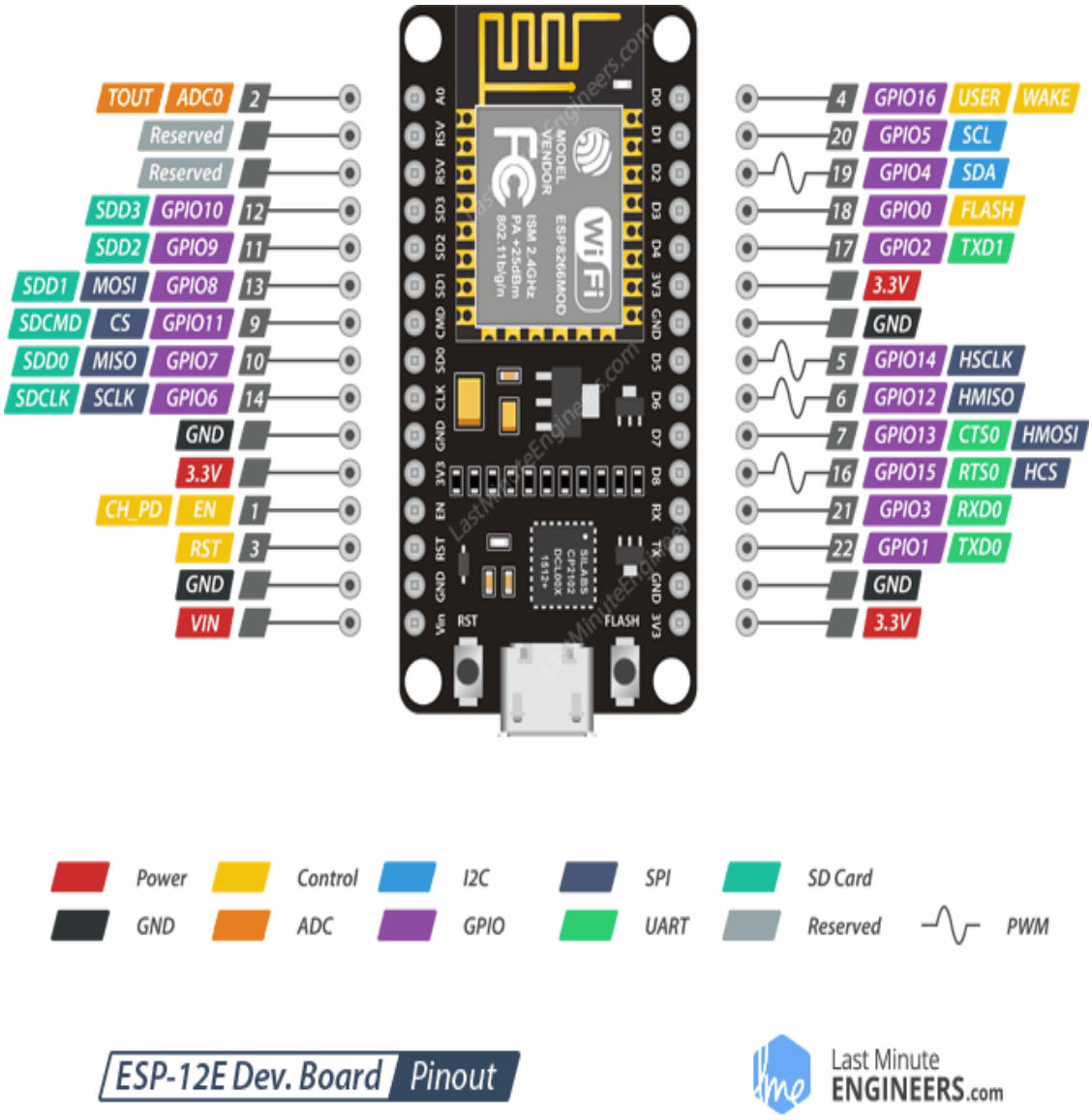


Fig 9. NodeMCU ESP8266 Pinout [9] [16/06/2021]

Power Pins: There are four power pins viz. one VIN pin & three 3.3V pins. The VIN pin can be used to directly supply the ESP8266 and its peripherals, if you have a regulated 5V voltage source. The 3.3V pins are the output of an on-board voltage regulator. These pins can be used to supply power to external components.

GND Pin: GND pin is a ground pin of ESP8266 NodeMCU development board.

12C Pins: 12C pins are used to hook up all sorts of I2C sensors and peripherals in your project. Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized programmatically, and the clock frequency is 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave device.

GPIO Pins: ESP8266 NodeMCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

ADC Channel: The NodeMCU is embedded with a 10-bit precision SAR ADC. The two functions can be implemented using ADC viz. Testing power supply voltage of VDD3P3 pin and testing input voltage of TOUT pin. However, they cannot be implemented at the same time.

UART Pins: ESP8266 NodeMCU has 2 UART interfaces, i.e. UART0 and UART1, which provide asynchronous communication (RS232 and RS485), and can communicate at up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 & CTS0 pins) can be used for communication. It supports fluid control. However, UART1 (TXD1 pin) features only data transmit signal so, it is usually used for printing log.

SPI Pins: ESP8266 features two SPIs (SPI and HSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

- 4 timing modes of the SPI format transfer
- Up to 80 MHz and the divided clocks of 80 MHz
- Up to 64-Byte FIFO

SDIO Pins: ESP8266 features Secure Digital Input/output Interface (SDIO) which is used to directly interface SD cards. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.

PWM Pins: The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 μs to 10000 μs, i.e., between 100 Hz and 1 kHz.

Control Pins: Control Pins are used to control ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.

- EN pin – The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.
- RST pin – RST pin is used to reset the ESP8266 chip.
- WAKE pin – Wake pin is used to wake the chip from deep-sleep.

2.3 GPS module (NEO 6M GPS Module)

The NEO-6M module comes with a dimension of 16 x 12.2 x 2.4 mm package. It has 6 Ublox positioning engines offering unmatched performance. It is a good performance GPS receiver with a compact architecture, low power consumption, and reliable memory options. It is ideal for battery-operated mobile devices considering its architecture and power demands. The Time-to-First-Fix is less than 1 second and it enables it to find the satellites almost instantly. The output is in the format of NMEA standards, which can be decoded to find the coordinates and time of the location. Ublox NEO-6M GPS module can receive coordinates when the indicator light on the module blinks; at that time the GPS has the location of the coordinates where the device is located. When tracking the location of the device when GPS is active, the message sent to the cell phone is a Google Maps address that has been had the actual coordinate position

Fig 10. Ublox Neo 6M GPS Module [10] [17/06/2021]

Working of NEO-6M GPS module

The heart of the module is a NEO-6M GPS chip from u-blox. It can track up to 22 satellites on 50 channels and achieves the industry's highest level of sensitivity i.e. -161 dB tracking, while consuming only 45mA supply current. The u-blox 6 positioning engine also boasts a Time-To-First-Fix (TTFF) of less than 1 second. One of the best features the chip provides is Power Save Mode (PSM). It allows a reduction in system power consumption by selectively switching parts of the receiver ON and OFF. This dramatically reduces power consumption of the module to just 11mA making it suitable for power sensitive applications like GPS wristwatch. The necessary data pins of NEO-6M GPS chip are broken out to a "0.1″ pitch headers. This includes pins required for communication with a microcontroller over UART. [11]

There is an LED on the NEO-6M GPS Module which indicates the status of Position Fix. It'll blink at various rates depending on what state it's in

- No Blinking means It is searching for satellites
- Blink every 1s means Position Fix is found

Converting longitude and latitude to Geohash

Geohash is a system for encoding a (latitude, longitude) pair into a single Base32 string and therefore easier to share, remember and store. In the Geohash system the world is divided into a rectangular grid. Each character of a Geohash string specifies one of 32 subdivisions of the prefix hash. Geo-hashing was originally developed as a URL-shortening service but it is now commonly used for spatial indexing (or spatial binning), location searching, mashups and creating unique place identifiers.

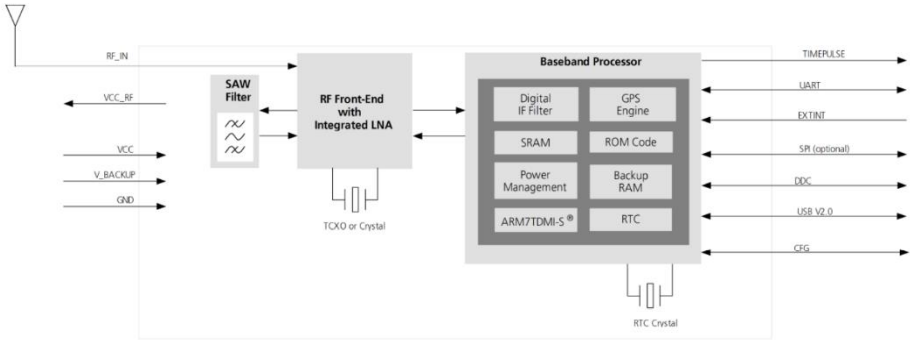


Fig 11. Internal block diagram of NEO-6M GPS module [12] [17/06/2021]

2.4 Accelerometer and Gyroscope

Accelerometer

Accelerometer is basically a sensor which works by sensing the acceleration of gravity and then we can calculate in what direction our device is facing. An accelerometer is a sensor that returns a real valued estimate of acceleration along the x, y and z axes from which velocity and displacement can also be estimated. Accelerometers can be used as motion detectors. Accelerometers are used in vehicle's electronic stability control systems to measure the vehicle's actual movement. [13]Modern accelerometers are the smallest MEMS, consisting of a cantilever beam with proof mass. Accelerometers are available as two-dimensional and three-dimensional forms to measure velocity along with orientation. [14]

Working Principle of Accelerometer

The basic underlying working principle of an accelerometer is such as a dumped mass on a spring. When acceleration is experienced by this device, the mass gets displaced till the spring can easily move the mass, with the same rate equal to the acceleration it sensed. Then this displacement value is used to measure the acceleration. The main working principle of an accelerometer is that it converts mechanical energy into electrical energy. When a mass is kept on the sensor which is actually just like a spring it starts moving down. Since it is moving down it starts experiencing the acceleration. That acceleration when gets converted into an amount of electric signal which is used for the measurements of variation in the position of the device. The accelerometer can be found with both the analog as well as digital form of devices. [15]

Gyroscope

A gyroscope is a device used for measuring or maintaining orientation and angular velocity. Gyroscopes based on other operating principles also exist such as the microchip-packaged MEMS gyroscopes found in electronic devices (sometimes called gyro meters), solid-state ring lasers, fiber optic gyroscopes, and the extremely sensitive quantum gyroscope. Gyroscopes are devices mounted on a frame and able to sense an angular velocity if the frame is rotating. Many classes of gyroscopes exist, depending on the operating physical principle and the involved technology. Gyroscopes can be used to construct gyrocompasses which complement or replace magnetic compasses (in ships, aircraft and spacecraft, vehicles in general) to assist in stability (bicycles, motorcycle and ships) or be used as part of an inertial guidance system. [16]

Working Principle of Gyroscope

A gyroscope sensor works on the principle of conservation of angular momentum. It works by preserving the angular momentum. In a gyroscope sensor, a rotor or a spinning wheel is mounted on a pivot. The pivot allows the rotation of the rotor on a particular axis which is called a gimbal. Here, we will use two gimbals at a time. One gimbal will be mounted on another. This will give the rotor three degrees of freedom. Whenever we spin the rotor of the gyroscope, the gyroscope will continue to point in the same direction. [17]

MPU-6050 Six-Axis (Gyro + Accelerometer)

The MPU-6050 devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die, together with an onboard Digital Motion Processor™ (DMP™), which processes complex 6-axis Motion Fusion algorithms. The device can access external magnetometers or other sensors through an auxiliary master I²C bus, allowing the devices to gather a full set of sensor data without intervention from the system processor. The devices are offered in a 4 mm x 4 mm x 0.9 mm QFN package.

Fig 12. MPU6000 family diagram [18] [16/06/2021]

The InvenSense MotionApps™ Platform that comes with the MPU-6050 abstracts motion-based complexities, offloads sensor management from the operating system, and provides a structured set of APIs for application development.

For precision tracking of both fast and slow motions, the parts feature a user-programmable gyro full-scale range of ±250, ±500, ±1000, and ±2000 °/sec (dps), and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g. Additional features include an embedded temperature sensor and an on-chip oscillator with ±1% variation over the operating temperature range.The MPU6050 in our project so that we can get indications whenever the vehicle has faced an accident. [19]

Fig 13. MPU6050 [20] [17/06/2021]

2.5 Limit Switch (Impact Sensor)

Limit switches are one of the most commonly used electronic components, the limit switches are used in a situation where we need to define the limits. Limit Switches are one of the most commonly used electronic components. These switches are used for defining the limits, Left and Right, Up and Down, etc.

Fig 14. Limit switch module [21] [17/06/2021]

Limit Switches are sensors that tell you when a component is touching it or not. They can be used to prevent mechanisms from moving too far in one direction or another, or - if placed on the outside of the robot - if you've hit something. They do this by sending a signal to the motor controller or roboRIO reporting it has gone the maximum distance.

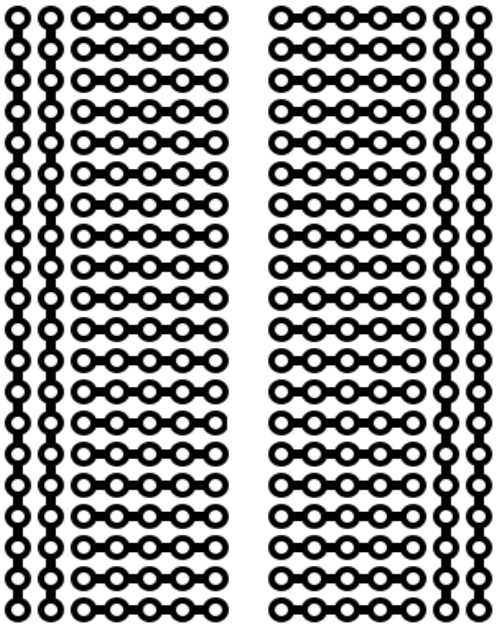
There is nothing special about a limit switch that makes it a limit switch. Any type of switch will do, as long as it can reset itself when nothing is touching it (e.g. a light switch wouldn't work well as a limit switch since it doesn't automatically flip one way or another). Commonly used as limit switches are microswitches, since they have settings for Normally Open and Normally Closed, however, their small size makes them prone to breaking easily. They also require the mechanism to contact it head on, which can cause for awkward placement in a rotating object such as an arm. For these more advanced mechanisms, industrial limit switches might be useful because they can work with many different motions.

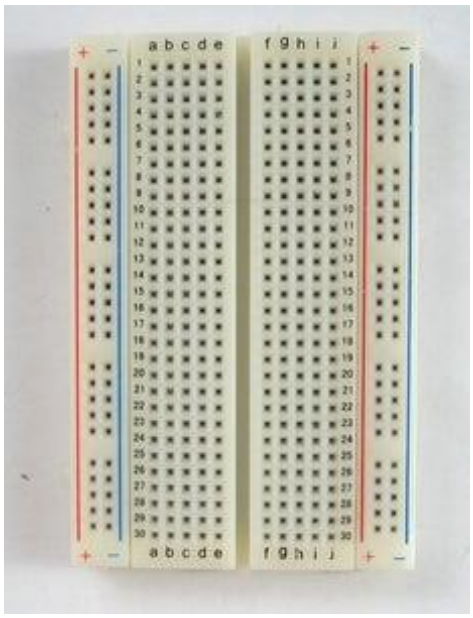
2.6 Breadboard

A breadboard also known as protoboard is a type of solderless electronic circuit building. You can build an electronic circuit on a breadboard without any soldering! Best of all it is reusable. It was designed by Ronald J Portugal of EI Instruments Inc. in 1971.

Building or prototyping circuits on a breadboard are also known as '**breadboarding** '.
There are various types of breadboard. Breadboard can be found in various sizes and functions. Early amateur radio hobbyists used cutting board for bread to prototype their radios thus the name breadboard came.

Some breadboards got built-in power supply, some got power supply rail and some got only the prototyping section.





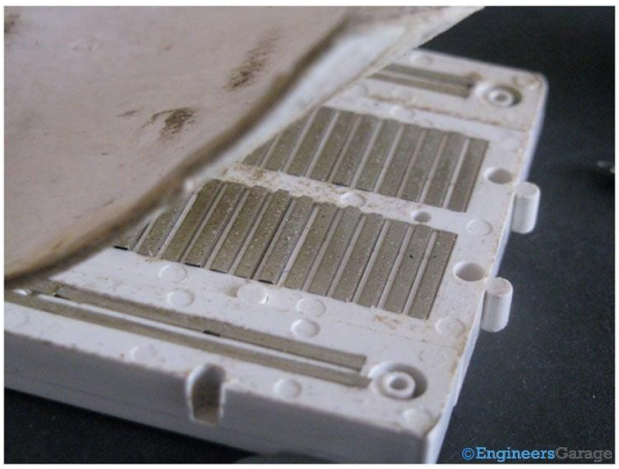


Fig 15. Breadboard [22] [16/06/2021]

Basically, a bread board is an array of conductive metal clips encased in a box made of white ABS plastic, where each clip is insulated with another clips. There are a number of holes on the plastic box, arranged in a particular fashion. A typical bread board layout consists of two types of region also called strips. Bus strips and socket strips. Bus strips are usually used to provide power supply to the circuit. It consists of two columns, one for power voltage and other for ground.
Socket strips are used to hold most of the components in a circuit. Generally it consists of two sections each with 5 rows and 64 columns. Every column is electrically connected from inside. Nickel Silver clips hold the components.

How to use a breadboard?

For our first circuit we will be using resistor networks.

Things we will need

1. Breadboard
2. Resistors 4x100ohm (brown,black,brown) (R1, R2, R3, R4)
3. Multimeter

Step 1

Take R1 and put one lead in **A5** and another lead in **A15**. Take R2 and put its leadsto **B15** and **B25.** Now two resistors are in series mode

Step 2

Now take your multi meter and set to measure resistance. Then put one lead on A5 another on A15.It should measure 100ohm.Now move the lead on A15 to A25 it should measure 200ohm.

Step 3

Take **R3** and put one lead in **D5** another in **D15** and put **R4** in **E15** and **E25.**

Step 4

Now, measure from **D5** and **D15**. It will be **50ohm** and measure **D5** and **E25. It** should measure **100ohm**

So it means **R1** and **R2** are series because they are not in the same lane and **R1** and **R3** are parallel because they are in the same lane.[23]

2.7 Jumper wire

Jumper wires are simply wires that have connector pins at each end, allowing them to be used to connect two points to each other without soldering. Jumper wires are typically used with breadboards and other prototyping tools in order to make it easy to change a circuit as needed. Though jumper wires come in a variety of colors, the colors don't actually mean anything. This means that a red jumper wire is technically the same as a black one. But the colors can be used to your advantage in order to differentiate between types of connections, such as ground or power. Jumper wires typically come in three versions: male-to-male, male-to-female and female-to-female. The difference between each is in the end point of the wire. Male ends have a pin protruding and can plug into things, while female ends do not and are used to plug things into. [24]

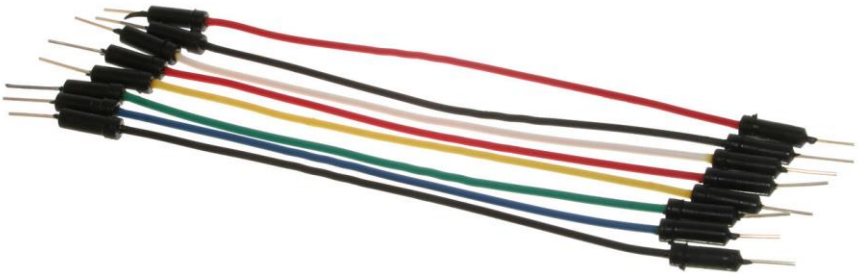


Fig 16. Jumper wires [25] [19/06/2021]

2.8 PCB (Printed Circuit Board)

A printed circuit board, or PC board, or PCB, is a non-conductive material with conductive lines printed or etched. Electronic components are mounted on the board and the traces connect the components together to form a working circuit or assembly. A printed circuit board mechanically supports and electrically connects electronic components using conductive tracks, pads and other features etched from one or more sheet layers of copper laminated onto and/or between sheet layers of a non-conductive substrate. A printed circuit board (PCB) is a thin board made from fiberglass, composite epoxy, or other laminate materials. PCBs are found in various electrical and electronic components such as beepers, radios, radars, computer systems, etc. Different types of PCBs are used based on the applications. [26]

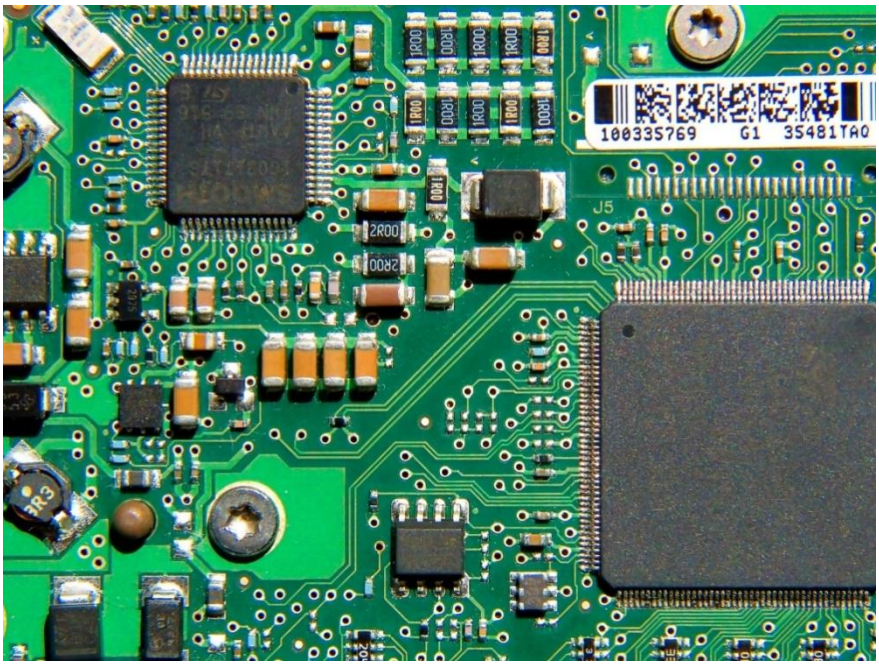


Fig 17. PCB [19/06/2021]

2.9 Soldering

Soldering is a joining process used to join different types of metals together by melting solder. Solder is a metal alloy usually made of tin and leads which is melted using a hot iron. The iron is heated to temperatures above 600 degrees Fahrenheit which then cools to create a strong electrical bond. The three main types of solder are lead-free solder, lead-based solder, and flux-core solders. There is another type known as silver alloy solder. These types are made on the composition of alloys. Apart from this, there are other solder types depending on the form, core style, and application. [27]

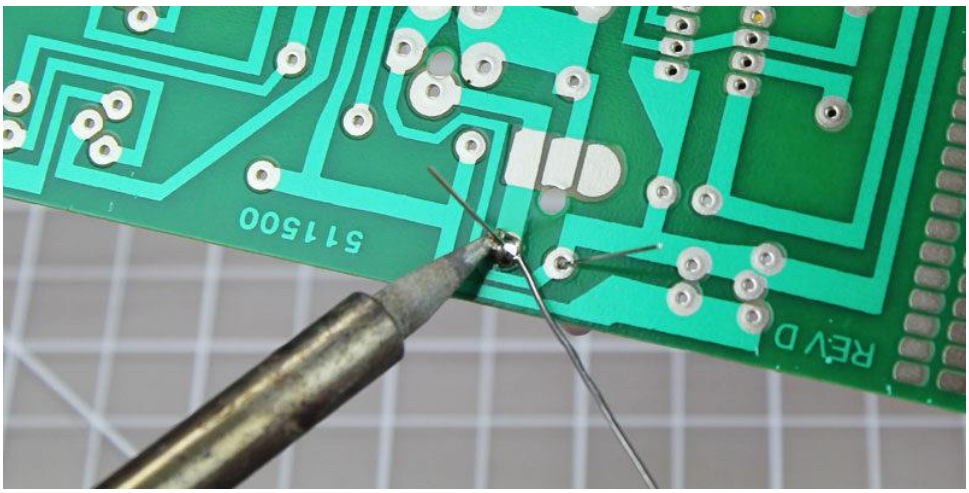


Fig 18. Soldering [19/06/2021]

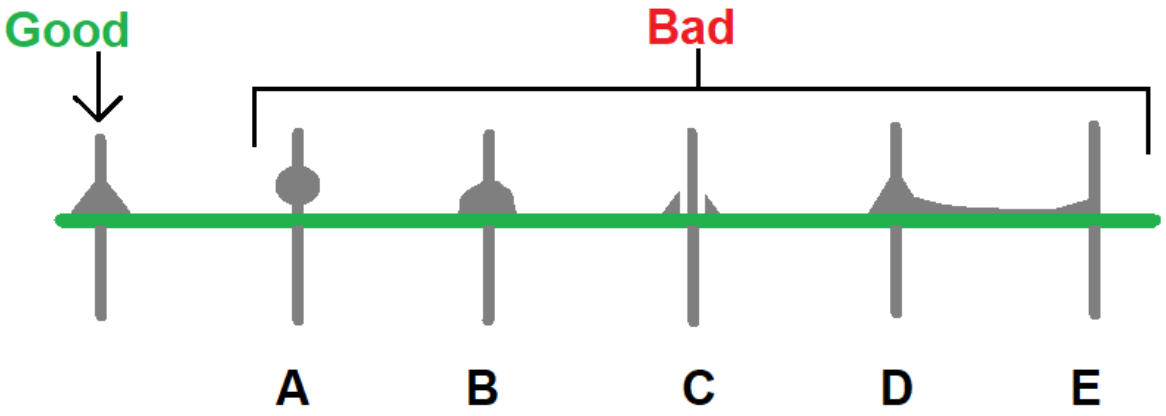
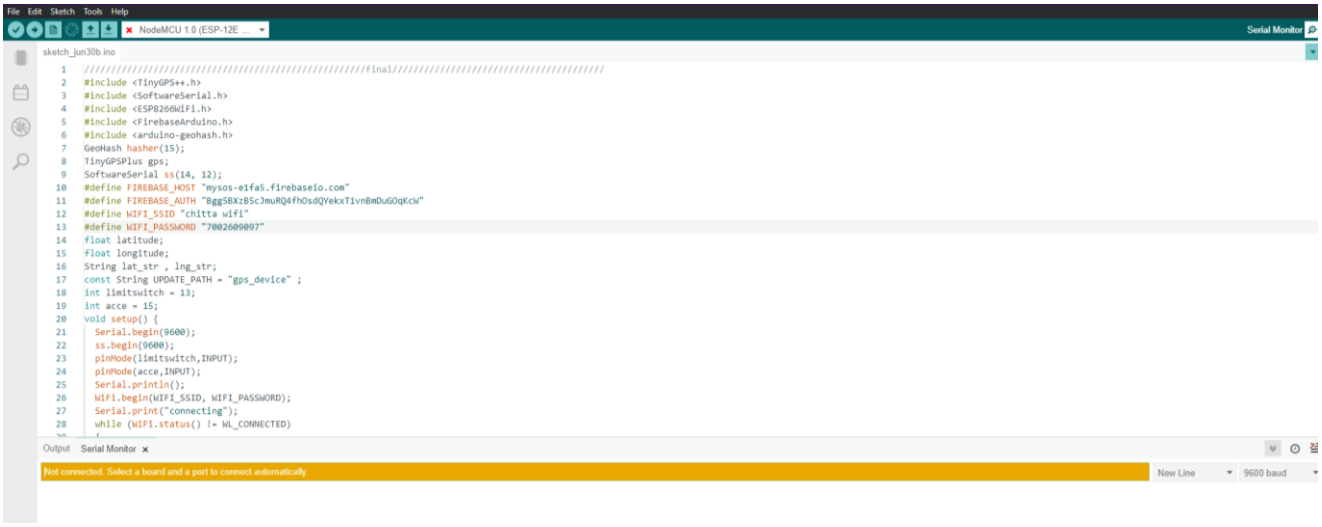


Fig 19. Difference Between good soldering and bad soldering [28] [18/06/2021]

CHAPTER 3: Software and Cloud

3.1 Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.



Writing Sketches

Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.

Verify Checks your code for errors compiling it.

Upload Compiles your code and uploads it to the configured board.

Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"

New Creates a new sketch.

Open Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the **File | Sketchbook** menu instead.

Save Saves your sketch.

Serial Monitor Opens the serial monitor.

Additional commands are found within the five menus: **File**, **Edit**, **Sketch**, **Tools**, **Help**. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

File

- *New* Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.
- *Open* Allows to load a sketch file browsing through the computer drives and folders.
- *Open Recent* Provides a short list of the most recent sketches, ready to be opened.
- *Sketchbook* Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.
- *Examples* Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
- *Close* Closes the instance of the Arduino Software from which it is clicked.
- *Save* Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as." window.
- *Save as...* Allows to save the current sketch with a different name.
- *Page Setup* It shows the Page Setup window for printing.
- *Print* Sends the current sketch to the printer according to the settings defined in Page Setup.
- *Preferences* Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
- *Quit* Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

Edit

- *Undo/Redo* Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
- *Cut* Removes the selected text from the editor and places it into the clipboard.
- *Copy* Duplicates the selected text in the editor and places it into the clipboard.
- *Copy for Forum* Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
- *Copy as HTML* Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
- *Paste* Puts the contents of the clipboard at the cursor position, in the editor.
- *Select All* Selects and highlights the whole content of the editor.
- *Comment/Uncomment* Puts or removes the // comment marker at the beginning of each selected line.
- *Increase/Decrease Indent* Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
- *Find* Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
- *Find Next* Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
- *Find Previous* Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

Sketch

- *Verify/Compile* Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.

- *Upload* Compiles and loads the binary file onto the configured board through the configured Port.

- *Upload Using Programmer* This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so a *Tools -> Burn Bootloader* command must be executed.

- *Export Compiled Binary* Saves a .hex file that may be kept as archive or sent to the board using other tools.

- *Show Sketch Folder* Opens the current sketch folder.

- *Include Library* Adds a library to your sketch by inserting #include statements at the start of your code. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.

- *Add File...* Adds a supplemental file to the sketch (it will be copied from its current location). The file is saved to the data subfolder of the sketch, which is intended for assets such as documentation. The contents of the data folder are not compiled, so they do not become part of the sketch program.

Tools

- *Auto Format* This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.

- *Archive Sketch* Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.

- *Fix Encoding & Reload* Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.

- *Serial Monitor* Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.

- *Board* Select the board that you're using.

- *Port* This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.

- *Programmer* For selecting a hardware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're burninga bootloader to a new microcontroller, you will use this.

- *Burn Bootloader* The items in this menu allow you to burn a bootloader onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino board but is useful if you purchase a new AT mega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the **Boards** menu before burning the bootloader on the target board. This command also set the right fuses.

Help

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

- *Find in Reference* This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

Sketchbook

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the **File** > **Sketchbook** menu or from the **Open** button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the **Preferences** dialog.

Tabs, Multiple Files, and Compilation

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

Before compiling the sketch, all the normal Arduino code files of the sketch (.ino, .pde) are concatenated into a single file following the order the tabs are shown in. The other file types are left as is.

Uploading

Before uploading your sketch, you need to select the correct items from the **Tools > Board** and **Tools > Port** menus. The boards are described below. On the Mac, the serial port is probably something like **/dev/tty.usbmodem241** (for an Uno or Mega2560 or Leonardo) or **/dev/tty.usbserial-1B1** (for a Duemilanove or earlier USB board), or **/dev/tty.USA19QW1b1P1.1** (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably **COM1** or **COM2** (for a serial board) or **COM4**, **COM5**, **COM7**, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be **/dev/ttyACMx** , **/dev/ttyUSBx** or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the **Upload** item from the **Sketch** menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino **bootloader**, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the **Sketch > Import Library** menu. This will insert one or more **#include** statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its **#include** statements from the top of your code.

There is a list of libraries in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch. See these instructions for installing a third-party library.

Third-Party Hardware

Support for third-party hardware can be added to the **hardware** directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the **hardware** directory, then unzip the third-party platform into its own sub-directory. (Don't use "Arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

Serial Monitor

This displays serial sent from the Arduino board over USB or serial connector. To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to **Serial.begin** in your sketch. Note that on Windows, Mac or Linux the board will reset (it will rerun your sketch) when you connect with the serial monitor. Please note that the Serial Monitor does not process control characters; if your sketch needs a complete management of the serial communication with control characters, you can use an external terminal program and connect it to the COM port assigned to your Arduino board.

You can also talk to the board from Processing, Flash, MaxMSP,

Boards

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader.

Arduino Software (IDE) includes the built in support for the boards in the following list, all based on the AVR Core. The Boards Manager included in the standard installation allows to add support for the growing number of new boards based on different cores like Arduino Due, Arduino Zero, Edison, Galileo and so on.

- *Arduino Yún* An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino Uno* An ATmega328P running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Diecimila or Duemilanove w/ ATmega168* An ATmega168 running at 16 MHz with auto-reset.
- *Arduino Nano w/ ATmega328P* An ATmega328P running at 16 MHz with auto-reset. Has eight analog inputs.
- *Arduino Mega 2560* An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino Mega* An ATmega1280 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino Mega ADK* An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino Leonardo* An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.

- *Arduino Micro* An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino Esplora* An ATmega32u4 running at 16 MHz with auto-reset.
- *Arduino Mini w/ ATmega328P* An ATmega328P running at 16 MHz with auto-reset, 8 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Ethernet* Equivalent to Arduino UNO with an Ethernet shield: An ATmega328P running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Fio* An ATmega328P running at 8 MHz with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328P, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino BT w/ ATmega328P* ATmega328P running at 16 MHz. The bootloader burned (4 KB) includes codes to initialize the on-board bluetooth module, 6 Analog In, 14 Digital I/O and 6 PWM.
- *LilyPad Arduino USB* An ATmega32u4 running at 8 MHz with auto-reset, 4 Analog In, 9 Digital I/O and 4 PWM.
- *LilyPad Arduino* An ATmega168 or ATmega132 running at 8 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328P* An ATmega328P running at 16 MHz with auto-reset. Equivalent to Arduino Duemilanove or Nano w/ ATmega328P; 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino NG or older w/ ATmega168* An ATmega168 running at 16 MHz *without* auto-reset. Compilation and upload is equivalent to Arduino Diecimila or Duemilanove w/ ATmega168, but the bootloader burned has a slower timeout (and blinks the pin 13 LED three times on reset); 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Robot Control* An ATmega328P running at 16 MHz with auto-reset.
- *Arduino Robot Motor* An ATmega328P running at 16 MHz with auto-reset.[29]

3.2 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

Project structure

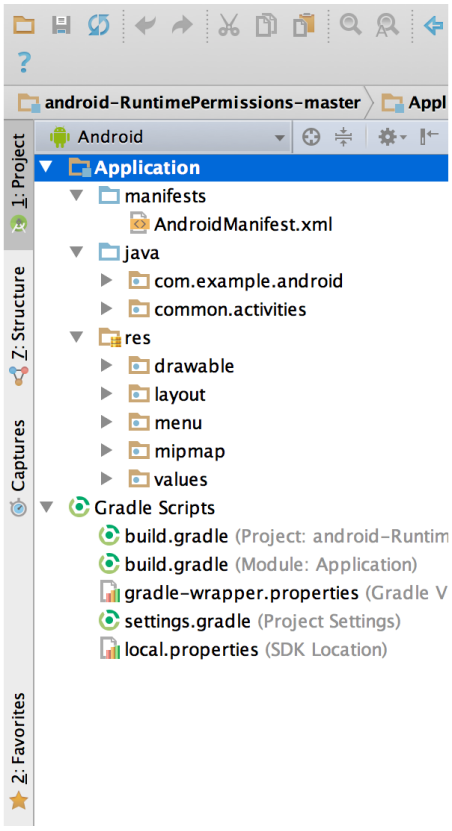


Fig 20. The project files in Android view. [30] [19/06/2021]

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 21. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as **Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

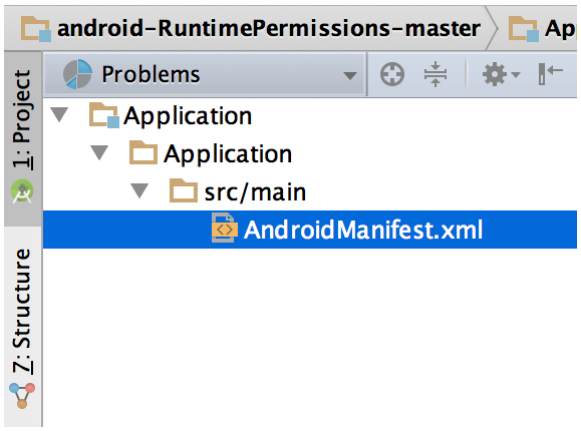


Fig21. The project files in Problems view, showing a layout file with a problem. [31] [19/06/2021]

The User Interference

The Android Studio main window is made up of several logical areas identified in figure 3.

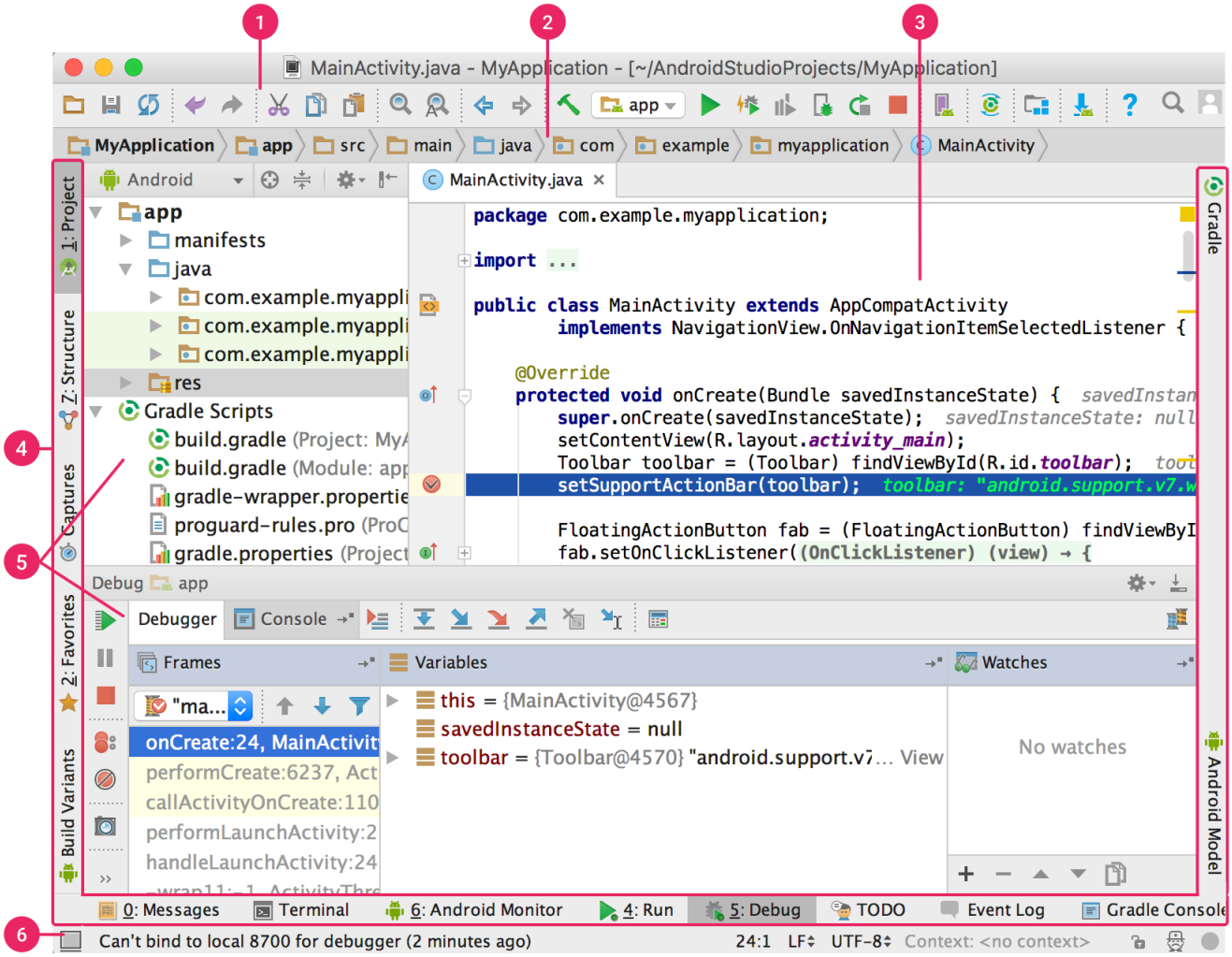


Fig 22. The Android Studio main window [18/06/2021]

1. The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

We can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. We can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful if, for example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

Gradle build system

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app, with different features using the same project and modules.
- Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are named build.gradle. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

Build variants

The build system can help you create different versions of the same application from a single project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs for different device configurations on Google Play.

Multiple APK support

Multiple APK support allows you to efficiently create multiple APKs based on screen density or ABI. For example, you can create separate APKs of an app for the hdpi and mdpi screen densities, while still considering them a single variant and allowing them to share test APK, javac, dx, and ProGuard settings.

Resource shrinking

Resource shrinking in Android Studio automatically removes unused resources from your packaged app and library dependencies. For example, if your application is using Google Play services to access Google Drive functionality and you are not currently using Google Sign-In, then resource shrinking can remove the various draw able assets for the Sign In Button buttons.

Note: Resource shrinking works in conjunction with code shrinking tools, such as ProGuard.

Managing dependencies

Dependencies for your project are specified by name in the build.gradle file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your build.gradle file. Android Studio configures projects to use the Maven Central Repository by default. For more information about configuring dependencies,

Debug and profile tools

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

Inline debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object
- Method return values
- Lambda and operator expressions
- Tooltip values

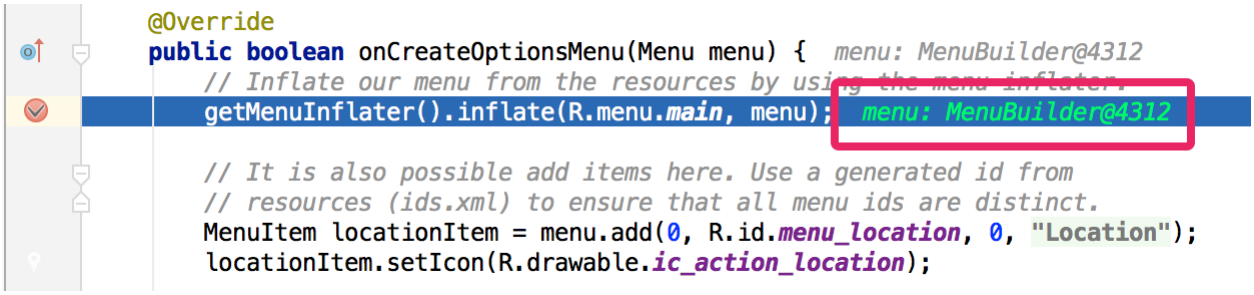


Fig 23. An inline variable value. [22/05/2021]

To enable inline debugging, in the **Debug** window, click **Settings** and select the checkbox for **Show Values Inline**.

Performance profilers

Android Studio provides performance profilers so you can more easily track your app's memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyze network requests. With your app running on a device or emulator, open the **Android Profiler** tab.

Heap dump

When you're profiling memory usage in Android Studio, you can simultaneously initiate garbage collection and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks

Memory Profiler

You can use Memory Profiler to track memory allocation and watch where objects are being allocated when you perform certain actions. Knowing these allocations enables you to optimize your app's performance and memory use by adjusting the method calls related to those actions.

Data file access

The Android SDK tools, such as Systrace, and logcat, generate performance and debugging data for detailed app analysis.

To view the available generated data files, open the Captures tool window. In the list of the generated files, double-click a file to view the data. Right-click any .hprof files to convert them to the standard Investigate your RAM usage file format.

Code inspections

Whenever you compile your program, Android Studio automatically runs configured Lint and other IDE inspections to help you easily identify and correct problems with the structural quality of your code.

The Lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.

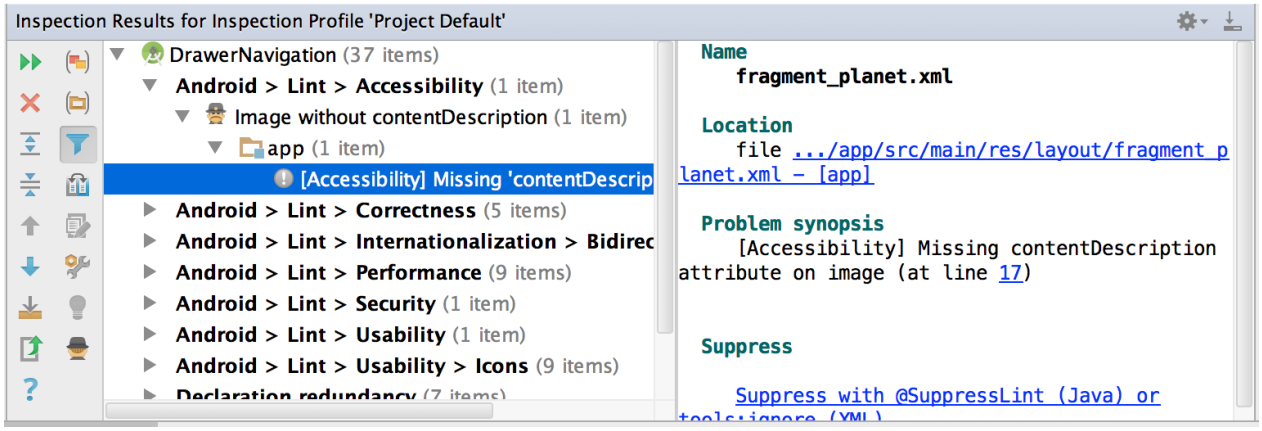


Fig 24. The results of a Lint inspection in Android Studio. [22/05/2021]

In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline your coding workflow.

Annotations in Android Studio

Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the Support-Annotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection.

Log messages

When you build and run your app with Android Studio, you can view adb output and device log messages in the **Logcat** window.

Performance profiling

If you want to profile your app's CPU, memory and network performance, open the Android Profiler, by clicking **View** > **Tool Windows** > **Android Profiler**.

3.3 MIT App Inventor

MIT App Inventor is an online platform designed to teach computational thinking concepts through development of mobile applications. Students create applications by dragging and dropping components into a design view and using a visual blocks language to program application behavior. The MIT App Inventor user interface includes two main editors: the design editor and the blocks editor. The design editor, or designer is a drag and drop interface to lay out the elements of the application's user interface (UI). The blocks editor is an environment in which app inventors can visually lay out the logic of their apps using color-coded blocks that snap together like puzzle pieces to describe the program. To aid in development and testing, App Inventor provides a mobile app called the App Inventor Companion (or just "the Companion") that developers can use to test and adjust the behavior of their apps in real time. In this way, anyone can quickly build a mobile app and immediately begin to iterate and test.

MIT App Inventor introduced mobile app development in educational contexts as a central goal. Prior to its release, most development environments for mobile applications were clunky, only accessible with expertise in systems level or embedded programming, or both. Even with Google's Android operating system and the Java programming language, designing the user interface was a complex task. Further, use of the platform required familiarity with Java syntax and semantics, and the ability to debug Java compilation errors (e.g., misspelled variables or misplaced semicolons) for success. These challenges presented barriers to entry for individuals not versed in computer science,

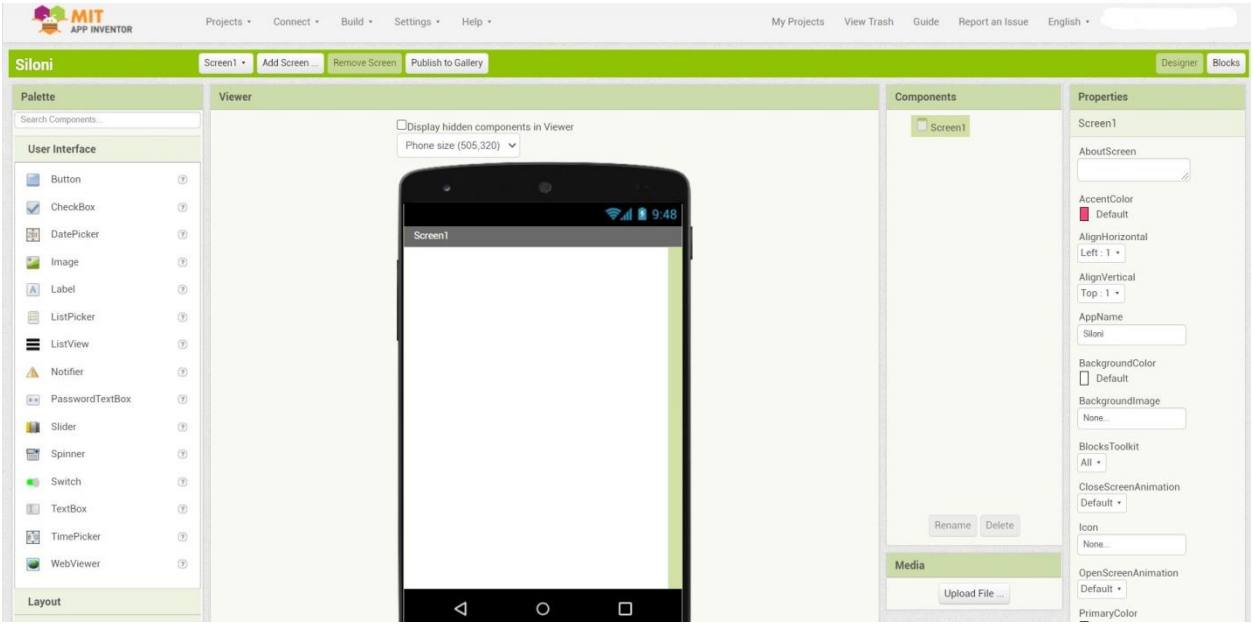


Fig 25. MIT Web Developer Main Screen (ai2.appinventor.mit.edu) [19/06/2021]

3.5 Firebase

Google Firebase is Google-backed application development software that enables developers to develop iOS, Android and Web apps. Firebase provides tools for tracking analytics, reporting and fixing app crashes, creating marketing and product experiment. Firebase gives a complete solution about authentication. Apart from authentication Firebase also manages a very good problem of database. Firebase is considered as web application platform. It helps developers' builds high-quality apps. It stores the data in JavaScript Object Notation (JSON) format which does not use query for inserting, updating, deleting or adding data to it. It is the backend of a system that is used as a database for storing data.

The services available are:

1. **Firebase Analytics:** It provides insight into app usage. It is a paid app measurement solution that also provides user engagement. This unique feature enables the application developer to understand how users are using the application. The SDK has the feature of capturing events and properties on its own and also allows getting custom data.
2. **Firebase Cloud Messaging (FCM):** It is formerly known as Google Clouds Messaging (GCM), FCM is a paid service which is a cross-platform solution for messages and notifications for Android, Web Applications, and IOS.
3. **Firebase Auth:** Firebase Auth supports social login provider like Facebook, Google GitHub, and Twitter. It is a service that can authenticate users using only client-side code and it is a paid service. It also includes a user management system whereby developers can enable user authentication with email and password login stored with Firebase.
4. **Real-time Database:** Firebase provides services like a real-time database and backend. An API is provided to the application developer which allows application data to be synchronized across clients and stored on Firebase's cloud. The client libraries are provided by the company which enables integration with Android, IOS, and JavaScript applications.
5. **Firebase Storage:** It facilitates easy and secure file transfer regardless of network quality for the Firebase apps. It is backed by Google Cloud Storage which is cost-effective object storage service. The developer can use it to store images, audio, video, or other user-generated content.
6. **Firebase Test Lab for Android:** It provides cloud-based infrastructure for testing Android apps. With one operation, developers can initiate testing of their apps across a wide variety of devices and device configurations. The various test results like screenshots, videos and logs are available in the Firebase console. Even if a developer hasn't written any test code for their app, Test Lab can exercise the app automatically, looking for crashes.
7. **Firebase Crash Reporting:** The detailed reports of the errors are created in the app. The errors are grouped into clusters of similar stack traces and triaged by the severity. The other features are: the developer can log custom events to help capture the steps leading up to a crash.
8. **Firebase Notifications:** It enables targeted user notifications for mobile app developers and the services are freely available.

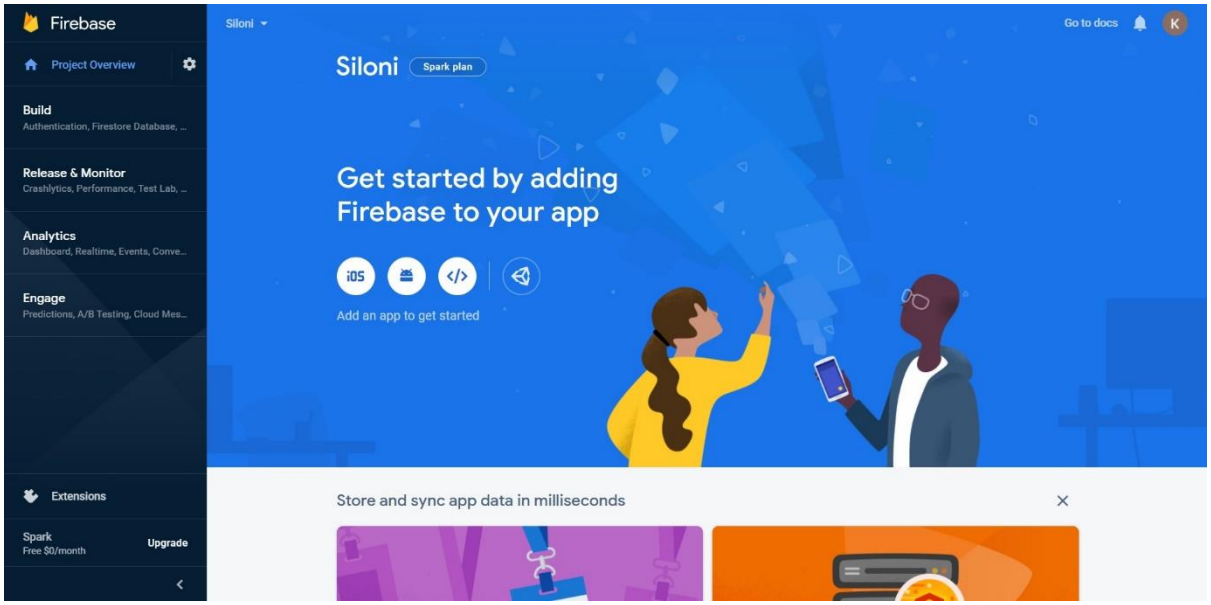


Fig 26. Firebase conosole [19/06/2021]

Using firebase features in Android Application

1. **Authentication:** After adding firebase and authentication dependency to the Android application the user can create login ID.
2. **Database**: Firebase real-time database feature is very easy to use. Once the Firebase and database dependency is added to the app, unstructured data can be added to database.
3. **Storage:** The files like images, audio and video can be stored in the app. The data stored is highly secured and is robust in nature means it resumes from the last point if any network error occurs.
4. **Inserting New Data:** An object class or a map can be used to insert new data. Once the object is created, navigate the Firebase reference to the position where a child can be added.
5. **Updating Data:** Navigate the Firebase reference to the parent of the item that one wants to update. Then a map containing the update values can be created.
6. **Removing Data:** Navigate the Firebase reference to the item that one wants to remove.

CHAPTER 4: Working of Application Software

4.1 User Application

Siloni application is a Google – Map based online emergency application useful to find the specified emergency services like, Police, Fire station and Hospitals, likely to be found in a given circular area. The main objective of this application is to receive various emergency services with ease through just a mobile app. The app contains a Emergency button through which one can avail from emergency service.

The User Application is made using Android Studio.

4.1.1 Designing of application with Android Studio

Activity Main Design

We have put a background of first page then design a logo and put a activity starter button, When Emergency button is pressed then the activity map will be start.

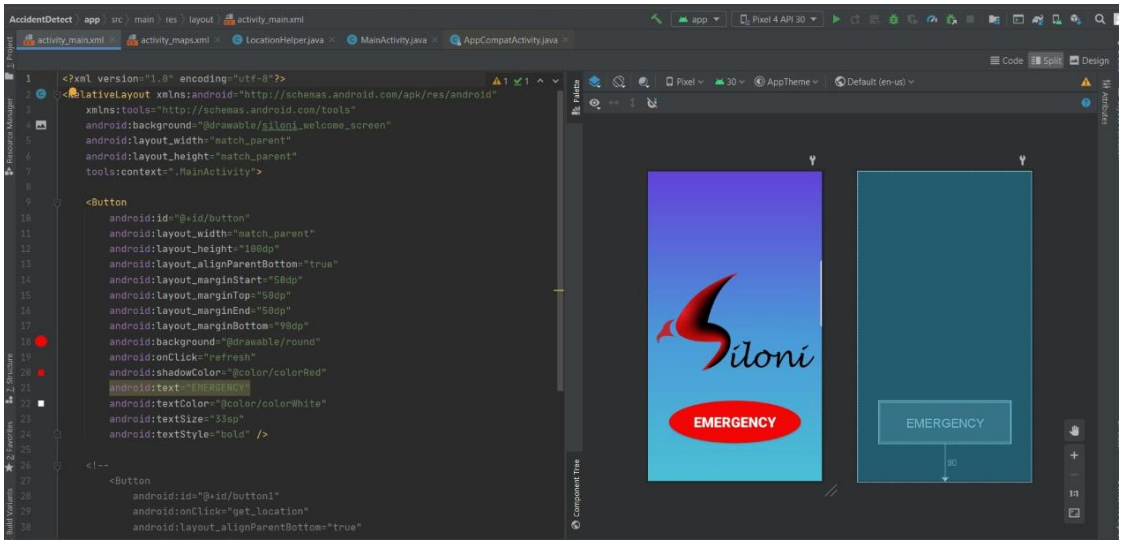


Fig 27. Front page of user application [25/05/2021]

XML code

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.and
roid.com/tools"
android:background="@drawable/siloni_welcome_screen"
android:layout_width="match_parent" android:layout_height="match_parent" tools:context=".MainActivity"><
Button android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:layout_alignParentBottom="true"
    android:layout_marginStart="50dp"
    android:layout_marginTop="50dp"
    android:layout_marginEnd="50dp"
    android:layout_marginBottom="90dp"
    android:background="@drawable/round"
    android:onClick="refresh"
    android:shadowColor="@color/colorRed"
```

```
android:text="EMERGENCY"
android:textColor="@color/colorWhite"
android:textSize="33sp" android:textStyle="bold"/>
<!--<Button
        android:id="@+id/button1"
android:onClick="get_location"
android:layout_alignParentBottom="true"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginStart="50dp"
android:layout_marginTop="80dp"
android:layout_marginEnd="50dp"
android:layout_marginBottom="100dp"
        android:text="GET ACCIDENT LOCATION" />
    --></RelativeLayout>
```

Activity Map Design

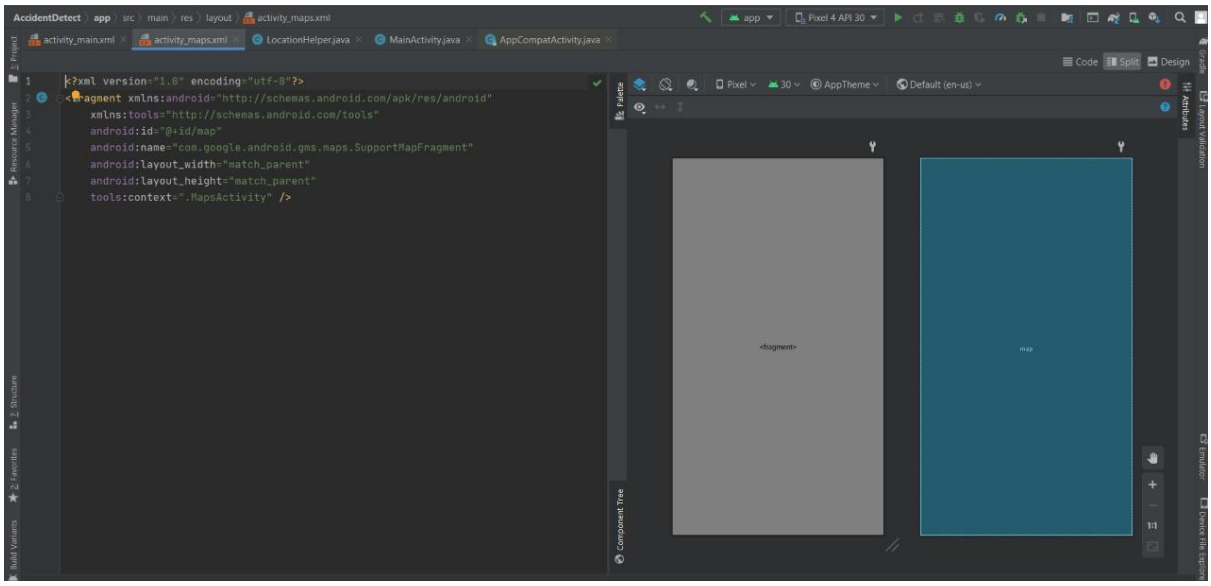


Fig 28. Map activity of user application [25/05/021]

XML code

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools=http://schemas.android.com/tools
 android:id="@+id/map"
android:name="com.google.android.gms.maps.SupportMapFragment"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MapsActivity"/>
```

4.1.2 Coding and testing with Android Studio

Main Activity (Front page)

JAVA CODE

```
package com.example.accidentdetect;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void refresh(View view){
startActivity(new Intent(this,MapsActivity.class));
    }
    public void get_location(View view){
startActivity(new Intent(getApplicationContext(),MapsActivity.class));
    }
}
```

Map Activity (Calling Latitude and Longitude and sending to Firebase database

JAVA CODE

```
package com.example.accidentdetect;
import android.Manifest;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.provider.Settings;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.core.app.ActivityCompat;
import androidx.fragment.app.FragmentActivity;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.database.DataSnapshot;
```

```
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.karumi.dexter.Dexter;
import com.karumi.dexter.PermissionToken;
import com.karumi.dexter.listener.PermissionDeniedResponse;
import com.karumi.dexter.listener.PermissionGrantedResponse;
import com.karumi.dexter.listener.PermissionRequest;
import com.karumi.dexter.listener.single.PermissionListener;
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback,
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener,
com.google.android.gms.location.LocationListener{
    private GoogleMap mMap;
    private GoogleApiClient mGoogleApiClient;
    private Location mLocation;
    private LocationManager mLocationmanager;
    private LocationRequest mLocationRequest;
    private com.google.android.gms.location.LocationListener listener;
    private long UPDATE_INTERVAL = 2000;
    private long FASTEST_INTERVAL = 5000;
    private LocationManager locationManager;
    private LatLng latLng;
    private boolean isPermission;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        if (requestSinglePermission()) {
            // Obtain the SupportMapFragment and get notified when the map is ready to be used.
            SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
.findFragmentById(R.id.map);
            mapFragment.getMapAsync(this);

            mGoogleApiClient = new GoogleApiClient.Builder(this)
.addConnectionCallbacks(this)
.addOnConnectionFailedListener(this)
.addApi(LocationServices.API)
.build();
            mLocationmanager = (LocationManager) this.getSystemService(LOCATION_SERVICE);
checkLocation();
                }    }
    private boolean checkLocation() {
        if(!isLocationEnabled()){
showAlert();
        }
        return isLocationEnabled();
    }
    private void showAlert() {
        final AlertDialog.Builder dialog = new AlertDialog.Builder(this);
dialog.setTitle("Enable Location")
.setMessage("Please turn on location Service")
.setPositiveButton("LOCATION SETTING",new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    Intent myIntent=new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                    startActivity(myIntent);
                }              })
.setNegativeButton("CANCEL", new DialogInterface.OnClickListener() {
```

```
        @Override
        public void onClick(DialogInterface dialog, int which) {
        }       });
dialog.show();
   }
   private boolean isLocationEnabled() {
      locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
      return locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER) ||
            locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
   }
   private boolean requestSinglePermission() {
      Dexter.withActivity(this)
.withPermission(Manifest.permission.ACCESS_FINE_LOCATION)
.withListener(new PermissionListener() {
               @Override
               public void onPermissionGranted(PermissionGrantedResponse response) {
                  isPermission = true;
               }
               @Override
               public void onPermissionDenied(PermissionDeniedResponse response) {
                  if(response.isPermanentlyDenied()){
                     isPermission = false;
                  }                  }
               @Override
               public void onPermissionRationaleShouldBeShown(PermissionRequest permission,
PermissionToken token) {
               }
            }).check();
      return isPermission;
   }
   @Override
   public void onMapReady(GoogleMap googleMap) {
      mMap = googleMap;

      if(latLng!=null){
      mMap.addMarker(new MarkerOptions().position(latLng).title("Current Location"));
      mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng,16));
   }}
   @Override
   public void onConnected(@Nullable Bundle bundle) {
      if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
            PackageManager.PERMISSION_GRANTED &&
            ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)!=
                  PackageManager.PERMISSION_GRANTED){
         return;
      }
startLocationUpdates();
mLocation=LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
if(mLocation == null){
startLocationUpdates();
      }
else{
         Toast.makeText(this,"LocationNotDetected",Toast.LENGTH_SHORT).show();
      }}
   private void startLocationUpdates() {
      mLocationRequest = LocationRequest.create()
.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
.setInterval(UPDATE_INTERVAL)
```

```
.setFastestInterval(FASTEST_INTERVAL);
if(ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
            PackageManager.PERMISSION_GRANTED &&
            ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)!=
                  PackageManager.PERMISSION_GRANTED){
        return;
    }
LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest,this);
}
    @Override
    public void onConnectionSuspended(int i) {
    }
    @Override
    public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
    }
    @Override
    public void onLocationChanged(Location location) {
        String msg = "Update Location: " +
              Double.toString(location.getLatitude())+","+
              Double.toString(location.getLongitude());
        Toast.makeText(this,msg,Toast.LENGTH_SHORT).show();

        LocationHelper helper = new LocationHelper(
location.getLatitude(),
location.getLongitude()
        );

        FirebaseDatabase.getInstance().getReference("")
.setValue(helper).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if(task.isSuccessful()){
                    Toast.makeText(MapsActivity.this, "CALLING EMERGENCY SERVICE",
Toast.LENGTH_SHORT).show();
}else {
                    Toast.makeText(MapsActivity.this, "Location Not saved", Toast.LENGTH_SHORT).show();
 }    }});

        latLng = new LatLng(location.getLatitude(),location.getLongitude());
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
.findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }
    @Override
    protected void onStart() {
super.onStart();
if(mGoogleApiClient !=null){
            mGoogleApiClient.connect();
        }
    }
    @Override
    protected void onStop() {
super.onStop();
if(mGoogleApiClient.isConnected()){
 mGoogleApiClient.disconnect();} }}
```

4.1.3 Working

User Android App Architecture

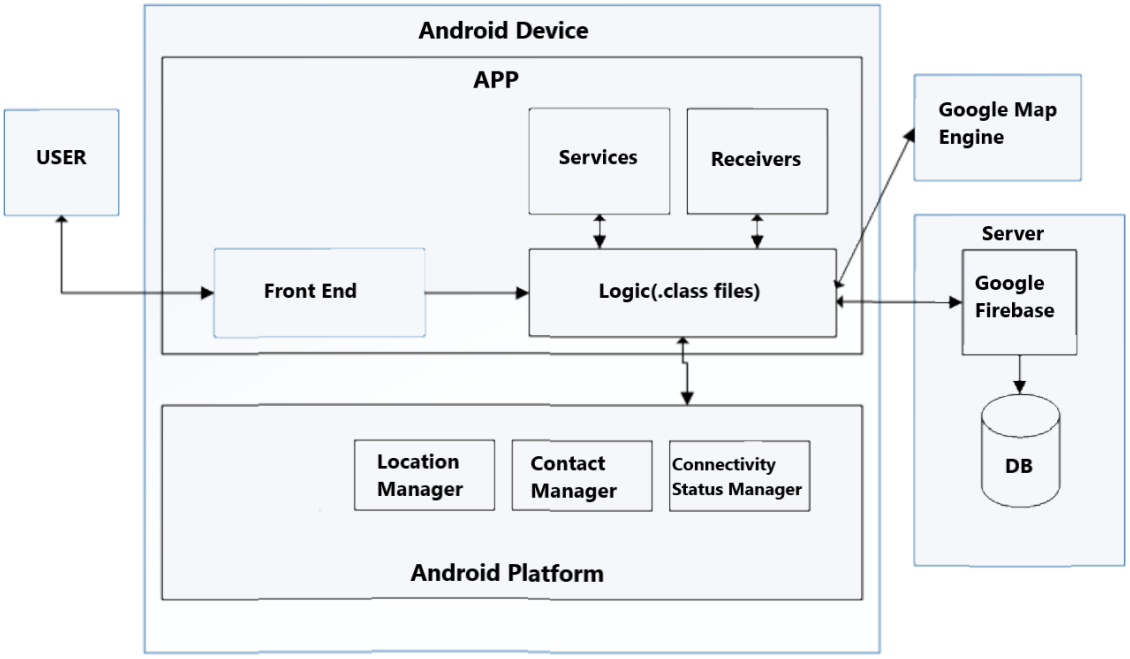


Fig 29: User app architecture

The different components in the user app architecture are –

- User – This is the person who installs the application on his Android device. The user provides various inputs like username, password, and contact numbers etc. and triggers various events on the application.
- Front End – This is the part of the application that is visible to the user. A screen presented to the user is usually an Activity, Fragment or a Dialog Box. They contain various elements like text box or buttons to take inputs from and provide outputs to the user.
- Logic – These are the java files that contain the logic of the application. They contain various methods and classes that meet the functional requirements of the application. These files also contain code to communicate with other components in the application. For example, a file called Map.java will make use of Google maps Android API to connect the Android app with Google Maps Engine to render map and markers of them.
- Services – This is the component of the application that is typically used to perform long background tasks that do not have a user interface. For example – a service is used to track the location of the device at every fixed interval of time.
- Receivers – This is the component of the application that typically listens for some events or responses from other services. For example – A receiver is used to fetch the location co-ordinates from the location service and then add this location to the database for future references.

- Location Manager – It is used to fetch the location of the Android device. The app uses both the GPS provider and the network provider to find the location for the device. GPS provides more accurate data about the location but usually takes sufficient time to start up after the connection is relinquished. Network provider on the other hand is quicker but the accuracy is lesser than GPS.
- Contacts Manager – A system service that provides the contact to use so that the user can select a contact that is already present in the contact book. When the user clicks on the number text box to enter a number it opens up the contact book application. If the user selects a contact and if that contact has a number associated, it is send to the SOS application and displayed in these text boxes.
- Connectivity status manager- This system service tells the application about changes in the connectivity status for the device. The application uses it to make sure that if there is no active internet connection on the device at the time of sending the fetched location to the database.
- Google Maps Engine – The app uses google maps Android API to work with maps. When this API is used, calls are made to the google maps engine to fetch the map or place various markers on it.
- Google Firebase- This is the server where the various information received from the app is fetched and stored.
- Database- This is the Firebase Realtime database on the server. It is used to store the data for the app.

Flowchart of User App

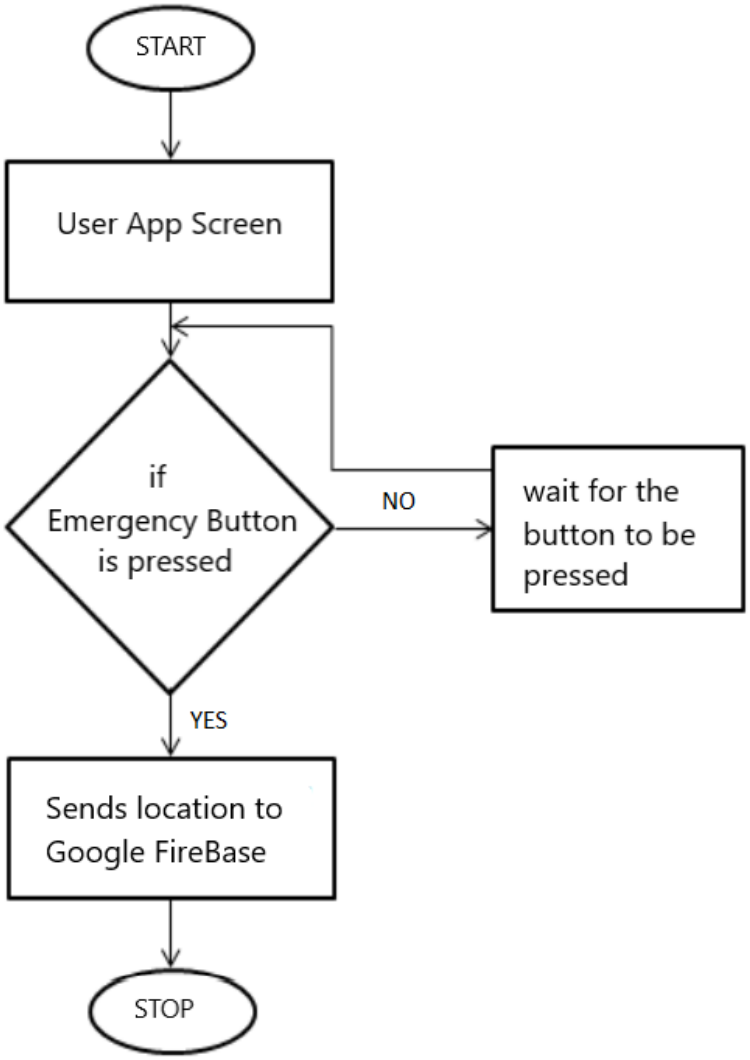


Fig30: Flowchart of user app

Working Steps

1.) When a user will install and open the Emergency App it will show them a screen where there is a button named "Emergency".

2.) The App will check whether the Emergency button is pressed ot not. The function of this Emergency button is to track the location of the device using the App through the GPS available in any phone and will send the location to the Google Firebase.

3.) If the Emergency button is pressed then within no time the app will send the location to the Google Firebase and if the button is not pressed no such function will occur.

User App Interface

Fig 31. User App front page

This is the main screen of the application. Once the user has logged in, every time he opens the User app, this is the first screen presented to him. The screen contains an Emergency Button. The user can call for emergency services by just pressing on the Emergency Button.

As soon as the button is pressed its location is updated in the server of the Google Firebase from where the rescuer app tracks its location and comes to help.

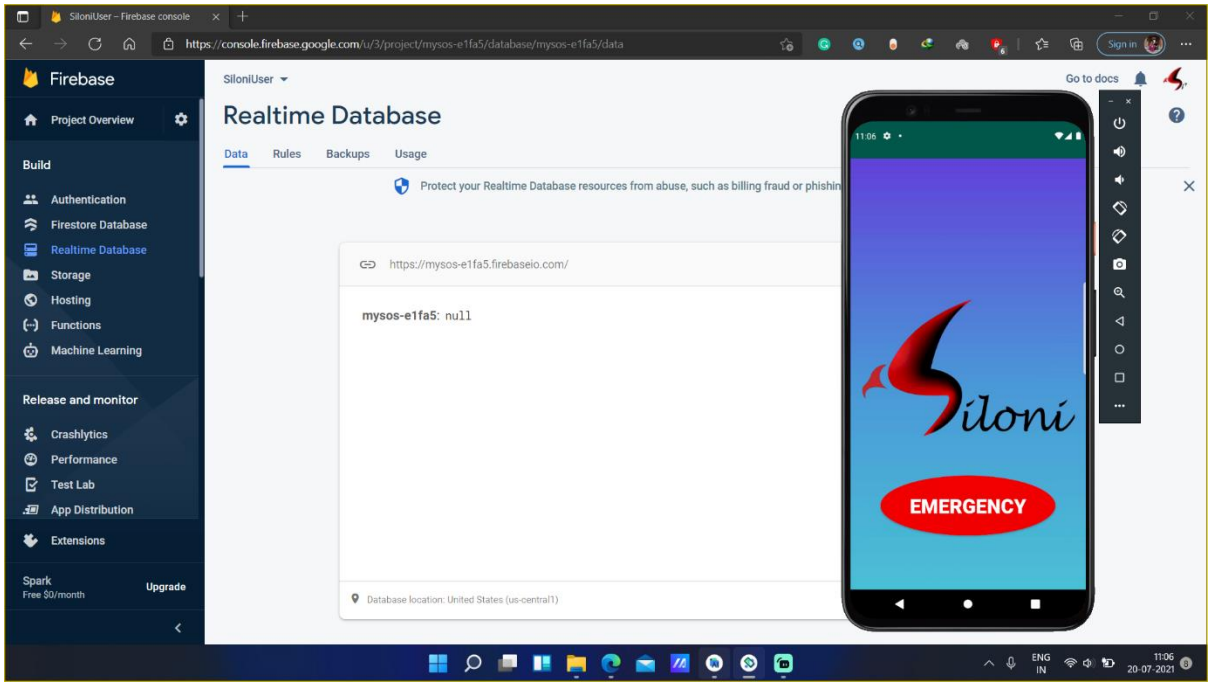


Fig 32. Firebase Database when no emergency button is pressed. [28/05/2021]

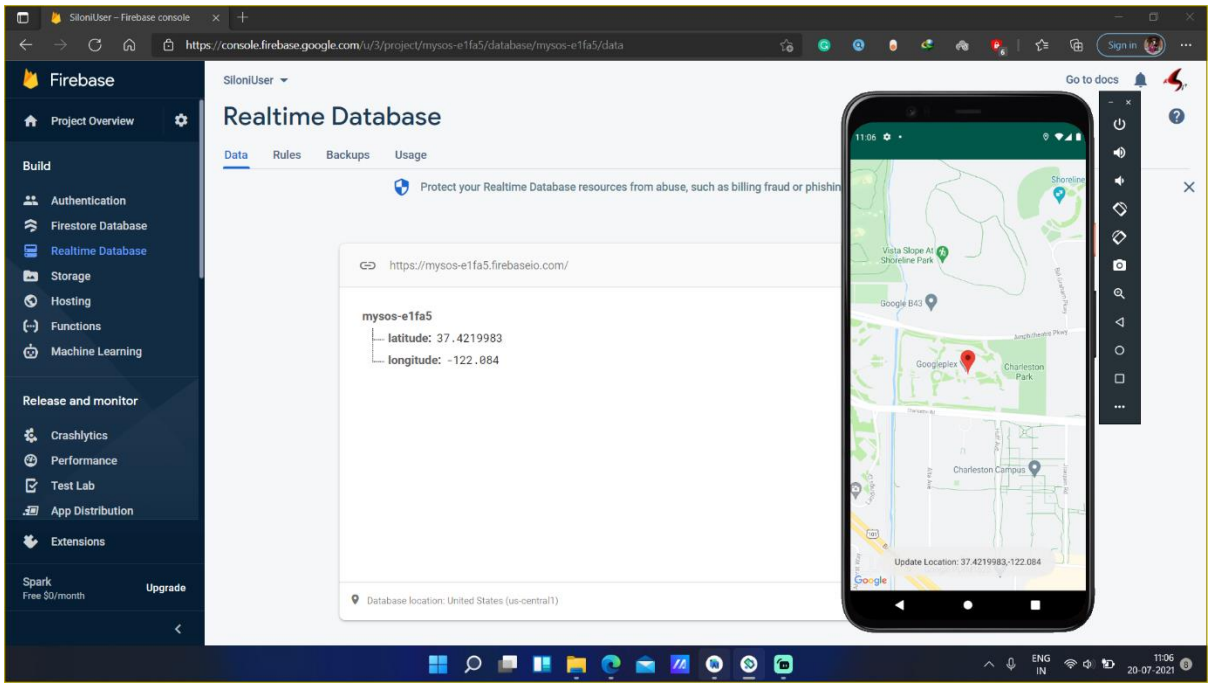


Fig 33. Firebase Database when emergency button is pressed. [28/05/2021]

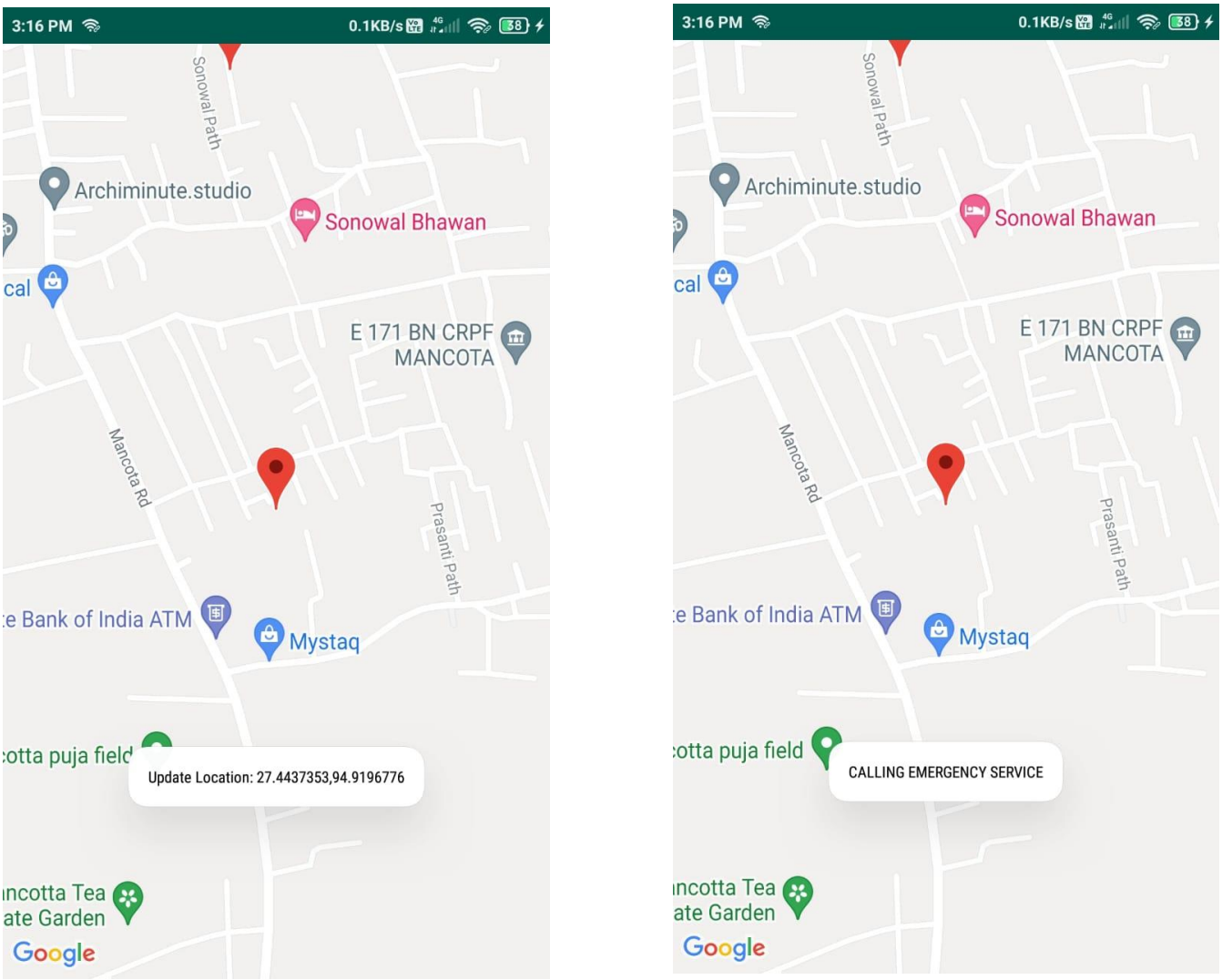


Fig 34: User app calling emergency service [28/05/2021]

The App sends the user's location periodically to the database to avoid change in location over time. It keeps real time monitoring of the User's App until the rescuer team arrives and marks them safe through their App.

4.2 Rescuer Application

Rescuer application is a Google – Map based online emergency application used to track down the emergency locations received from the Firebase. The received location can be from any emergency device installed in the car or from anyone using the Siloni App.

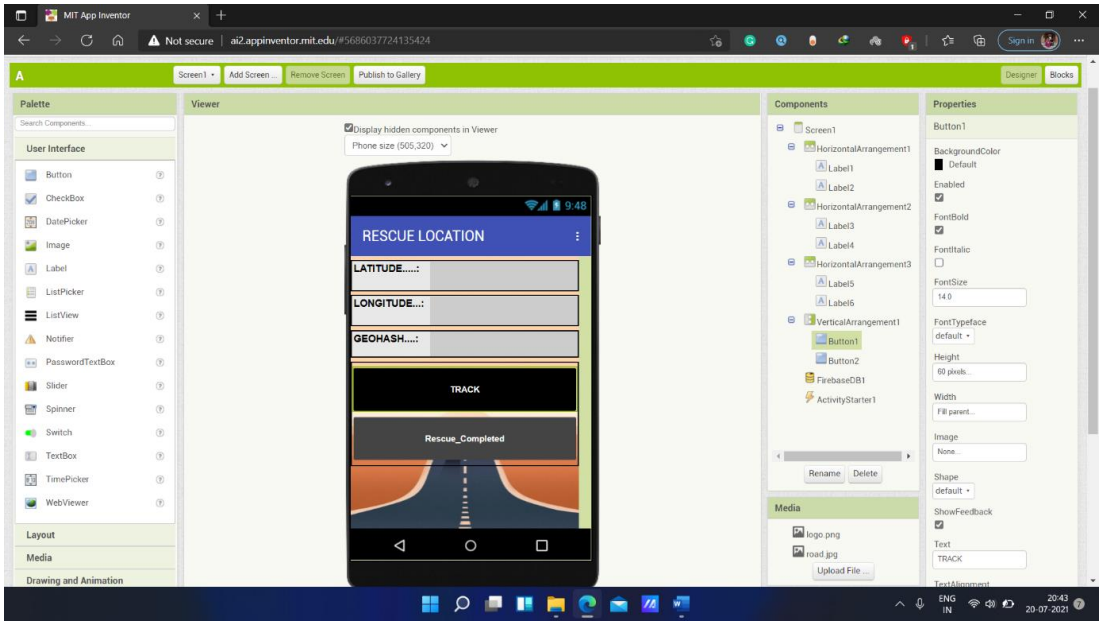
The rescuer application is made by using MIT App Inventor which is a web application integrated development environment originally provided by Google and now maintained by the Massachusetts Institute of Technology

4.2.1 Designing of Application with MIT App Inventor

The MIT App Inventor comprises two parts, i.e. a Component Designer and Blocks Editor. The Components Designer allows the user to drag and drop elements such as text boxes and buttons to create a user interface of their Android application. Meanwhile, the Blocks Editor allows the user to program the components to behave in certain manners by arranging the logic blocks, which are designed similar to puzzle pieces with their own unique tabs and blanks. This enables the users to create a specific or a general program while also preventing most of the possible errors. In addition, due to being aimed for educational purposes, the ease of use of MIT App Inventor also makes it suitable for prototyping process before furthering the developing of Android application to implement more advanced features

Component Designing

Components are core abstractions in MIT App Inventor. Components reduce the complexity of managing interactions with platform-specific application programming interfaces (APIs) and details concerning state management of device hardware. This allows the user to think about the problem at hand rather than the minutia typically required of application developers. Components are made up of three major elements: properties, methods, and events. Properties control the state of the component and are readable and/or writable by the app developer. For example, the enabled property of the location sensor includes the functionality required to configure the GPS receiver and to manage its state while the app is in use. Methods operate on multiple inputs and possibly return a result. Events respond to changes in the device or app state based on external factors.

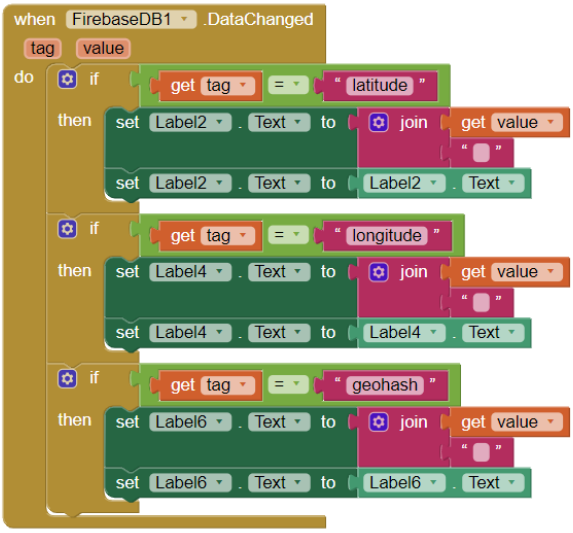


In the App there are three blocks where to show the Latitude, Longitude and the GeoHash that is fetched from the server.

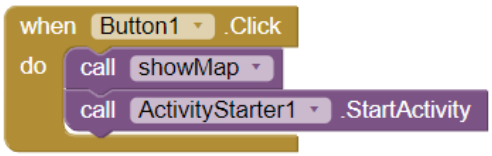
4.2.2Coding and Testing with MIT App

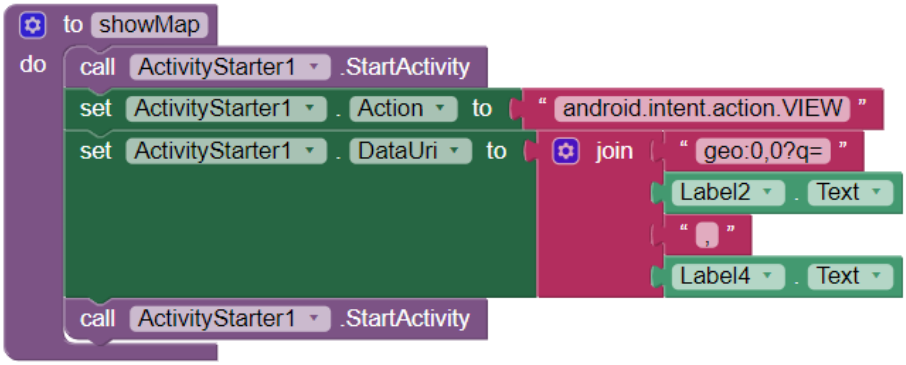
In MIT App Inventor, users code application behavior using a block-based programming language. There are two types of blocks in App Inventor: built-in blocks and component blocks. The built-in blocks library provides the basic atoms and operations generally available in other programming languages, such as Booleans, strings, numbers, lists, mathematical operators, comparison operators, and control flow operators. Developers use component blocks (properties, methods, and events) to respond to system and user events, interact with device hardware, and adjust the visual and behavioral aspects of components.

In our app there are many blocks which have their specific tasks.

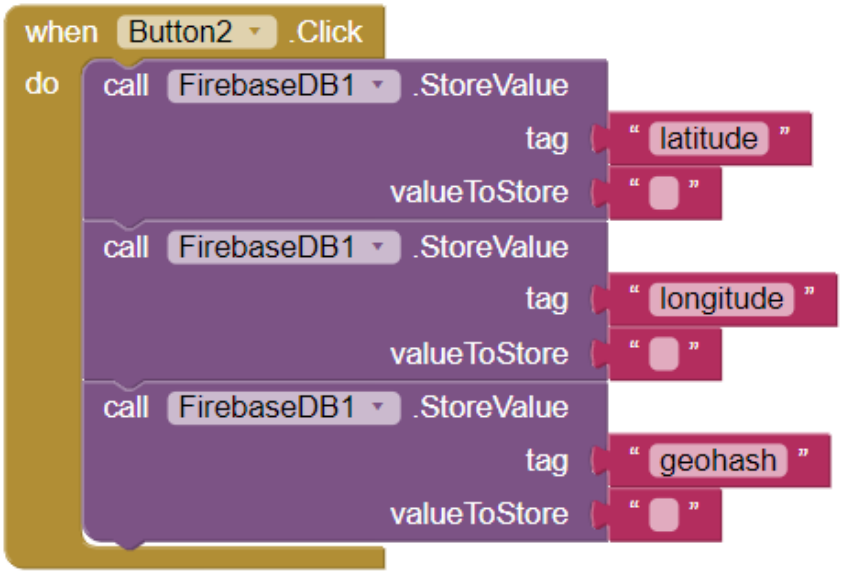


This is the main block of the Application. This part of the application is used to check if any Location is received in the Firebase. Whenever any location is found, the location is shown in the Latitude Longitude and Geohash Labels Respectively.





This Block is for the Track Button of the Application. After a location is received from the Google Firebase and is shown in the Latitude, Longitude and Geohash section of the app, the track button when pressed opens up map services of the smartphone along with the tracked location. With the help of map service available one can receive the route from its location to the tracked location.



This Block is for the Rescue_Completed Button. After pressing this button the location that is stored in the firebase gets erased.

4.2.3 Working

Flowchart

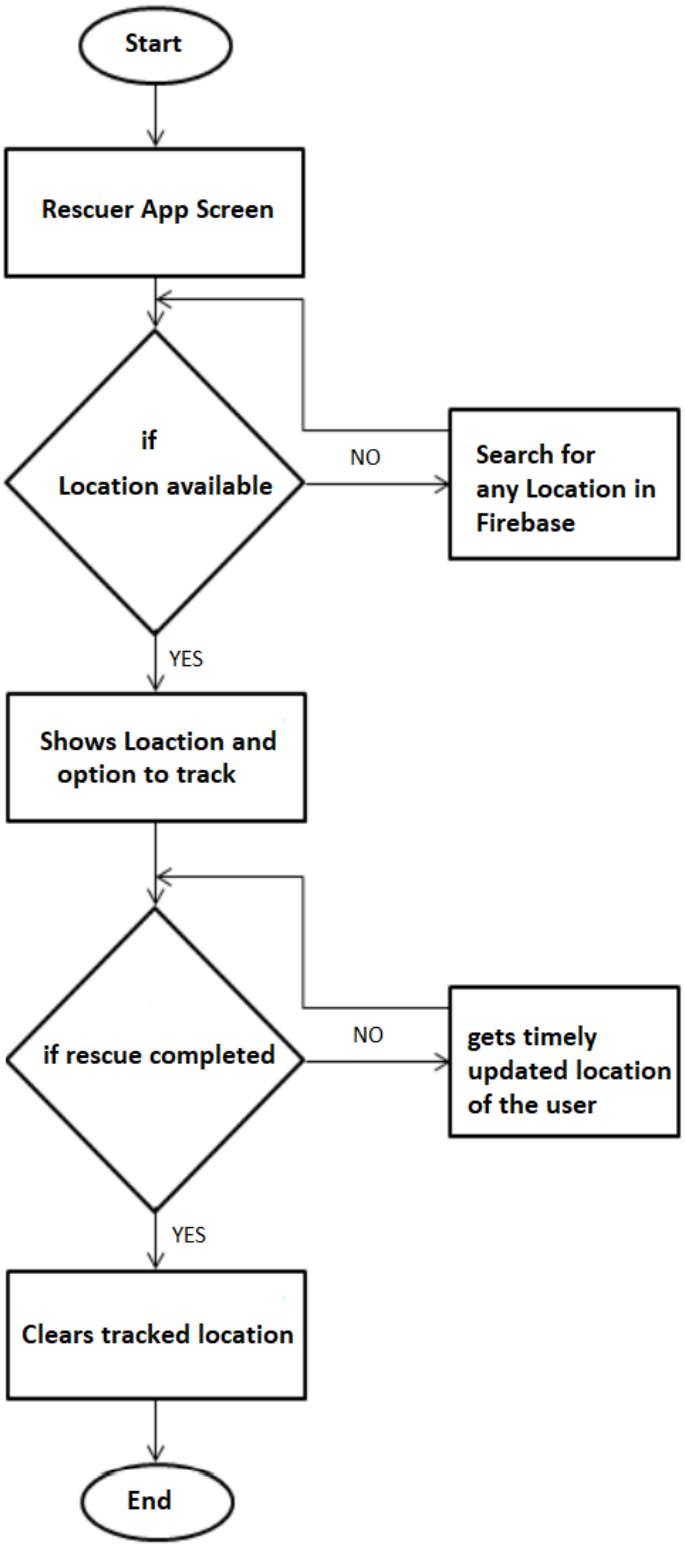


Fig 35 : Flow chart of rescuer app

Working Steps

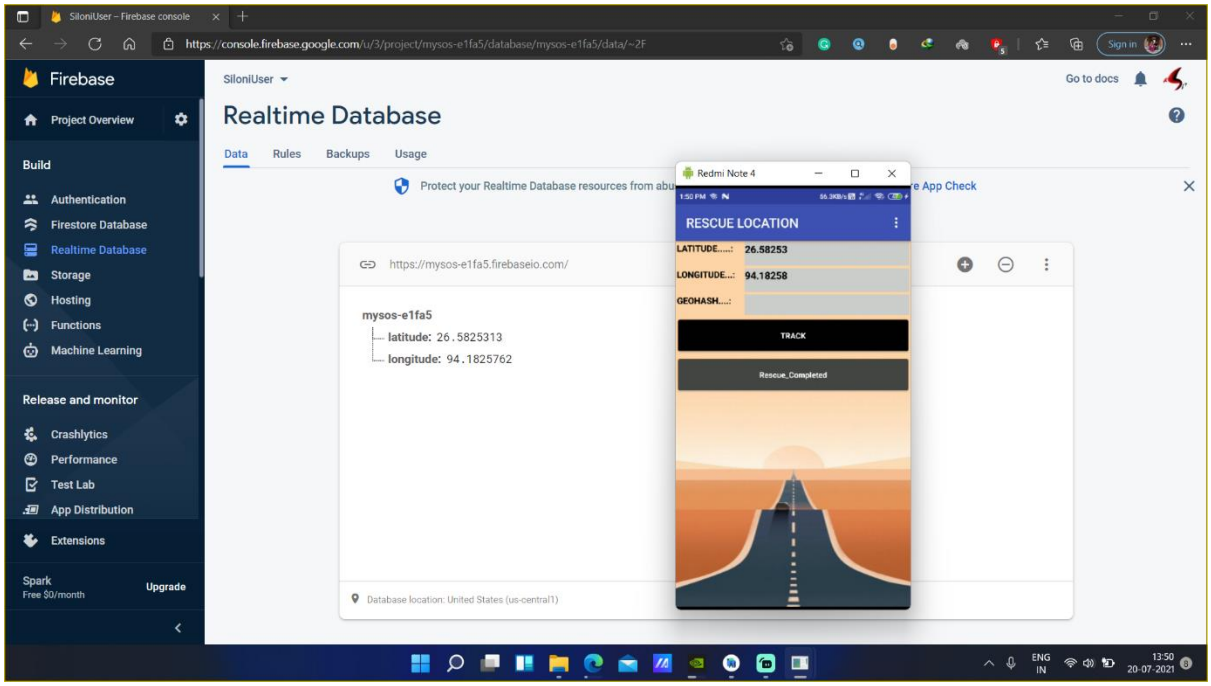
1. When the Rescuer App is installed and opened three sections are displayed named Latitude, Longitude and GeoHash along with two buttons are available at the bottom named Track and Rescue_Completed.

2. If any location is retrieved from the Google Firebase the location is showed in the Latitude, Longitude and GeoHash section of the Application.

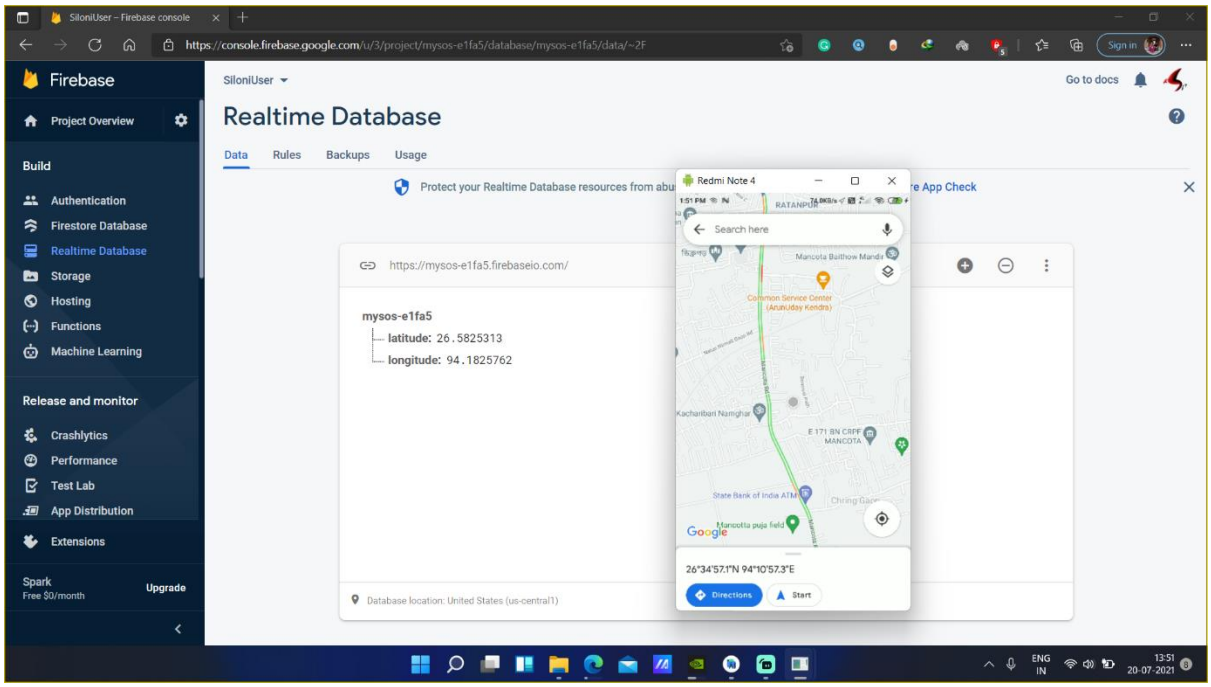
3. The two buttons at the bottom of the App decides what to do with the received location. If the Track bottom is selected the Maps of the phone will open with the location marked in it. And if the Rescue_Completed is pressed, it will clear the location in both the Google Firebase and the Rescuer App.

4. When the Track Button is pressed, the google maps will open and will show the tracked location in it. From this the Rescuer App can get the route from its location to the received location.

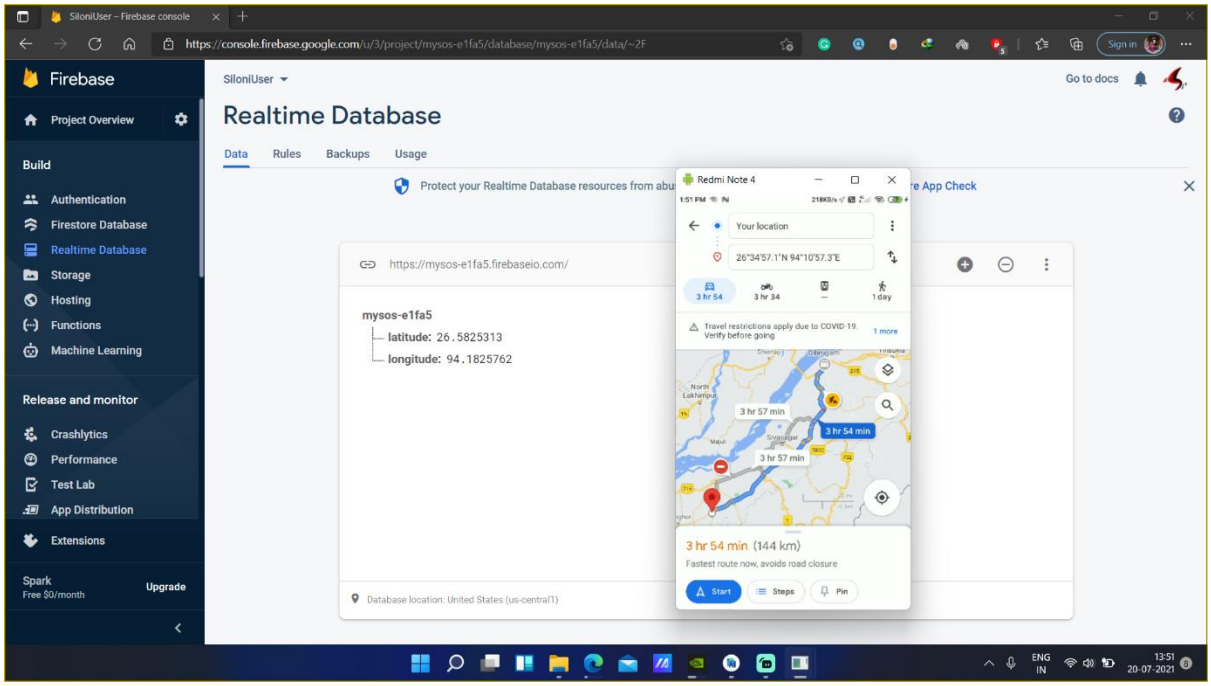
5. After the job is done the rescuer can press the Rescue_Completed Button and it will erase all the earlier received data and the app will go in the first stage.



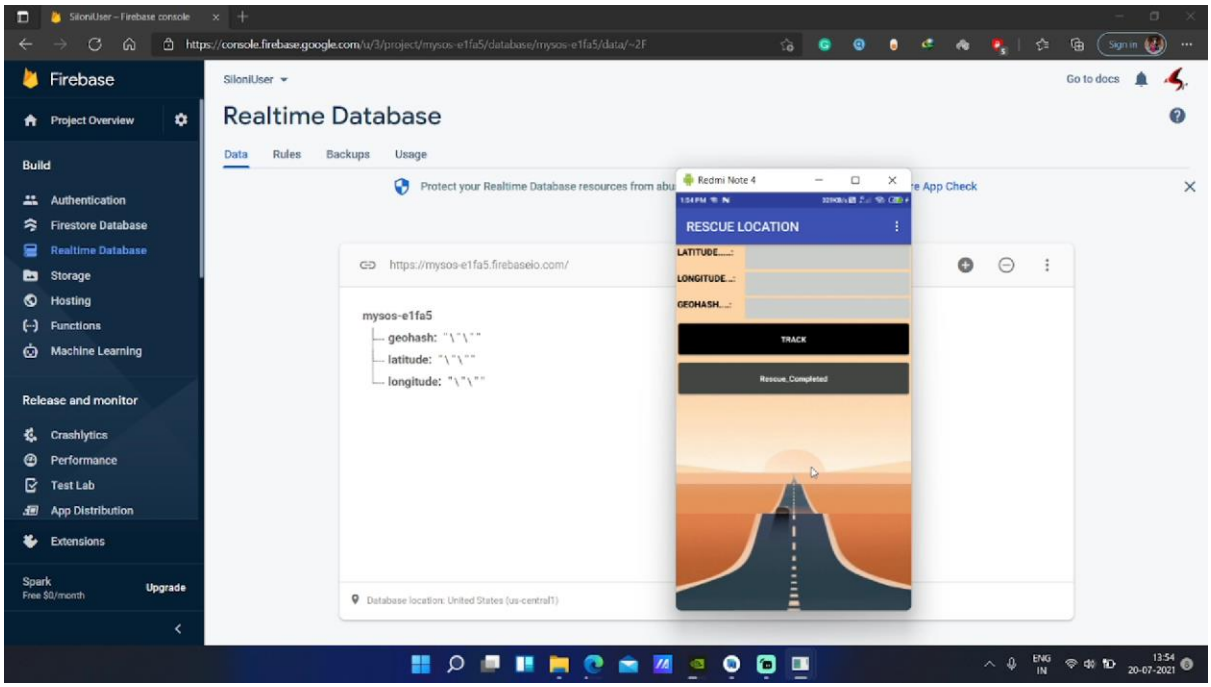
Whenever a location is updated in the Firebase, the Rescuer Application will detect it instantly and show on its interface.



When the Track Button is pressed, the Maps gets opened and checks for the received location.

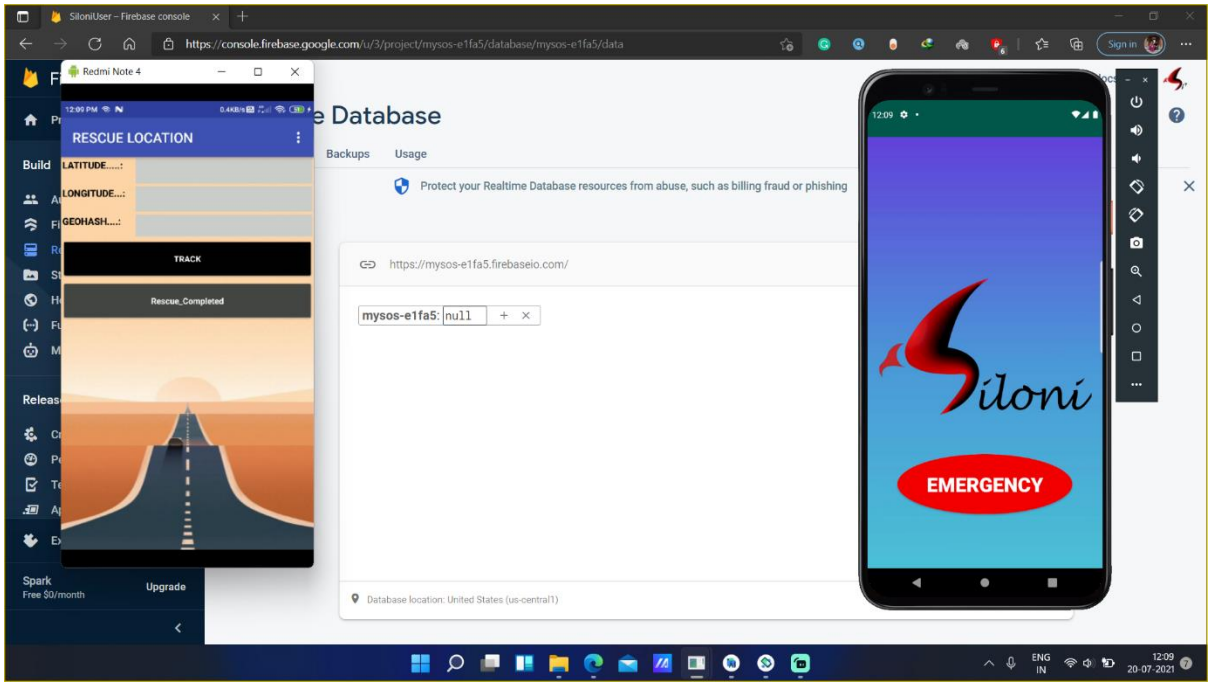


The Rescuer can then find its route to its destination location.

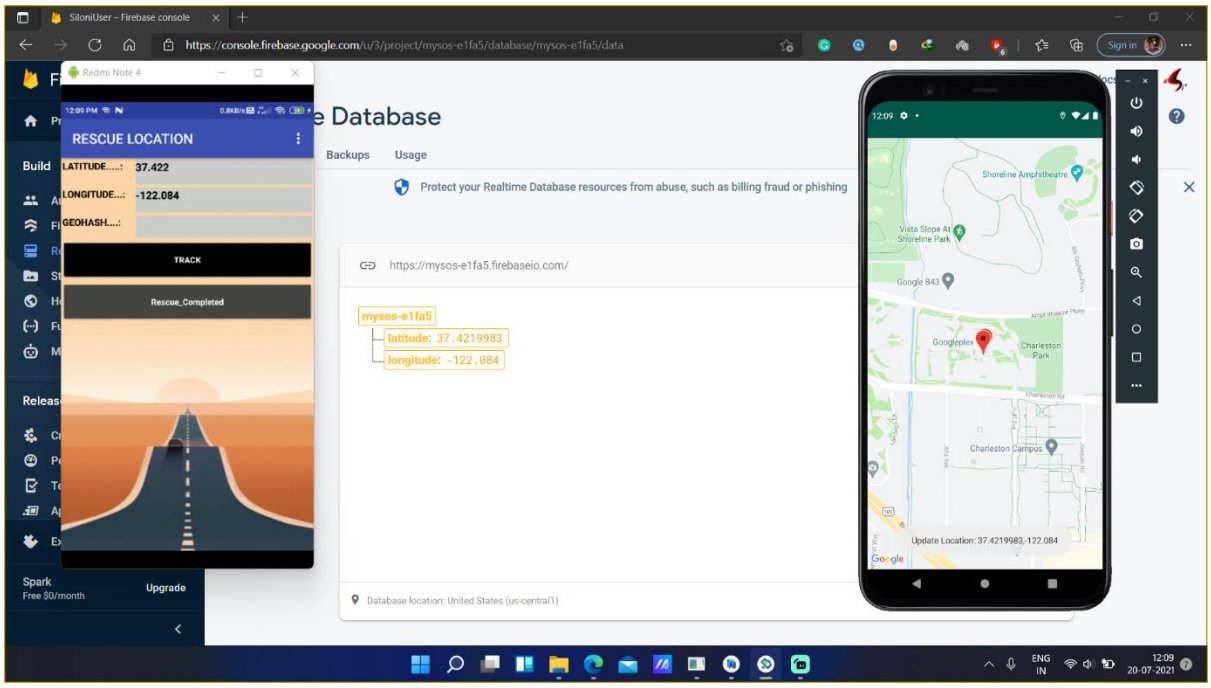


After the Rescuer team rescues the User who called for the emergency, they can click on the Rescue_Complete Button. This Button resets the location stored in both the Firebase and Rescuer Application.

How Both the Applications run together?



When no Emergency Button is pressed in the User Application no location is received in the Firebase nor in the Rescuer App.



When the Emergency Button is pressed, location is received in the Firebase and in the Rescuer App.

CHAPTER 5: Cloud storage and Real-time database

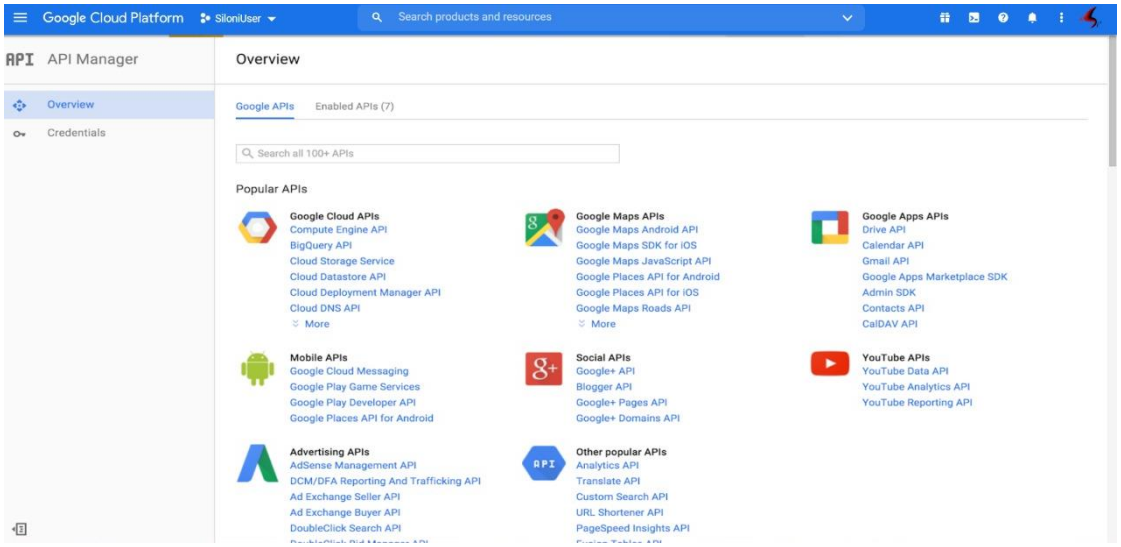
5.1 Firebase

Fig 36. Setting up firebase rule as 'true' to receive and store data. [22/05/2021]

5.2 Google API

Google APIs are application programming interfaces (APIs) developed by Google which allow communication with Google Services and their integration to other services. Examples of these include Search, Gmail, Translate or Google Maps. Third-party apps can use these APIs to take advantage of or extend the functionality of the existing services.

The APIs provide functionality like analytics, machine learning as a service (the Prediction API) or access to user data (when permission to read the data is given). Another important example is an embedded Google map on a website, which can be achieved using the Static Maps API, Places API or Google Earth API.



CHAPTER 6: Working of Hardware

6.1 Tilt Sensing

6.1.1 Arduino NANO with Accelerometer

The signal from the MPU6050 goes to the Arduino Nano where a code is created giving certain condition:

1. When the vehicle is hit from the left side and gets tilted to the right.
2. When the vehicle is hit from the right side and gets tilted to the left.
3. When the vehicle is hit from the front and gets tilted to the back.
4. When the vehicle is hit from the back side and gets tilted to the front.

If any out of these four conditions arises then the led (pin 12) and limit switch (pin 13) will turn HIGH indicating that an accident has taken place. During that time the signal from the MPU6050 goes to Arduino Nano from where it goes to the NodeMCU. In the NodeMCU the led and the limit switch gets high.

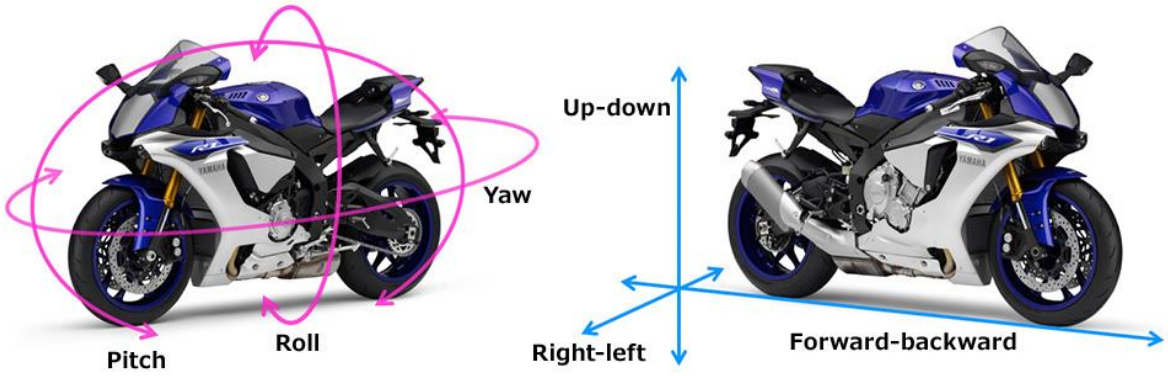


Fig 37. Possible circumstances of tilting or accident [16/06/2021]

In above fig if the vehicle falls in any of angle then the Accelerometer will send high signal to NodeMCU via Arduino NANO

We are giving a high signal to NodeMCU indirectly via Arduino NANO because if we connect MPU6050 directly to NodeMCU then the program shows much delay. Therefore to minimize the delay we are using MPU6050 indirectly.

Circuit diagram of connection of Arduino NANO and MPU6050

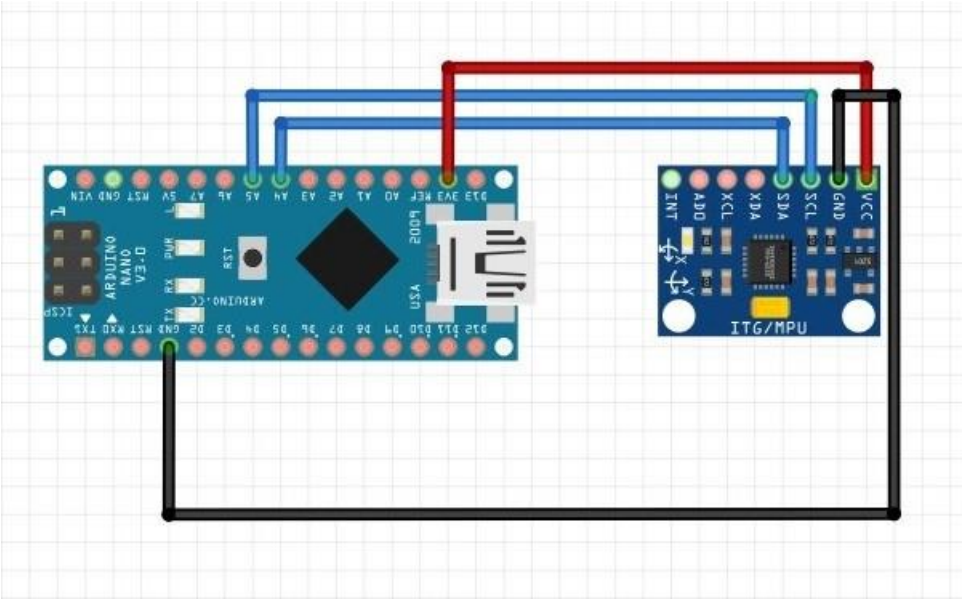


Fig 38. Circuit diagram of connection of Arduino NANO and MPU6050 [28/05/2021]

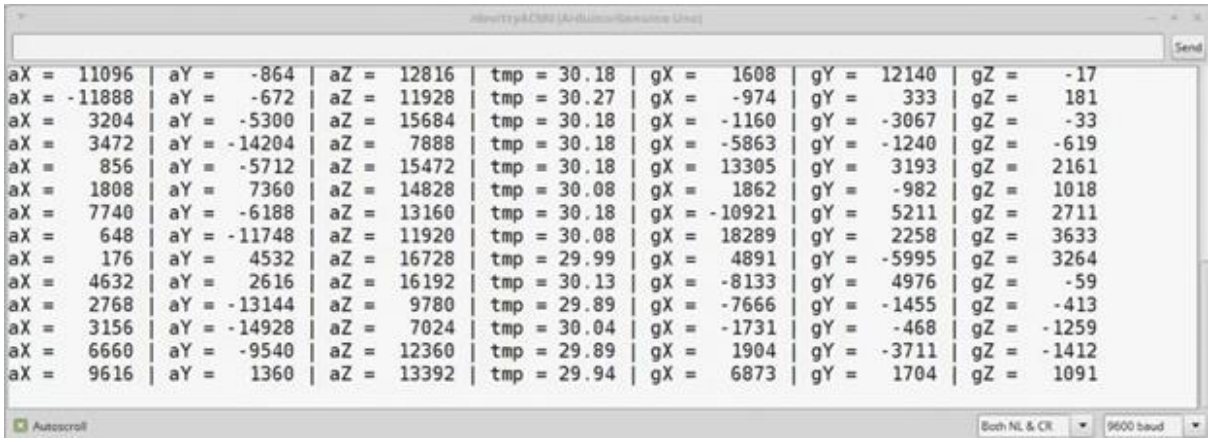
Arduino Nano Code

```
#include "Wire.h" // This library allows you to communicate with I2C devices.

const int MPU_ADDR = 0x68; // I2C address of the MPU-6050. If AD0 pin is set to HIGH, the I2C address will be 0x69

int16_t accelerometer_x, accelerometer_y, accelerometer_z; // variables for accelerometer raw data

int16_t gyro_x, gyro_y, gyro_z; // variables for gyro raw data

int16_t temperature; // variables for temperature data

char tmp_str[7]; // temporary variable used in convert function

char* convert_int16_to_str(int16_t i) { // converts int16 to string. Moreover, resulting strings will have the same length in the debug monitor.

sprintf(tmp_str, "%6d", i);

  return tmp_str;

}

void setup() {

  pinMode(13,OUTPUT);

  pinMode(12,OUTPUT);

  Serial.begin(9600);

  Wire.begin();

  Wire.beginTransmission(MPU_ADDR); // Begins a transmission to the I2C slave (GY-521 board)
```

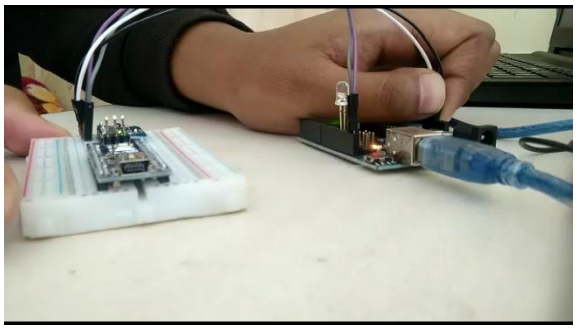
```
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0); // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
}
void loop() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H) [MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2, p.40]
  Wire.endTransmission(false); // the parameter indicates that the Arduino will send a restart. As a result, the connection is kept active.
  Wire.requestFrom(MPU_ADDR, 7*2, true); // request a total of 7*2=14 registers
    // "Wire.read()<<8 | Wire.read();" means two registers are read and stored in the same variable
  accelerometer_x = Wire.read()<<8 | Wire.read(); // reading registers: 0x3B (ACCEL_XOUT_H) and 0x3C (ACCEL_XOUT_L)
  accelerometer_y = Wire.read()<<8 | Wire.read(); // reading registers: 0x3D (ACCEL_YOUT_H) and 0x3E (ACCEL_YOUT_L)
  accelerometer_z = Wire.read()<<8 | Wire.read(); // reading registers: 0x3F (ACCEL_ZOUT_H) and 0x40 (ACCEL_ZOUT_L)
  temperature = Wire.read()<<8 | Wire.read(); // reading registers: 0x41 (TEMP_OUT_H) and 0x42 (TEMP_OUT_L)
  gyro_x = Wire.read()<<8 | Wire.read(); // reading registers: 0x43 (GYRO_XOUT_H) and 0x44 (GYRO_XOUT_L)
  gyro_y = Wire.read()<<8 | Wire.read(); // reading registers: 0x45 (GYRO_YOUT_H) and 0x46 (GYRO_YOUT_L)
  gyro_z = Wire.read()<<8 | Wire.read(); // reading registers: 0x47 (GYRO_ZOUT_H) and 0x48 (GYRO_ZOUT_L)
  // print out data
  Serial.print("aX = "); Serial.print(convert_int16_to_str(accelerometer_x));
  Serial.print(" | aY = "); Serial.print(convert_int16_to_str(accelerometer_y));
  Serial.print(" | aZ = "); Serial.print(convert_int16_to_str(accelerometer_z));
  // the following equation was taken from the documentation [MPU-6000/MPU-6050 Register Map and Description, p.30]
  Serial.print(" | tmp = "); Serial.print(temperature/340.00+36.53);
  Serial.print(" | gX = "); Serial.print(convert_int16_to_str(gyro_x));
  Serial.print(" | gY = "); Serial.print(convert_int16_to_str(gyro_y));
  Serial.print(" | gZ = "); Serial.print(convert_int16_to_str(gyro_z));
  Serial.println();
if(accelerometer_x >12000  )
  {digitalWrite(13,HIGH);
  digitalWrite(12,HIGH);}
```

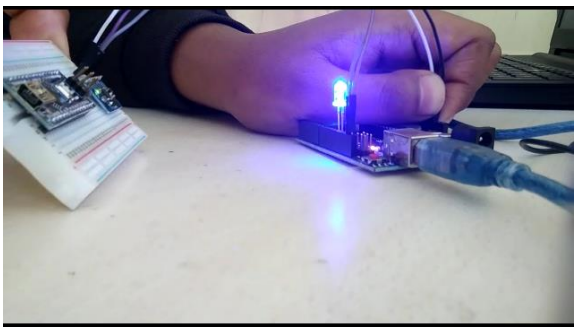
```
 else if(accelerometer_x <-12000  )
 {digitalWrite(13,HIGH);
 digitalWrite(12,HIGH);}
  else if(accelerometer_y >9000  )
 {digitalWrite(13,HIGH);
 digitalWrite(12,HIGH);}
   else if(accelerometer_y <-9000  )
 {digitalWrite(13,HIGH);
 digitalWrite(12,HIGH);}
else
 {digitalWrite(13,LOW);
 digitalWrite(12,LOW);
}
```

Serial Monitor Output



Testing





6.2 Impact Sensing and Sharing GPS coordinate to Firebase

6.2.1 NodeMCU with Impact Sensor

We have used a limit switch as an impact sensor. But in real life scenario a practical impact sensor will the use.

As like practical impact sensor the limit switch replicate the process. It will send a high signal to then NodeMCU.

Limit switch is connected to NodeMCU with GPIO 13.

When limit switch is triggered, it will send a high input to NodeMCU and then NodeMCU will send GPS coordinates to the Firebase.

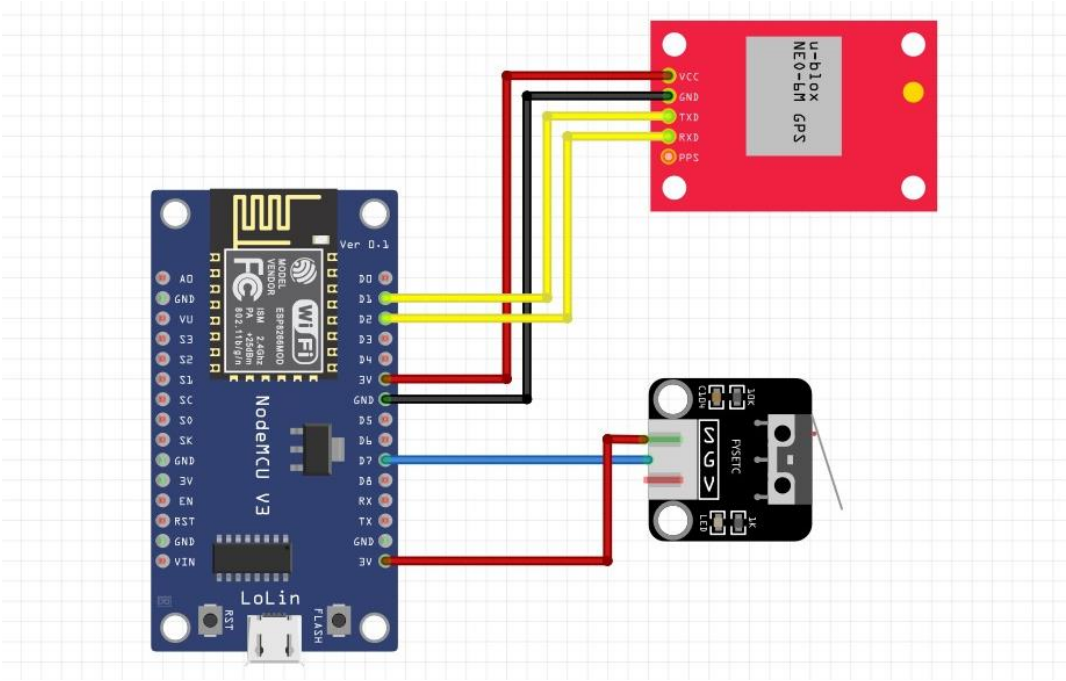


Fig 39. Limit switch connect with NodeMCU [28/05/2021]

6.2.2 Sending GPS coordinate to the database (Firebase)

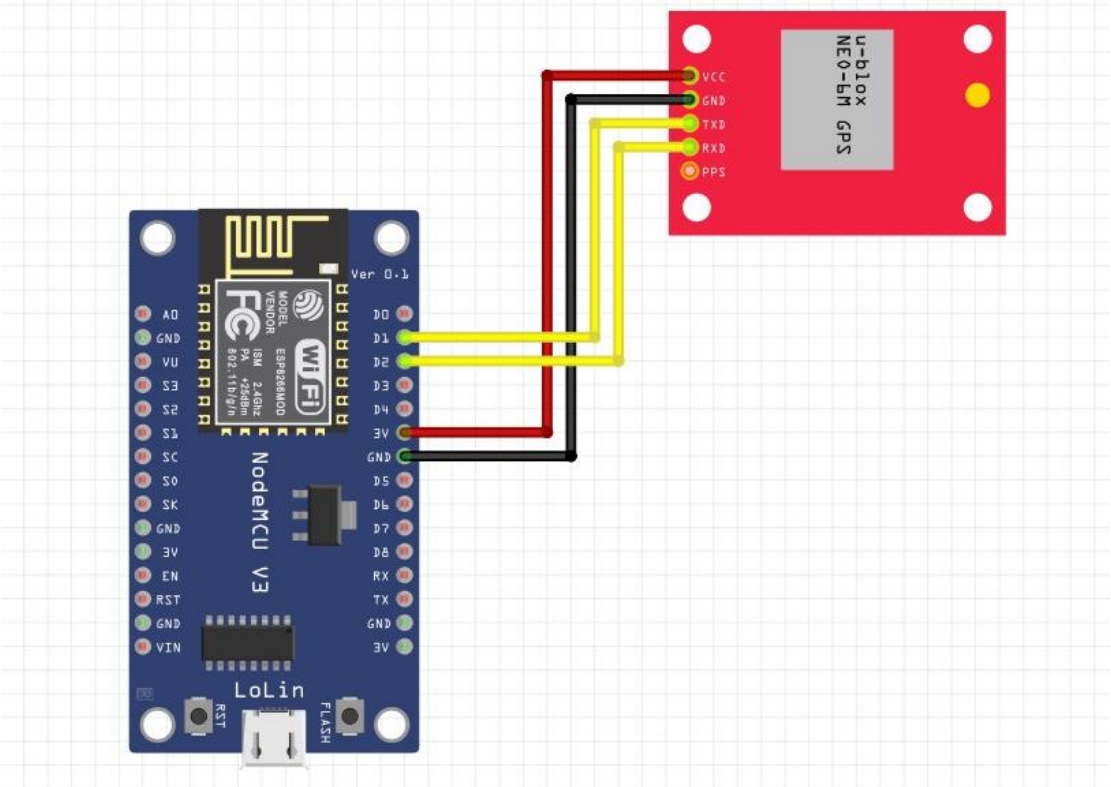
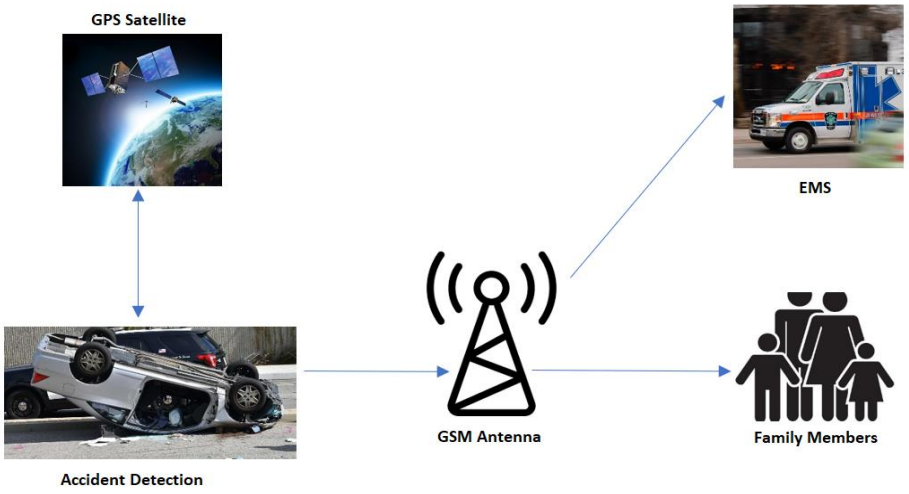


Fig 40. GPS module connected to NodeMCU [28/05/2021]

Once NodeMCU is powered up the GPS module will try to gain GPS coordinates from the satellite. After the GPS module connected to the satellite red led from GPS module will blink rapidly. And the coordinates will be ready to dispatch to the server once the impact sensor and the accelerometer will give high input that means accident has been occur. For better understanding a flow chart is provided below.

System Architecture



Block Diagram

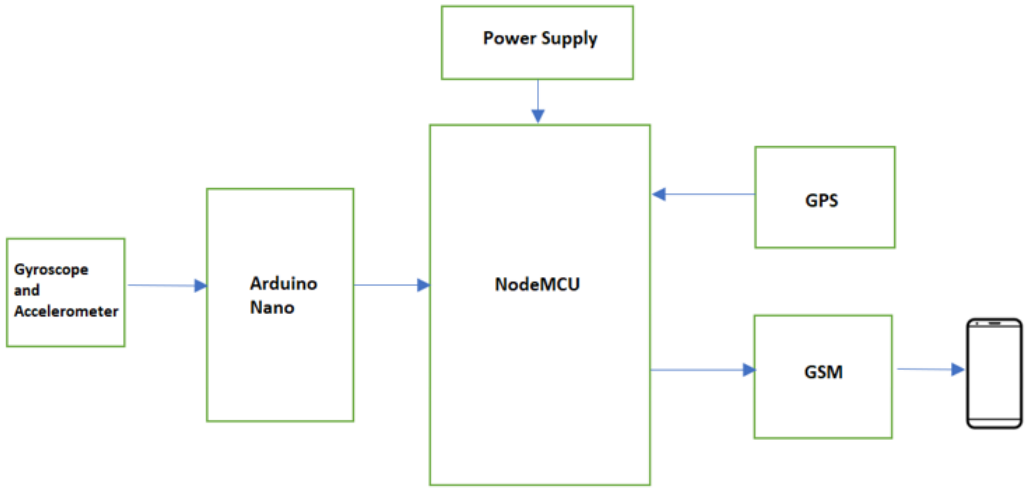
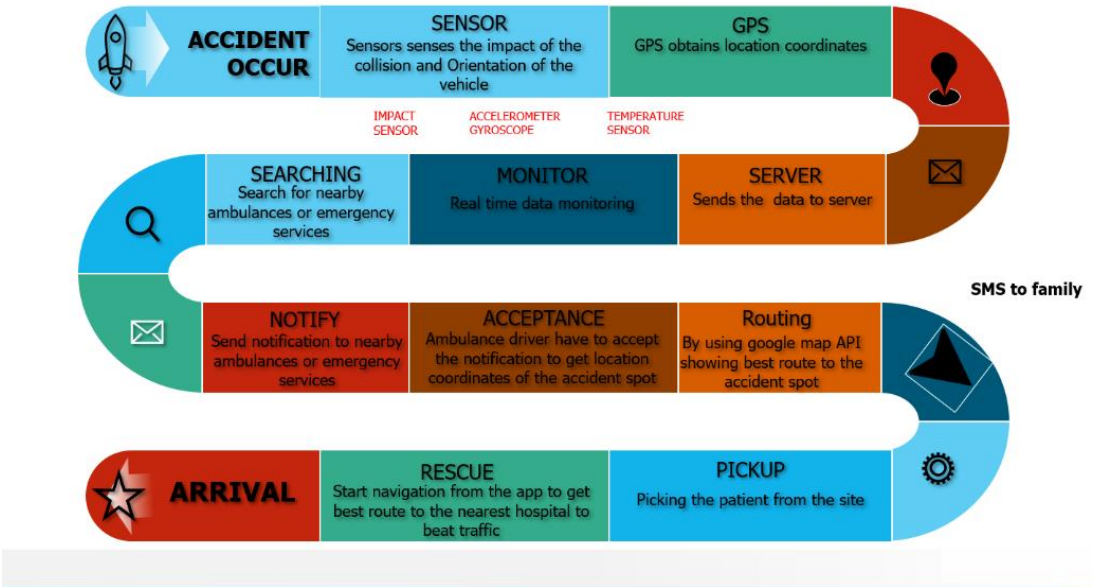
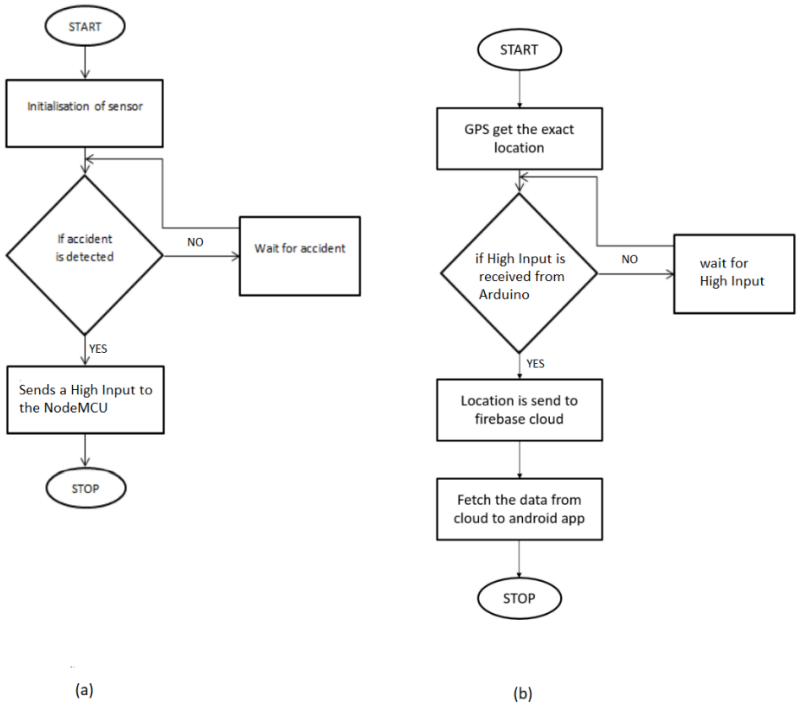


Fig 41. Block Diagram of the Project

Working Process:



Flowchart



Flowchart (a) represents the working of the Arduino Nano of how it detects when an accident occurs.

Flowchart (b) represents the working of the NodeMCU of how it tracks the location of the user and sends it to the Google Firebase.

Circuit Diagram

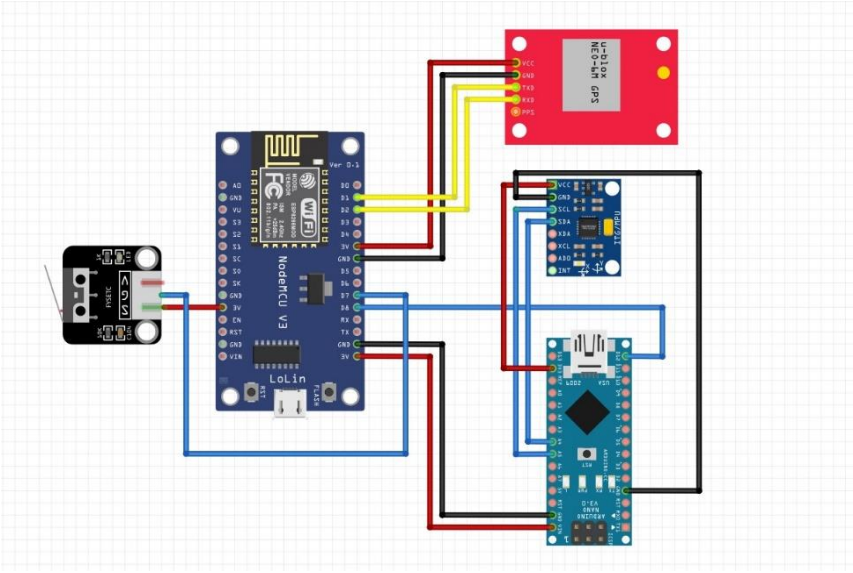


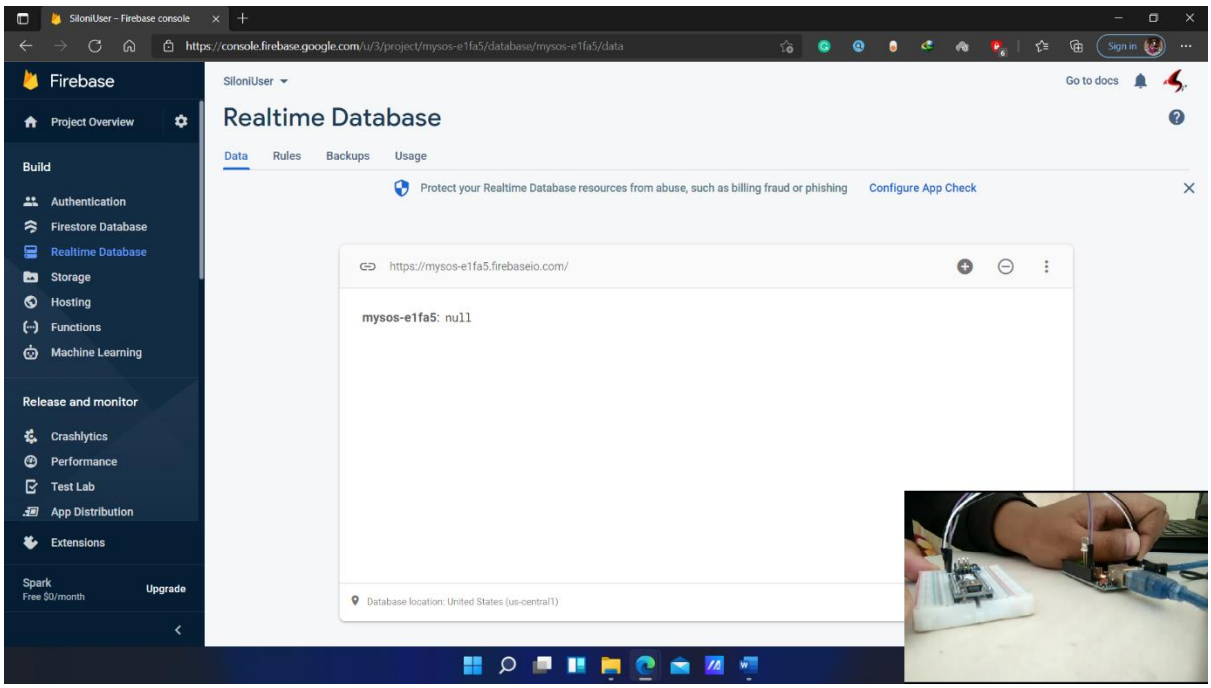
Fig 42 : Final circuit diagram [28/05/2021]

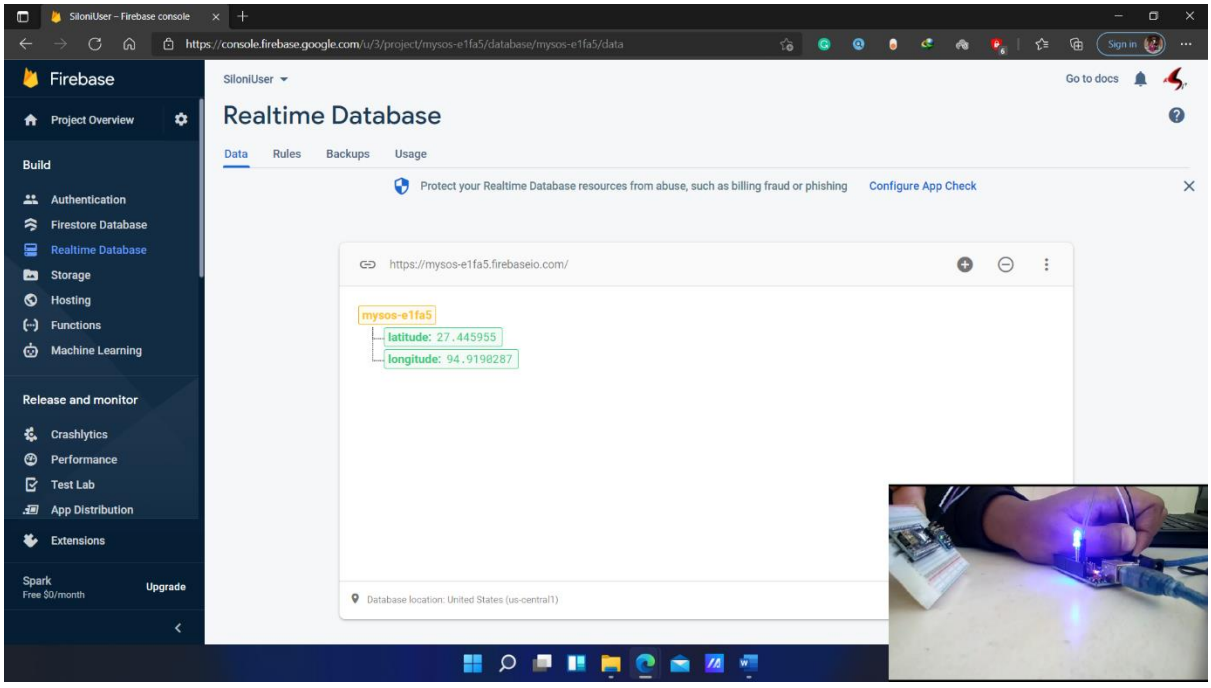
FINAL HARDWARE CODE

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

#include <FirebaseArduino.h>

#include <arduinoGeohash.h>

GeoHash hasher(15);

TinyGPSPlus gps;

SoftwareSerial ss(16, 5);

#define FIREBASE_HOST "mysos-e1fa5.firebaseio.com"

#define FIREBASE_AUTH "Bgg5BXzB5cJmuRQ4fhOsdQYekxT1vnBmDuGOqKcW"

#define WIFI_SSID "chitta"

#define WIFI_PASSWORD "7002609097"

float latitude;

float longitude;

String lat_str , lng_str;

const String UPDATE_PATH = "gps_device" ;

int limitswitch = 13;

int acce = 15;

void setup() {

  Serial.begin(9600);

ss.begin(9600);

  pinMode(limitswitch,INPUT);

  pinMode(acce,INPUT);

  Serial.println();

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.print("connecting");

  while (WiFi..status() != WL__CONNECTED)

  {

    Serial.print(".");

delay(500);

  }

  Serial.println();

  Serial.print("connected: ");

  Serial.println(WiFi.localIP());
```

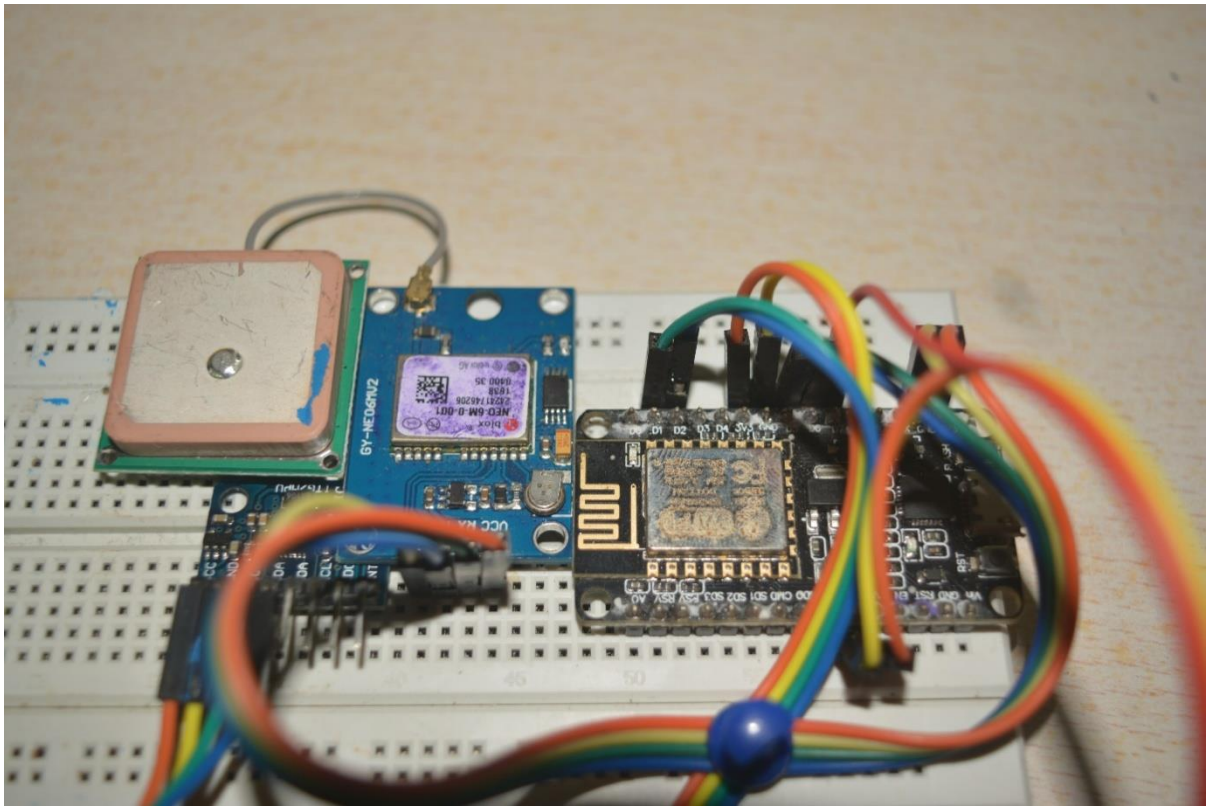
```
 Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}

void loop() {

 while(ss.available() > 0) {gps.encode(ss.read());}

   if (gps.location.isUpdated())
   {
     latitude = gps.location.lat();
     longitude = gps.location.lng();
     String lat_str = String(latitude, 15);
     String lng_str = String(longitude, 15);
   const char* geohash = hasherencode(latitude, longitude);

  if (digitalRead(limitswitch) == HIGH){
     Firebase.setString("geohash", geohash);
     Firebase.setString("latitude", lat_str);
     Firebase.setString("longitude", lng_str);

     Serial.print("GEOHASH=");Serial.println(geohash);
     Serial.print("LATITUTE="); Serial.println(latitude, 15);
     Serial.print("LONGITUDE="); Serial.println(longitude, 15)
  }
  else if (digitalRead(acce) == HIGH){
     Firebase.setString("geohash", geohash);
     Firebase.setString("latitude", lat_str);
     Firebase.setString("longitude", lng_str);

     Serial.print("GEOHASH=");Serial.println(geohash);
     Serial.print("LATITUTE="); Serial.println(latitude, 15);
     Serial.print("LONGITUDE="); Serial.println(longitude, 15);
  }   }}
```

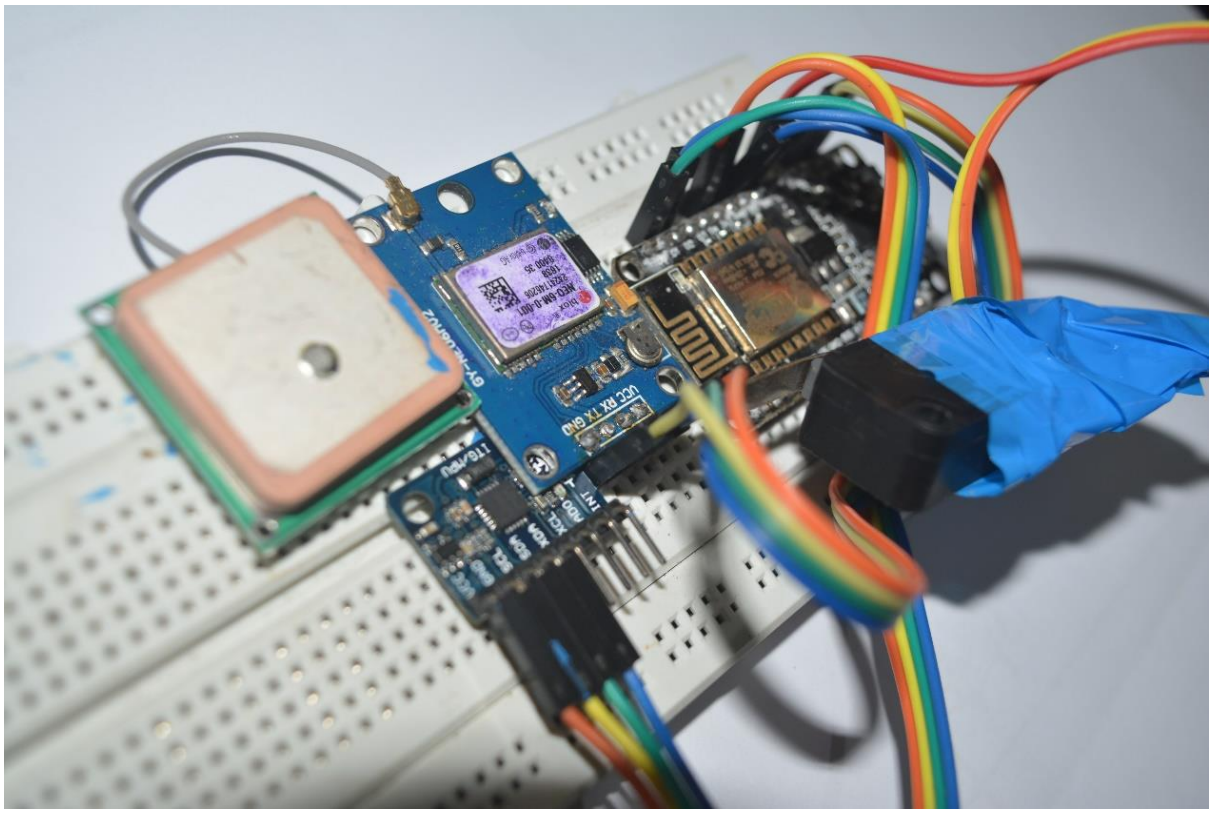
Firebase Output:

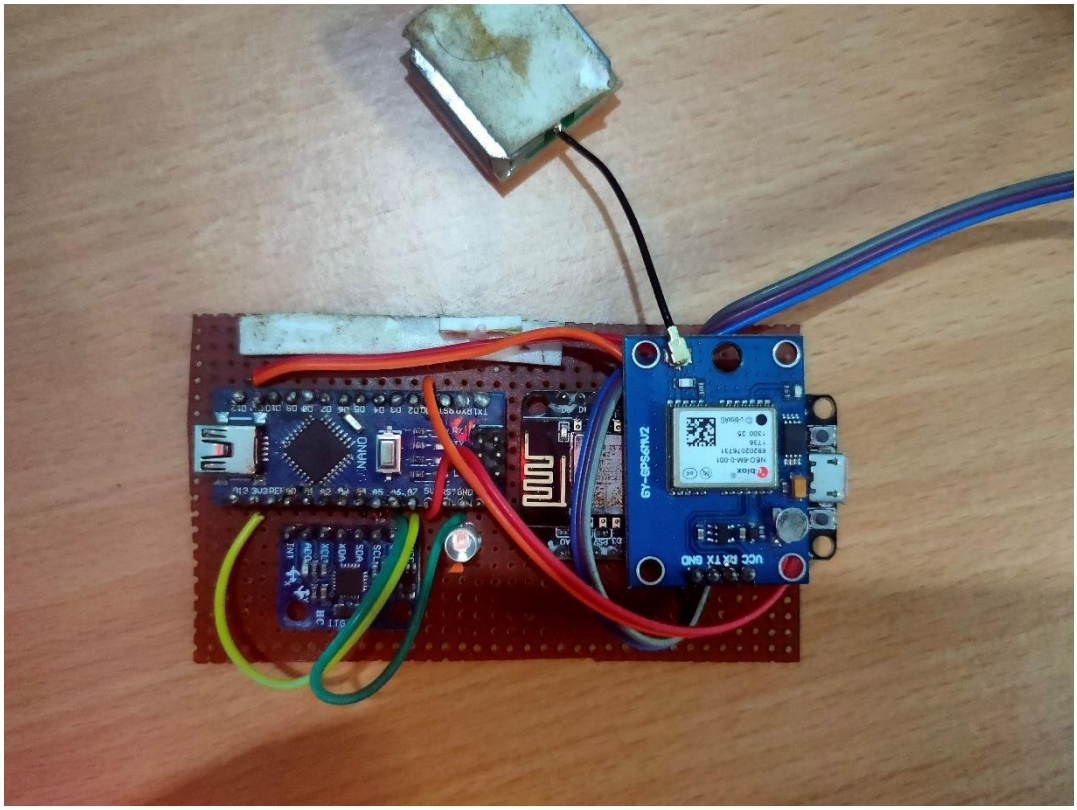


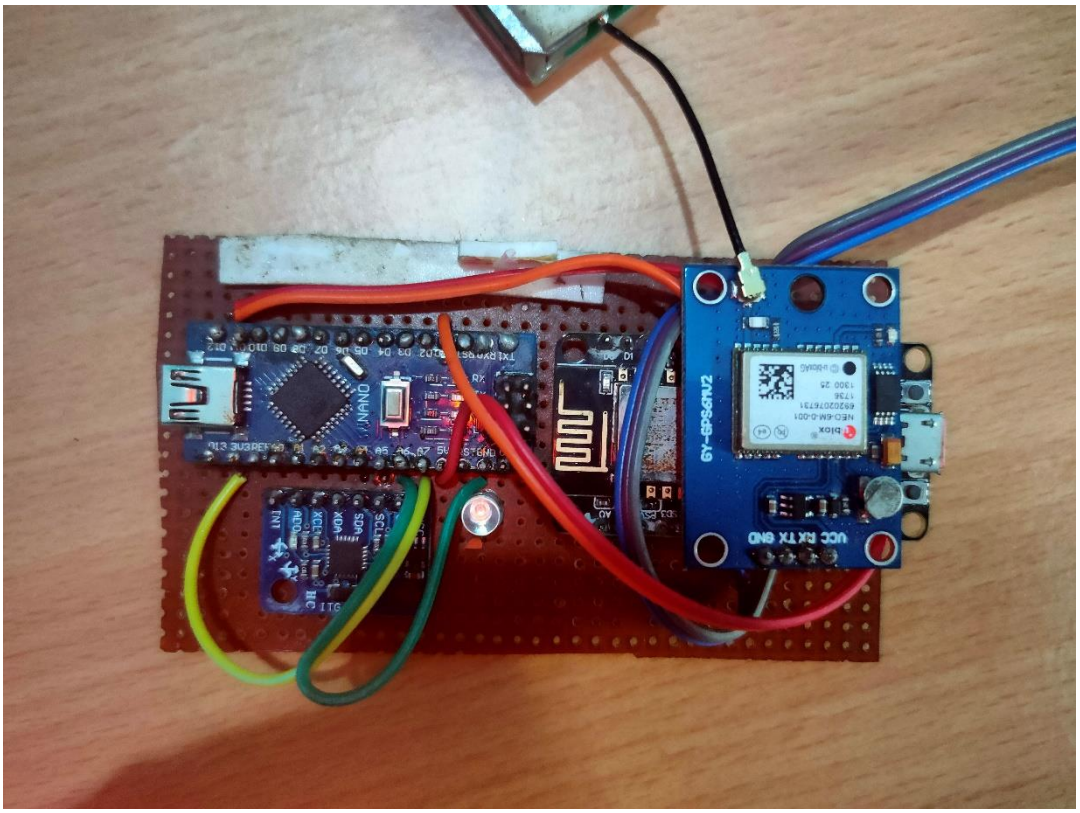


Chapter 7: Glimpses of the whole Project






GY-GPS6MV2
VCC RX TX GND


GY-GPS6MV2
VCC RX TX GND

Chapter 8: Conclusion

As far as mortality rate is considered, a major percent of deaths occur as a result of accidents. In this project the overall system to avoid collision at the initial stage has been developed. Moreover, if unavoidable accidents occur, the automatic Smart Rescue system and Smart Lifesaver equipment will ensure that the lives of victims are saved to a greater extent.

Internet of Things-enabled Intelligent Transportation Systems (ITS) is gaining significant attention in academic literature and industry, and is seen as a solution to enhancing road safety. The current study proposes and implements a smart accident detection and rescue system for any certain part of the world. The proposed approach aims to take advantage of advanced specifications of smart phones to design and develop a low-cost solution for enhanced transportation systems that is deployable in legacy vehicles. In this context, a customized android application is developed to gather information regarding speed, gravitational force, temperature, and location. The information acquired is further processed to detect road incidents. This system ensures smart accident detection and rescue mechanism that can provide the exact location of the accident. It also aids any ambulance to get direction. Ambulance driver can take the shortest path to the accident spot and thereby rescue the victims in the least possible timeframe. Besides, our system puts a minimum of financial strain on the user to implement than many other systems, but with a more reliable output. In future machine learning techniques can be implemented to get even better insights into the situation in an automated manner.

REFERENCE

1. arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardNano, 16/06/2021
2. arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardNano, 16/06/2021
3. theengineeringprojects.com/2018/06/introduction-to-arduino-nano.html, 16/06/2021
4. etechnophiles.com/arduino-nano-every-pinout-specifications-schematic-datasheet/, 16/06/2021
5. A review of use of NodeMCU ESP8266 in IoT products, Yogendra Singh Parihar, NATIONAL INFORMATICS CENTRE, INDIA, 23/04/2021
6. lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/, 07/04/2021
7. INVESTIGATION OF WI-FI (ESP8266) MODULE AND APPLICATION TO AN AUDIO SIGNAL TRANSMISSION, TATAP PERETS ARNAUD, 09/04/2021
8. Comparative study between two Powerful NodeMCU Circuits: ESP32 and ESP8266, Mohamed Fezari, Badji Mokhtar-Annaba University, 22/05/2021
9. lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/, 16/06/2021
10. robu.in/product/ublox-neo-6m-gps-module/, 17/06/2021
11. projecthub/ruchir1674/how-to-interface-gps-module-neo-6m-with-arduino-8f90ad, 17/06/2021
12. NEO-6_DataSheet_(GPS.G6-HW-09005).pdf, 17/06/2021
13. Papers/AAAI/2005/IAAI05-013.pdf, 16/06/2021
14. elprocus.com/accelerometer-sensor-working-and-applications/, 16/06/2021
15. vedantu.com/physics/accelerometer, 16/06/2021
16. ncbi.nlm.nih.gov/pmc/articles/PMC5677445/, 16/06/2021
17. watelectronics.com/what-is-a-gyroscope-sensor-working-its-applications/, 16/06/2021
18. netdna-ssl.com/wp-content/uploads/2014/12/mpu-6000-family-diagram.png, 16/06/2021
19. invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/], 17/06/2021
20. electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module, 17/06/2021
21. indiamart.com/proddetail/3d-printer-mechanical-limit-switch-module-11032705133.html, 17/06/2021
22. instructables.com/How-to-use-a-breadboard/, 16/06/2021
23. instructables.com/How-to-use-a-breadboard/, 16/06/2021
24. blog.sparkfuneducation.com/what-is-jumper-wire, 19/06/2021
25. upload.wikimedia.org/wikipedia/commons/thumb/5/5c/A_few_Jumper_Wires.jpg/1280px-A_few_Jumper_Wires.jpg, 19/06/2021
26. sciencedirect.com/topics/engineering/printed-circuit-boards, 19/06/2021
27. twi-global.com/technical-knowledge/faqs/what-is-soldering, 16/06/2021
28. digikey.in/maker-media/35f5a67a-c375-4a39-a2ed-613740878835, 16/06/2021
29. arduino.cc/en/Guide/Environment, 19/06/2021
30. developer.android.com/studio/intro, 19/06/2021
31. developer.android.com/studio/intro, 19/06/2021