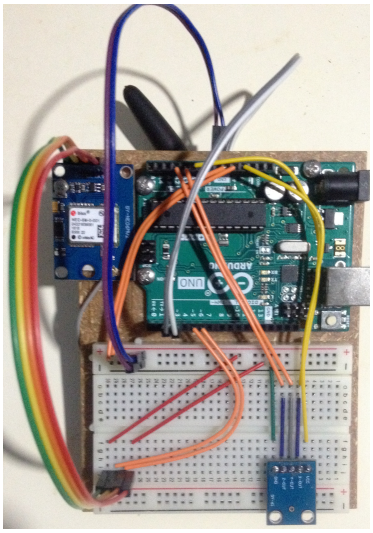
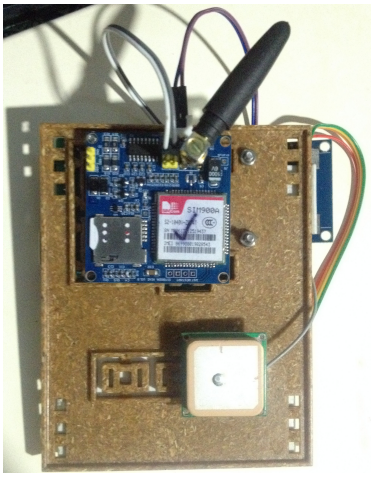
Oculus Aquila

Application of Interactive Pervasive Computing
for bike-accident alert and geo-tracking.



(a) Finished Prototype (Front).



(b) Finished Prototype (Back)

Biju Ale
SID: 9638350
15 March, 2019

A STW307CR individual project documented for the degree of
BSc. Computing (Hons.) in Computing

in collaboration with

Softwarica College of IT & E-commerce | Coventry University

Faculty of Engineering, Environment and Computing
Coventry University

Contents

List of Figures

Chapter 1

Introduction

1.1 Project Motivation

As an avid bike rider myself, who commutes 20km daily on the bustling streets of Kathmandu, security is of prime concern to me. Security in this context is bicentric, i.e. security as in the rider's safety during commute and security of the bike itself.

Chances of recovery are slim once a bike is stolen in Kathmandu. Unlike motor vehicles, bikes usually do not have registration IDs associated with it, making it difficult to track.

Likewise, tracking could also help responders attend an accident scenario in a bike trail or on regular rides.

The aforementioned problem is a well-defined one and could be addressed with the application of interactive pervasive computing.

1.2 Problem Statement

- Send an alert with GPS coordinates of the accident site during an accident.
- Track the bike and continuously send the GPS coordinates of the bike upon receiving track command remotely.

1.3 Aim

To develop a low-profile, low-powered, & a high-coverage IOT system that can be installed on a bike which can send GPS coordinates during an accident or upon remote request during a theft.

1.4 Objectives

1.4.1 Technical

- To define the scope of the problem to be solved.
- To identify the components required to construct the proposed solution.

- To model the structural aspect of the system.
- To model the dynamic aspect of the system.
- To design the sub-routines to be executed in the Arduino.
- To unit test each routine and to integrate them.
- To make use of IOT web services for dashboard.
- To document the project in report and video.

1.4.2 Academic

- To create a simple user-centric interactive pervasive computing application.
- To understand if a problem is solvable using pervasive computing and to come up with a prototype solution.
- To utilize IOT web services and web APIs and integrate them into the project for notification and/or data persistence.

1.4.3 Personal

- To develop hands-on skills in DIY-electronics project.
- To build a portfolio of IOT project work.
- To understand how GPS, accelerometer, and GSM units work.
- To understand the analysis, design, and implementation phases of IOT systems.

Chapter 2

Justification

2.1 Rich Picture

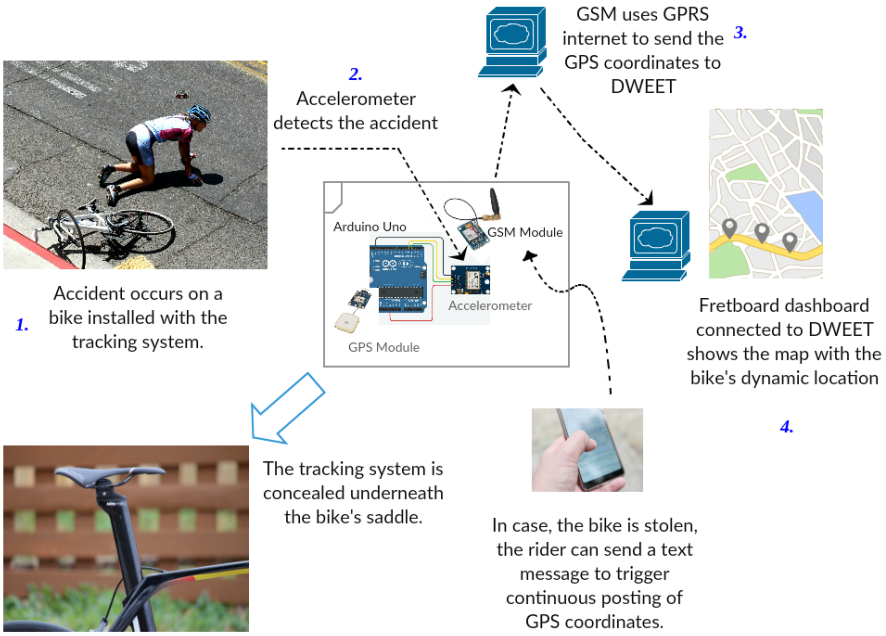


Figure 2.1: Rich Picture depicting the high-level conceptual overview of the system. (Image credit on Appendix section 6.1)

The system needs to take into two scenarios simultaneously. First, it must detect an accident and second, it must also detect a stolen state of the bike so that it can begin pushing the GPS coordinates in a certain interval.

For the first scenario, the accelerometer is used. Since the accelerometer measures the 'G' force on either of the 3-dimensions (Olsson 2012), it can be set up to detect whether the bike's planar position has been subverted, flipped to the right, or flipped to the left. For each of this position defined as the threshold

a constant check is done by the Arduino. Once the threshold of either of this event is exceeded, the state is mapped to an accident. This, in turn, triggers the web request by the GSM module to send the current GPS coordinate pulled from the GPS module.

The first scenario is a fully automated active system. In the second scenario, the system must act passively. The only way to map a stolen bike is to explicitly inform the Arduino that the bike has been stolen. Once a bike has been deemed stolen, the rider sends a text message to the GSM module's phone address. Upon receiving this message the Arduino begins tracking the bike and routinely pushing the GPS coordinates to the DWEET server.

In both cases, Dweet.io is used as a database server which stores the values in a JSON format (Schwartz 2016). This is then used as the data source for the freeboard dashboard. The dashboard is set up to display the coordinates in Google Maps.

2.2 Components required

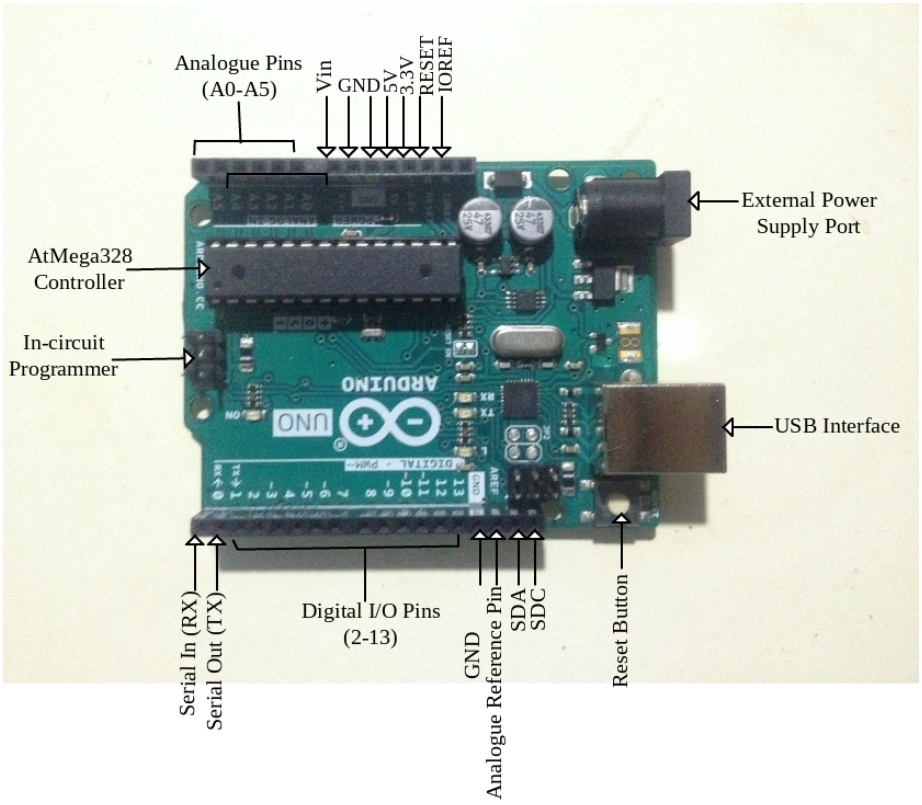
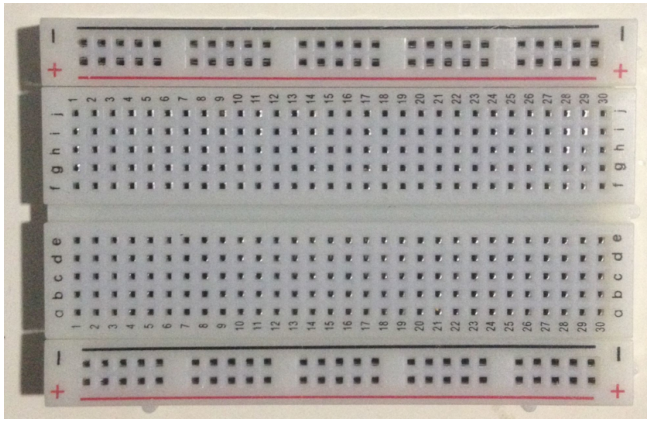


Figure 2.2: Arduino UNO R3

The main microcontroller prototyping platform. Uno is low-cost and has a rich support community on the web. It also has high support for a variety of components such as shields and sensors. (Blum 2018)



(a) Breadboard

(b) SIM Card

Figure 2.3: Breadboard and SIM card

Breadboard for constructing the system circuitry. Sim Card for GSM Module. Must be micro size i.e. (12*15)mm.

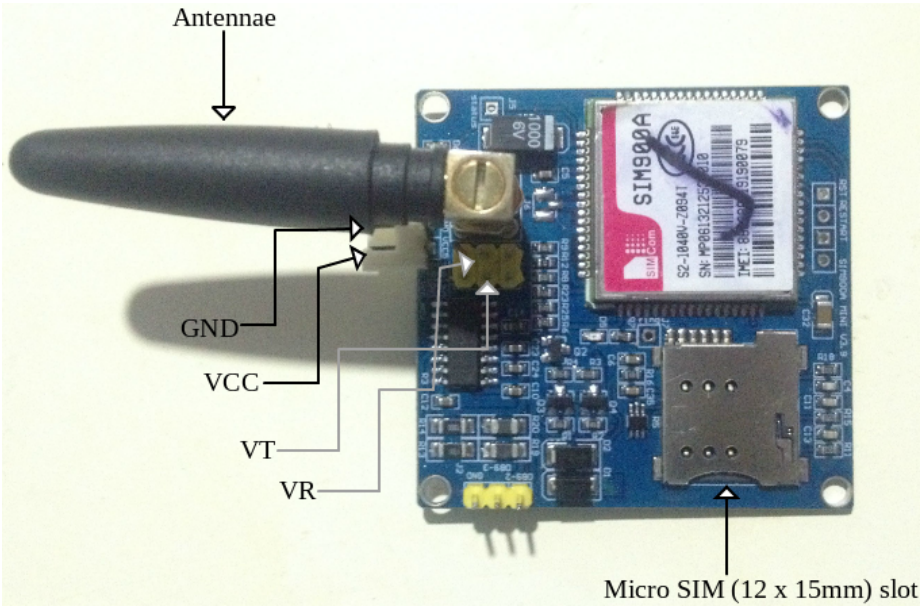


Figure 2.4: GSM Module - SIM900A

For networking with city wide network coverage.

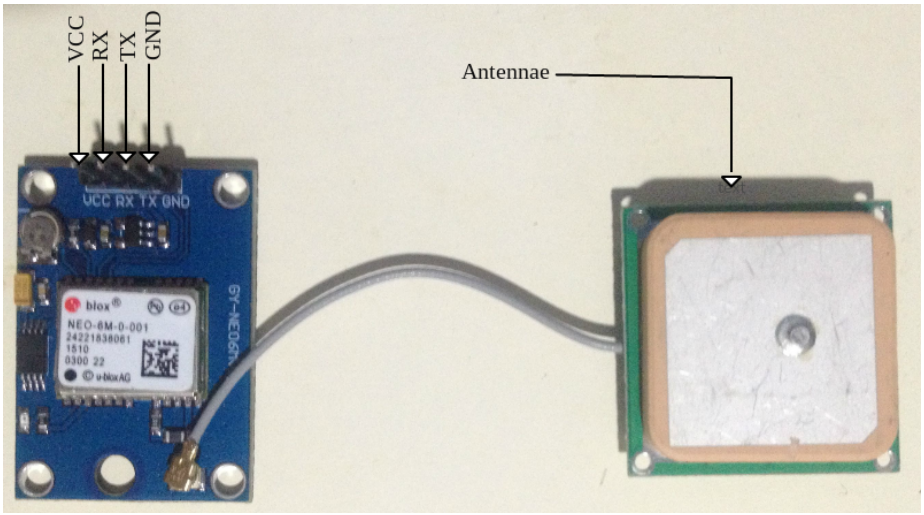


Figure 2.5: GPS Module - Neo-6M

To sense the real-time GPS coordinates.

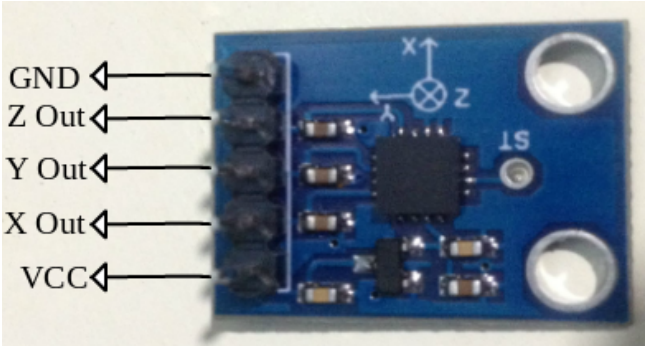


Figure 2.6: Accelerometer - GY-61 / ADXL335
To detect the orientation of the bike in 3-dimension, for accident detection.

Chapter 3

The Build

3.1 Working Mechanism

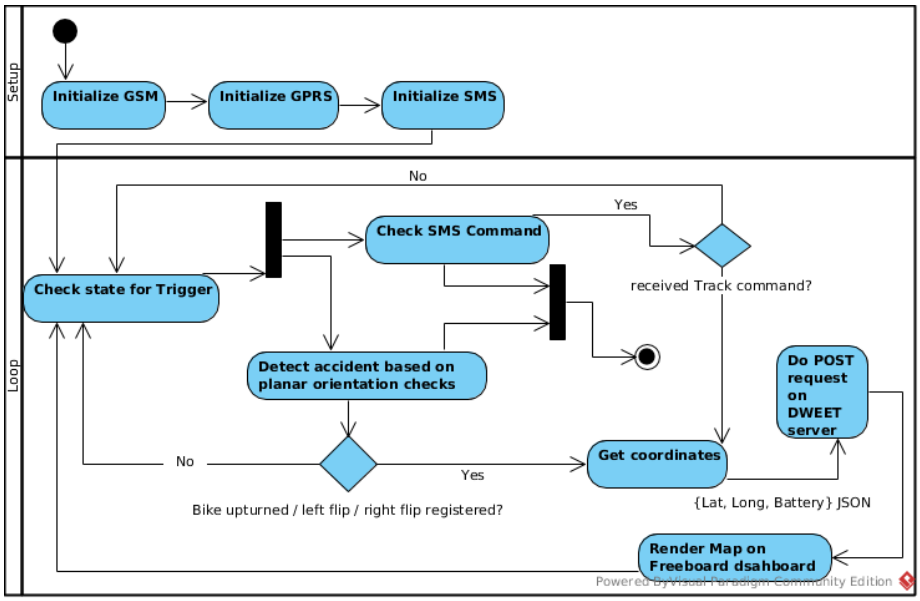


Figure 3.1: Activity Diagram depicting the dynamic model.
(View Legend in section 6.2 of Appendix)

Once the system is activated and deployed in the field, its state is changed by two events. First, when an SMS with track command as the text is received. And second when there is a detection of an accident.

The Arduino checks in an iterative clock cycle, if these events are fired. The GSM module's GPRS feature is used for wide-area network connectivity.

The accident is detected if the accelerometer senses the bike's planar orientation is upturned, flipped to the right, or flipped to the left.

The GPS coordinates are received by the GPS module, NEO-6M and upon the aforementioned events being trigger, the Arduino sends the coordinates via

the GSM to the Dweet server which is rendered in the Freeboard.io dashboard.

3.2 Block Diagram

The block diagram below shows the structural view on the relation and message flow between the components of the system.

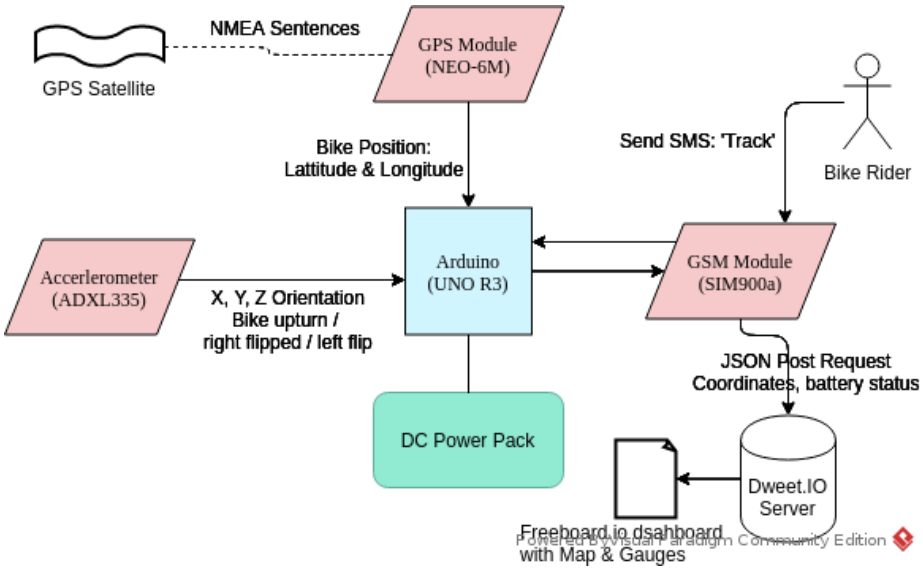


Figure 3.2: Block Diagram depicting the structural model.)

3.3 Software Tools Required

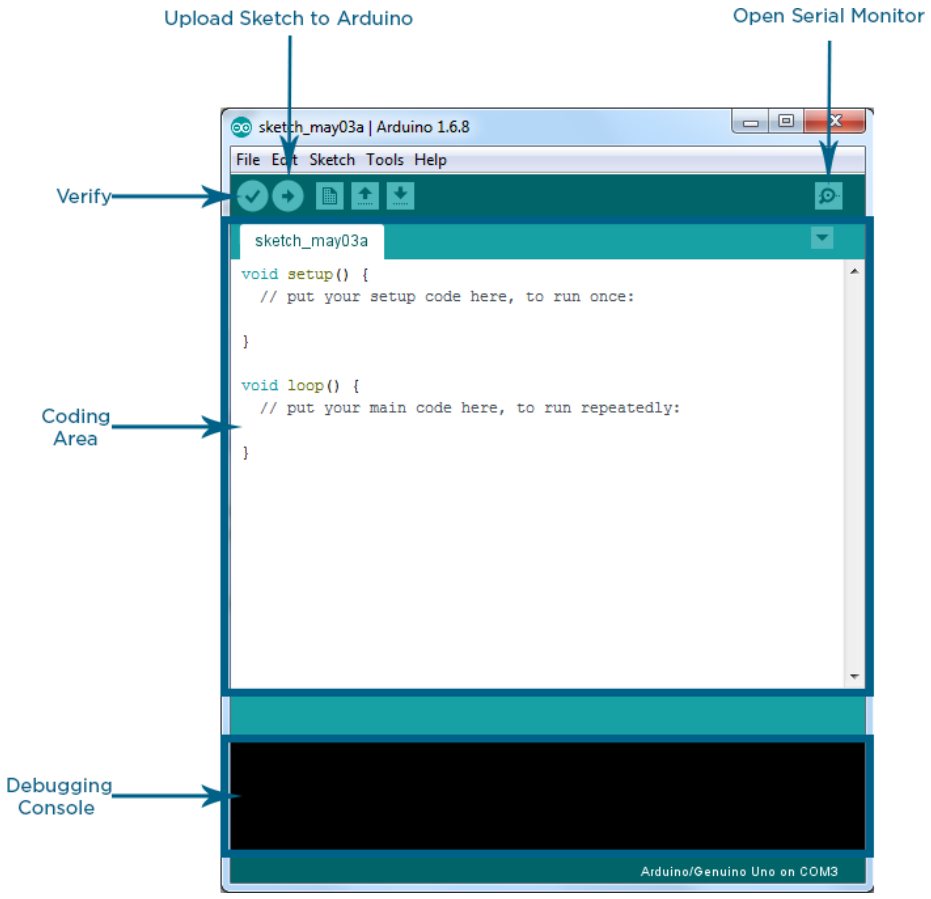


Figure 3.3: Navigating the Arduino IDE)

Software Tool	Category	Description.
Arduino IDE 1.8.9	Integrated Development Environment	IDE to compile, debug, and program the Arduino
Fritzing 0.9.3	Computer Aided Design (CAD)	For diagramming the schematics of the system.
Visual Paradigm CE 15.2	Modelling	To model the dynamic and structural aspects of the system
Creately (online)	Drawing	To draw the rich picture.

Table 3.1: Software tools required used to build the system.

3.4 Libraries

Liabrary Name	Version	Description.
TinyGPS++	1.0.2	To parse the NMEA sentences to the floating point latitude and longitude values.(Hart 2019)
Adafruit FONA library	1.0.6	To enable and employ the GPRS, and SMS functionality from the SIM. (Ada 2019)
SoftwareSerial	1.0	To set the baud rates of the sensors and to allow serial communication on the digital and analaog IO pins of the UNO.(Arduino.cc 2019)

Table 3.2: Libraries required to program the system.

3.5 Procedure to get started

- Set up the connections as shown in figure 3.4
- Take reference of table 3.9 to verify the correct pin configuration.
- Go to 'Tools' then 'Manage Libraries' option. Ensure all correct libaries as shown in table 3.4.
- Go to 'Tools' then 'Board'. Select Arduino/Genuino UNO as the board.
- Go to 'Tools' then 'Port'. Select the correct USB port on the computer where the Arduino is connected.
- Refer to the sketch shown in section 3.10.
- Press key combination - Ctrl/Cmd+ R to verify and upload the code.
- Press key combination - Ctrl/Cmd + U to upload the sketch to UNO.
- Wait fot the sketch to be uploaded.
- Test the system and view the serial monitor for the ouput (Ctrl/Cmd+Shift+M).

If any problems were encountered while following the procedure, refer to the troubleshooting section in 4.1

3.6 Circuit Layout

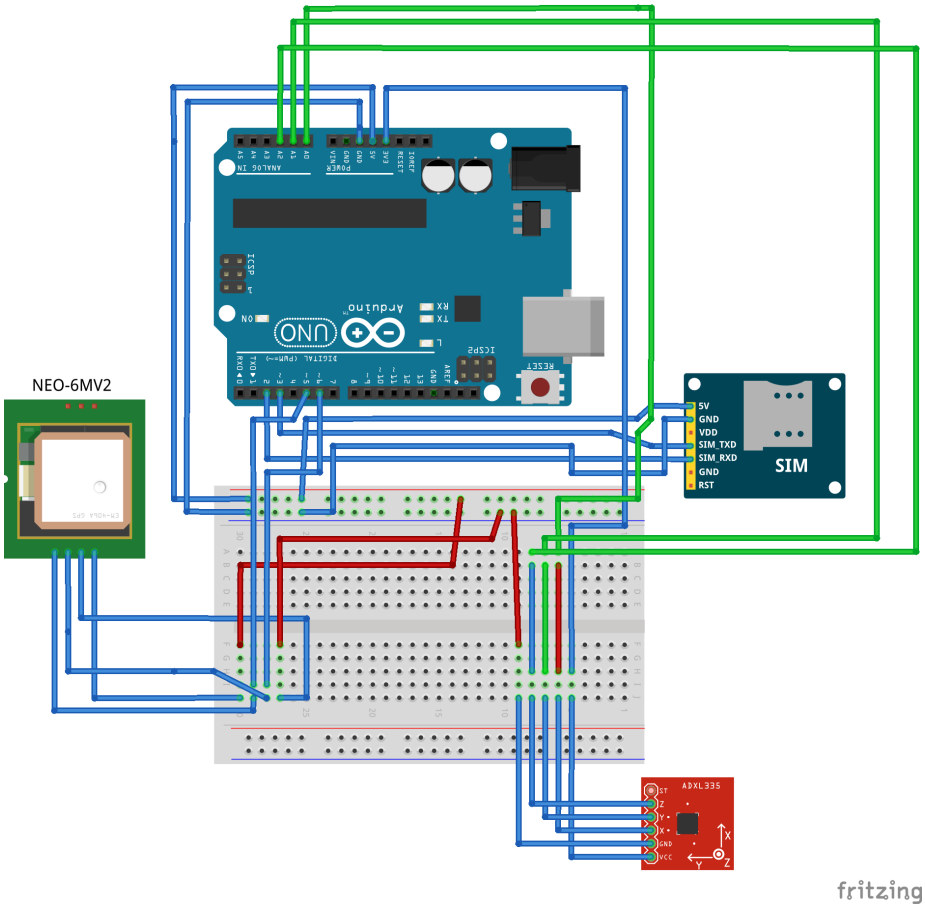


Figure 3.4: Breadboard diagram of the system.

3.7 Schematics

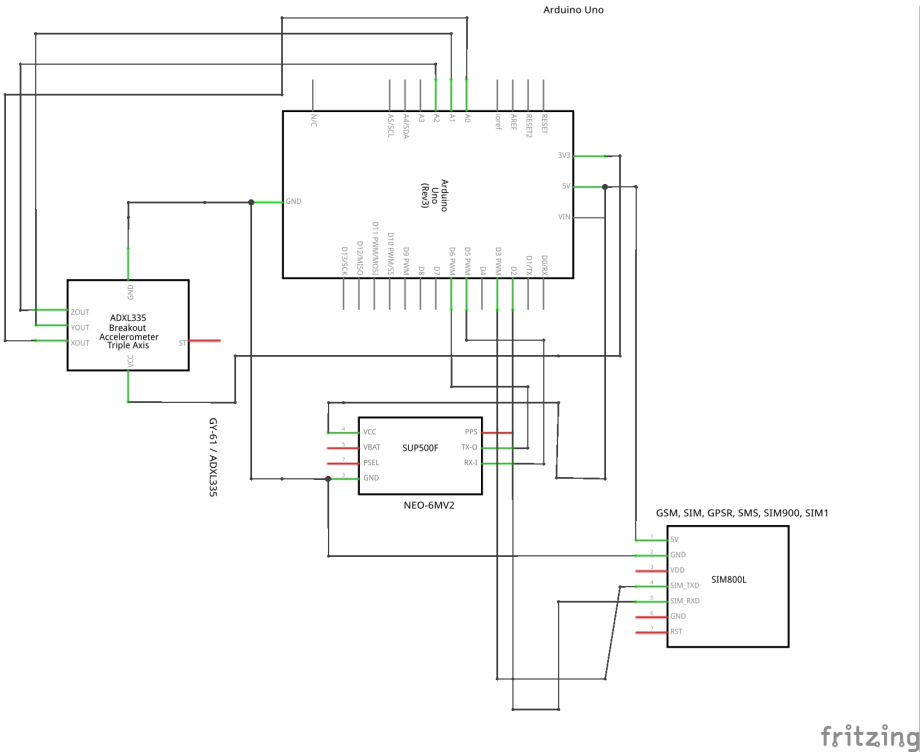


Figure 3.5: Complete schematics of the system.

3.8 PCB Diagram

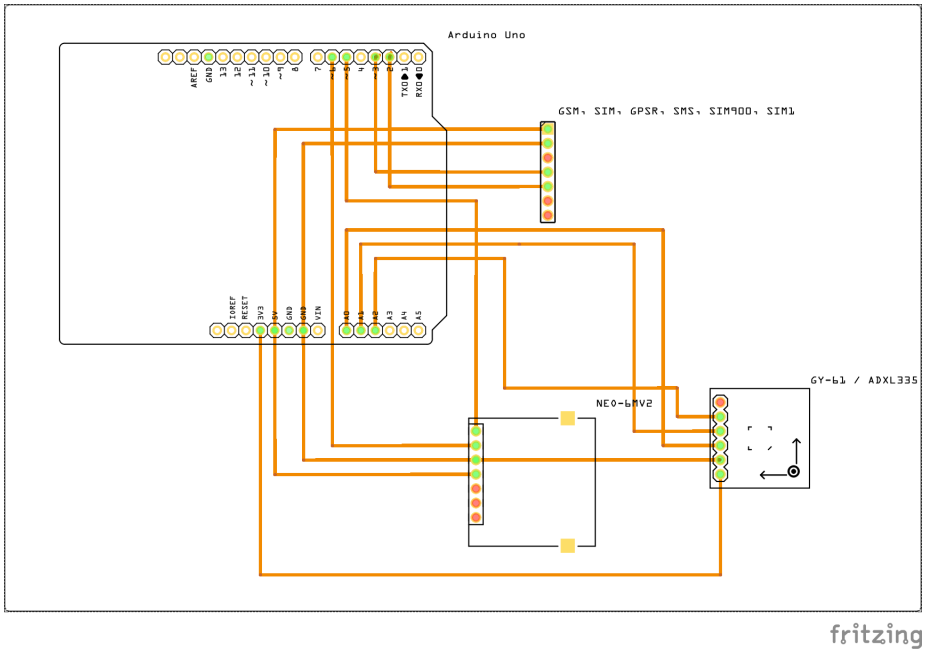


Figure 3.6: Printed circuit board diagram

3.9 Pin Configuration

Component	Pin	Pin No.
GPS (NEO-6M)	GND	GND
GPS (NEO-6M)	VCC	5V
GPS (NEO-6M)	VT	Digital 3
GPS (NEO-6M)	VR	Digital 2
GSM (Sim900a)	GND	GND
GSM (Sim900a)	VCC	5V
GSM (Sim900a)	TX	Digital 6
GSM (Sim900a)	RX	Digital 5
GSM (Sim900a) (Optional)	RST	Digital 4
Accelerometer ADXL335	GND	GND
Accelerometer ADXL335	VCC	3.3V
Accelerometer ADXL335	Xout	Analog 0
Accelerometer ADXL335	Yout	Analog 1
Accelerometer ADXL335	Xout	Analog 2

Table 3.3: Pin Configuration between the arduino and components.

3.10 The Sketch

3.10.1 Variable & Instance declaration

```
// Libraries
#include <TinyGPS++.h>
#include "Adafruit_FONA.h"
#include <SoftwareSerial.h>

TinyGPSPlus gps;
// Intializing GPS & GSM Pins
static const int RX_GPS = 5,
                 TX_GPS = 6,
                 FONA_VR = 2,
                 FONA_VT = 3,
                 FONA_RST = 4;

// ADXL335 Pins
static const int xPin = A0,
                 yPin = A1,
                 zPin = A2;

float zero_G = 512.0,//ADC is 0~1023 the zero g output equal to Vs/2
      scale = 102.4; //ADXL335330 Sensitivity is 330mv/g

// Location variables
double valLat = 27.708637;
double valLng = 85.331215;

//SMS Variables
uint8_t readline(char *buff, uint8_t maxbuff, uint16_t timeout = 0);
char fonaNotificationBuffer[64];//for notifications from the FONA
char smsBuffer[40];
char trackToken[] = "track", dontTrackToken[] = "stop";

// Instances
SoftwareSerial fonaSS = SoftwareSerial(FONA_VT, FONA_VR);
SoftwareSerial ssGPS = SoftwareSerial(TX_GPS, RX_GPS);
static const uint32_t GPSBaud = 9600;

SoftwareSerial *fonaSerial = &fonaSS;
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);

boolean trackGPS = false;
uint16_t vbat;
String dweetThing = "gps_tracker"; // Dweet.io Thing
```

Figure 3.7: Variable and instance declararation for all components.

The required variables are assigned. The pins are selected for analogue and digital IO operations.

3.10.2 Setup Function

```
void setup() {
  Serial.begin(9600);
  ssGPS.begin(GPSBaud);
  delay(100);

  Serial.println("\n**********INITIALIZING GSM...**********\n");
  initGSM();
  Serial.println("\n**********INITIALIZING GPRS...**********\n");
  initGPRS();
  Serial.println("\n**********INITIALIZING SMS...**********\n");
  initSMS();
}
```

Figure 3.8: Setup function

Serial baud rate is set. The GSM, GPRS, and the SMS features are intilialized. The setup function calls the following routines.

GSM intilialisation

```
void initGSM() {
  while (!Serial);
  fonaSerial->begin(4800);
  if (! fona.begin(*fonaSerial)) {
    Serial.println(F("Couldn't find FONA"));
    while (1);
  }
  Serial.println(F("FONA is OK"));
}

void initGPRS() {
  fona.setGPRSNetworkSettings(F("Web")); //fona.setGPRSNetworkSettings(F("your_APN")
  //, F("your_username"), F("your_password"));
  fona.enableGPRS(true);
}

void getBatteryStatus() {
  if (!fona.getBattPercent(&vbat)) {
    Serial.println(F("Failed to read Batt"));
  } else {
    Serial.print(F("Battery status: ")); Serial.print(vbat); Serial.println(F("%"));
  }
}
```

Figure 3.9: GSM Initialisation

GPRS intilialisation

```
void initGPRS() {
  //fona.setGPRSNetworkSettings(F("your_APN")
  fona.setGPRSNetworkSettings(F("Web"));
  //, F("your_username"), F("your_password"));
  fona.enableGPRS(true);
}
```

Figure 3.10: GPRS Initialisation

SMS intilialisation

```
void initSMS() {
  while (!Serial);
  Serial.println(F("FONA SMS caller ID test"));
  Serial.println(F("Initializing....(May take 3 seconds)"));

  // make it slow so its easy to read!
  fonaSerial->begin(4800);
  if (! fona.begin(*fonaSerial)) {
    Serial.println(F("Couldn't find FONA"));
    while (1);
  }
  Serial.println(F("FONA is OK"));

  // Print SIM card IMEI number.
  char imei[16] = {0}; // MUST use a 16 character buffer for IMEI!
  uint8_t imeiLen = fona.getIMEI(imei);
  if (imeiLen > 0) {
    Serial.print("SIM card IMEI: "); Serial.println(imei);
  }

//set up the FONA to send a +CMTI notification when an SMS is receive
  fonaSerial->print("AT+CNMI=2,1\r\n");  d

  Serial.println("FONA Ready");
}
```

Figure 3.11: SMS initialisation

3.10.3 Loop function

```
void loop() {
  delay(2000);
  checkSMS();
  detectBikeAccident();

  if (!trackGPS == false) {
    getGPSInfo();
    postToDweet();
  }
}
```

Figure 3.12: Loop function

Arduino constantly monitors for SMS command to track the GPS. Simultaneously it also monitors the accident. In either case, if the conditional is valid, it gets the GPS information and posts it to DWEET.

Bike Accident detection function

```
void detectBikeAccident() {
  int x = analogRead(xPin);
  int y = analogRead(yPin);
  int z = analogRead(zPin);



  //330 * 1024/3.3/1000
  float _x  = ((float)x - 331.5) / 65 * 9.8;
  float _y = ((float)y - 329.5) / 68.5 * 9.8;
  float _z = ((float)z - 340) / 68 * 9.8;

  Serial.print(_x);
  Serial.print("\t");
  Serial.print(_y);
  Serial.print("\t");
  Serial.print(_z);
  Serial.print("\n");

  if (isBikeUpTurn(_x, _y, _z) || isBikeLeftFlip(_x, _y, _z) || isBikeRightFlip(_x, _y, _z)) {
    Serial.println("\n**********Accident Detected!**********\n");
    getGPSInfo();
    postToDweet();
  }
}
```

Figure 3.13: Accident Detection function

The X,Y,Z orientation are read and checked for threshold defined for bike upturn, left flip, and right flip. It calls the following sub-routines.

```
boolean isBikeUpTurn(float _x, float _y, float _z) {
  boolean status = false;
  if (_x > 1.00 && _y > 1.20 && _z > 10.50) {
    status = true;
    Serial.println("\n**********Bike upturned!**********\n");
  }
  return status;
}

boolean isBikeRightFlip(float _x, float _y, float _z) {
  boolean status = false;
  if (_x > 11.00 && _y > 6.00 && _z > 3.00) {
    status = true;
    Serial.print("\n**********Bike flipped to the right!**********\n");
  }
  return status;
}

boolean isBikeLeftFlip(float _x, float _y, float _z) {
  boolean status = false;
  if (_x < 1.00 && _y < 10.00 && _z > -3.00) {
    status = true;
    Serial.print("\n**********Bike flipped to the left!**********\n");
  }
  return status;
}
```

Figure 3.14: Function to detect bike Upturn, left flip, right flip detection.

Upturn, left flip, right flip detection

SMS check function

```
void checkSMS() {
  char* bufPtr = fonaNotificationBuffer;    //handy buffer pointer
  if (fona.available())      //any data available from the FONA?
  {
    int slot = 0;            //this will be the slot number of the SMS
    int charCount = 0;
    //Read the notification into fonaInBuffer
    do  {
      *bufPtr = fona.read();
      Serial.write(*bufPtr);
      delay(1);
    } while ((*bufPtr++ != '\n') && (fona.available()) && (++charCount < (sizeof(fonaNotificationBuffer) - 1)));

    //Add a terminal NULL to the notification string
    *bufPtr = 0;

    //Scan the notification string for an SMS received notification.
    //  If it's an SMS message, we'll get the slot number in 'slot'
    if (1 == sscanf(fonaNotificationBuffer, "+CMTI: " FONA_PREF_SMS_STORAGE ",%d", &slot)) {
      Serial.print("slot: "); Serial.println(slot);

      char callerIDbuffer[32];  //Storing the SMS sender number in here

      // Retrieve SMS sender address/phone number.
      if (! fona.getSMSSender(slot, callerIDbuffer, 31)) {
        Serial.println("Didn't find SMS message in slot!");
      }
      Serial.print(F("FROM: ")); Serial.println(callerIDbuffer);

      // Retrieve SMS value.
      uint16_t smslen;
      if (fona.readSMS(slot, smsBuffer, 250, &smslen)) { // pass in buffer and max len!
        Serial.println(smsBuffer);
        if (strncmp(smsBuffer, trackToken, sizeof(smsBuffer)) == 0) { // If track request received
          Serial.println("Received Track Request.");
          //Send back an automatic response
          Serial.println("Sending reponse...");
          if (!fona.sendSMS(callerIDbuffer, "Tracking GPS...Check your freeboard.io dashboard.")) {
            Serial.println(F("Failed"));
          } else {
            Serial.println(F("Sent!"));
```

Figure 3.15: Function to monitor track command via SMS.

```
          }
          trackGPS = true;
        } else if (strncmp(smsBuffer, dontTrackToken, sizeof(smsBuffer)) == 0) { // If stop request receivec
          trackGPS = false;
        }
        // delete the original msg after it is processed
        // otherwise, we will fill up all the slots
        // and then we won't be able to receive SMS anymore
        if (fona.deleteSMS(slot)) {
          Serial.println(F("OK!"));
        } else {
          Serial.print(F("Couldn't delete SMS in slot ")); Serial.println(slot);
          fona.print(F("AT+CMGD=?\r\n"));
        }
      }
    }
  }
}
```

Figure 3.16: Function to monitor track command via SMS.

Function to monitor track command via SMS. This is checked in loop by the arduino.

Get GPS Coordinates

```
void getGPSInfo() {
  while (ssGPS.available() > 0) {
    gps.encode(ssGPS.read());
    if (gps.location.isUpdated()) {
      valLat = (gps.location.lat(), 6);
      Serial.print("Latitude = ");
      Serial.print(gps.location.lat(), 6);

      valLng = (gps.location.lng(), 6);
      Serial.print(" Longitude = ");
      Serial.println(gps.location.lng());
    }
  }
}
```

Figure 3.17: Function to get the parsed GPS coordinates.

NMEA Sentences - Parsing GPS Data The National Marine Electronics Association (NMEA) is an organisation that has standardized the communication interface for GPS receivers. The data intercepted by the GPS is read as what is called a NMEA sentence. (NMEA.org 2019)

NEO-6M GPS module in its registers NMEA sentences in raw readings. Each sentence is prepended by a dollar symbol and 5 uppercase letters. The content is separated by a comma.

The reference to the meanings of NMEA sentence are given below.

- **110617** – represents the time at which the fix location was taken, 11:06:17 UTC
- **41XX.XXXXX,N** – latitude 41 deg XX.XXXXX' N
- **00831.54761,W** – Longitude 008 deg 31.54761' W
- **1** – fix quality (0 = invalid; 1= GPS fix; 2 = DGPS fix; 3 = PPS fix; 4 = Real Time Kinematic; 5 = Float RTK; 6 = estimated (dead reckoning); 7 = Manual input mode; 8 = Simulation mode)
- **05** – number of satellites being tracked
- **2.68** – Horizontal dilution of position
- **129.0, M** – Altitude, in meters above the sea level
- **50.1, M** – Height of geoid (mean sea level) above WGS84 ellipsoid
- empty field – time in seconds since last DGPS update
- empty field – DGPS station ID number
- ***42** – the checksum data, always begins with *

Figure 3.18: NMEA codes and their meanings.
(DePriest 2019)

- **$GPGSA** – GPS DOP and active satellites
- **$GPGSV** – Detailed GPS satellite information
- **$GPGLL** – Geographic Latitude and Longitude
- **$GPRMC** – Essential GPS pvt (position, velocity, time) data
- **$GPVTG** – Velocity made good

Figure 3.19: NMEA acronyms and their meanings.
(DePriest 2019)

TinyGPS++ library, the library developed by Mikal Hart to interface with the NEO-6M (Hart 2019) was used to parse the NMEA sentence to specific string tokens. For the project, latitude and longitude values in 6 decimal places were extracted with the help of this library.

Post to DWEET.IO

```
void postToDweet() {
  // Prepare request
  String strLat = "", strLng = "";
  strLat = String(valLat, 6);
  strLng = String(valLng, 6);

  getBatteryStatus();
  uint16_t statuscode;
  int16_t length;
  char url[90];
  String request = "www.dweet.io/dweet/for/";
  request += dweetThing;
  request += "?latitude=" + strLat;
  request += "&longitude=" + strLng;
  request += "&battery=" + String(vbat);
  request.toCharArray(url, request.length() + 1);

  Serial.println(url);
    // Send location to Dweet.io
    if (!fona.HTTP_GET_start(url, &statuscode, (uint16_t *)&length)) {
      Serial.println("Failed!");
    }
    while (length > 0) {
      while (fona.available()) {
        char c = fona.read();
        // Serial.write is too slow, we'll write directly to Serial register!
  #if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
        loop_until_bit_is_set(UCSR0A, UDRE0); /* Wait until data register empty. */
        UDR0 = c;
  #else
        Serial.write(c);
  #endif
        length--;
      }
    }
    fona.HTTP_GET_end();
}
```

Figure 3.20: Function to Post coordinates to DWEET.IO

This function adapted from Swartzhz in his book, 'Internet of Things with arduino Cookbook' (Schwartz 2016), creates a JSON payload of GPS coordinates. Then it uses the GPRS of the SIM900 to make a POST request to the

DWEET.IO server. Once, the server receives the data it immediately updates the Freeboard dashboard's google map. (Schwartz 2016)

3.11 Final Prototype

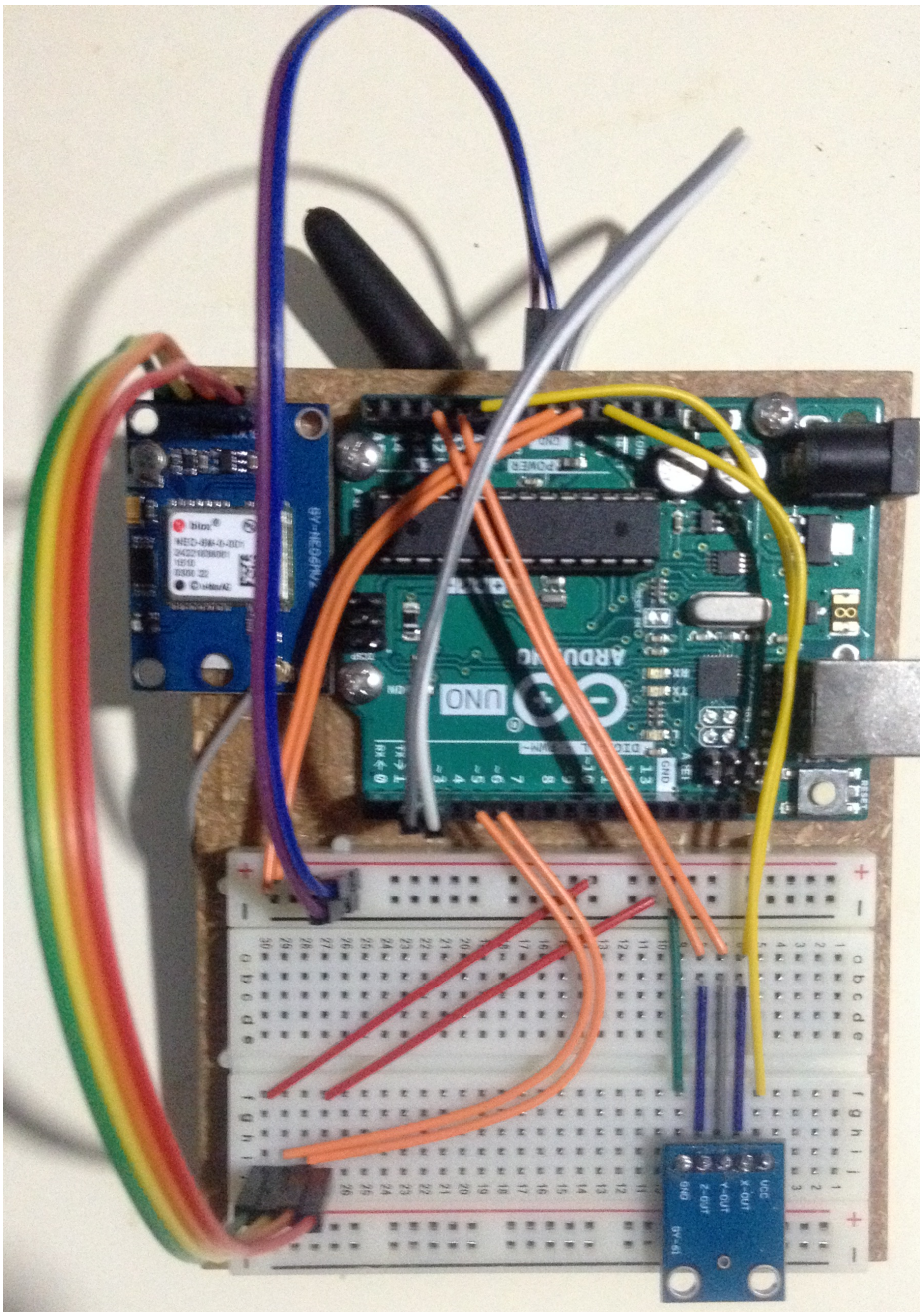


Figure 3.21: Final Prototype (Front View)

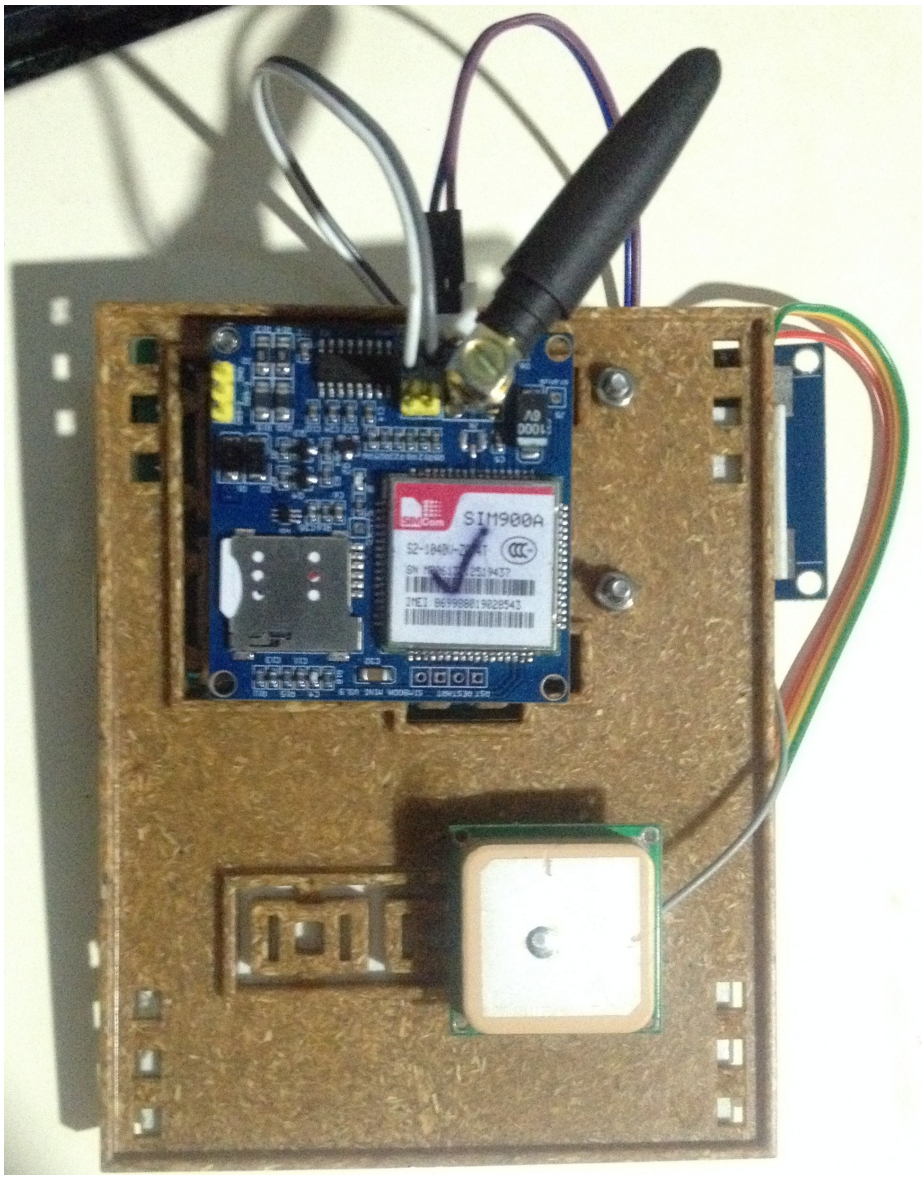


Figure 3.22: Final Prototype (Back View)

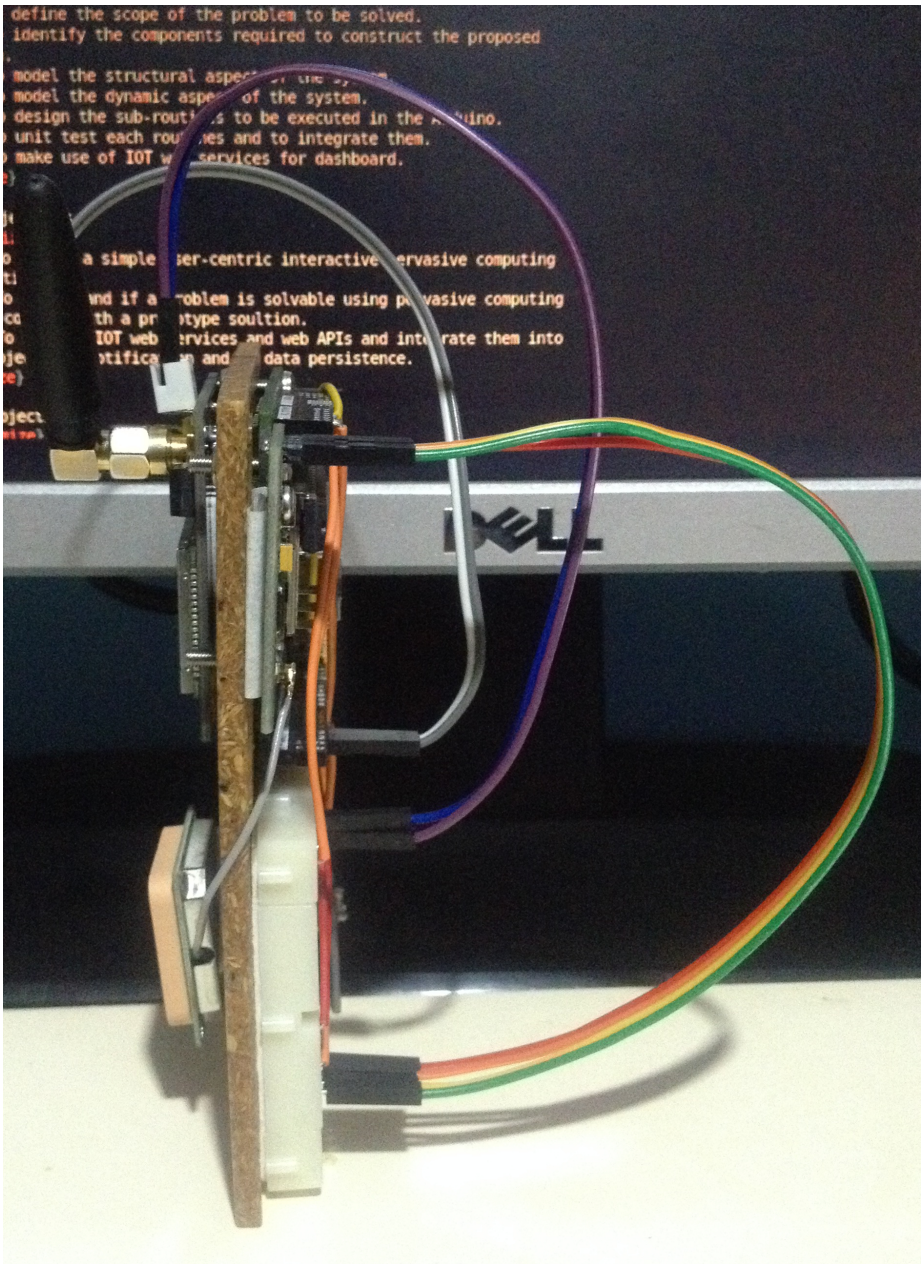


Figure 3.23: Final Prototype (Side View)

3.11.1 Tracking Result

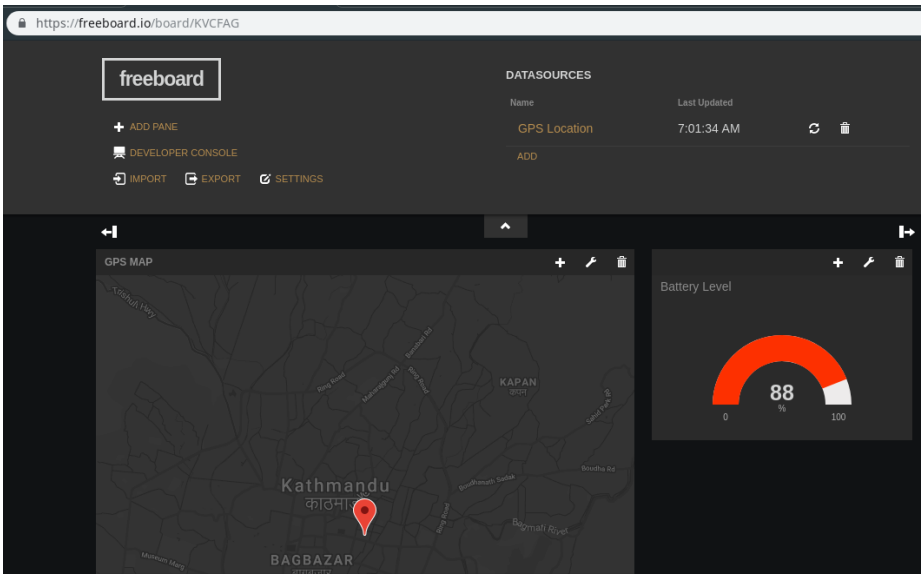


Figure 3.24: Freeboard.io dashboard with Google map marker showing the bike location and the gauge showing battery level.)

3.11.2 Video demonstration

Hyperlink: Google drive Link to the video demo.

https://drive.google.com/open?id=1-1lDFwuljtRxMAXbj-Njmb2W8TyHx9WU

Chapter 4

Troubleshooting

4.1 Troubleshooting

This section deals with the possible pittfalls or the things that could go wrong while configuring the system. Since, there are 3 components that needs to be integrated it is quite complex to manage everything and get it right in the first attempt. Following issues arose during implementation and testing. The steps that were followed to remedy them are also provided below.

4.1.1 Accelerometer Issues

- Check if the soldering on the pins are neatly done.
- Check if the pins are connected in the right order.
- Ensure the module is connected to 3.3V and not 5V. 5V would damage it.
- Ensure the pins are correctly defined in the IDE.

4.1.2 GPS Module Issues

- GPS module requires a clear line of sight towards the sky. It needs to make link with the satellite.
- Check if the soldering on the pins are neatly done.
- Check if the pins are connected in the right order.
- Ensure the pins are correctly defined in the IDE.
- Check if the antennae head is connected well with the module.

4.1.3 GSM Module Issues

- Wait for the module to gain signal. It blinks 3 times a second (Javed 2016) once it has good range.
- Ensure the SIM has enough credit.

- Ensure the SIM functions right in regular phone.
- Ensure the SIM card is clicked in place, into the slot.
- Check if the soldering on the pins are neatly done.
- Check if the pins are connected in the right order.
- Ensure the pins are correctly defined in the IDE.
- Check if the antennae head is tightened well.
- Test several functions with the library example of Adafruit Fona (Schwartz 2016).

4.1.4 Arduino UNO Issues

- Ensure the correct board is selected.
- Ensure the correct port is selected.
- Check for error codes and library import issues.
- Ensure compile time errors are fixed before uploading.

Chapter 5

Other Applications

5.1 Other Application

The possible deployment domains where my system can cater-to area as follows.

- To track the riders and assess their safety in a competetive cycle sport event such as trail bike, enduro, and mountain bike racing.
- Can be used to explore new or unfamiliar places without fear of losing the route. The route can be traced back to the origin. This could be benificial for someone new to a place such as a foreign immigrant or a newly admitted campus student who needs to navigate large areas.
- Can be installed on delivery bikes of the food or courier agencies to track orders and ensure their safety.

5.2 Future Work

5.2.1 Areas of Improvement

- Minify device footprint replacing UNO with Arduino Nano and the bread-board with matrix board.
- Use waterproof casing.
- Use 4G for better connectivity and low latency.

5.2.2 Areas of Extension

- Phone interface to program preferences and calibration.
- Use custom web server and richer dashboard.

- Leverage other GPS data over 4G - velocity, direction.

- Enable data persistence on local datastore for offline storage and on cloud for data analysis and analytics.

5.3 Lessons Learnt

- Analysis, design, implementation of Interactive Pervasive Computing solutions.
- Research, troubleshooting, and replicable documentation authoring skills.

- Translating data from physical phenomena into valuable utility for better decision making and for enhancing quality of life.

- Usabality and comprehensible results must be taken into account at every phase of system development.

- Engineering must be done with ethics and human service as an end in mind.

- Pervasive computing can be a means to a solving humanitarian issues.

Chapter 6

Appendix

6.1 Image Credit

Sources of image used in section 2.1. The permission set by the authors of all images used in this report allows for sharing, modification, and reuse.

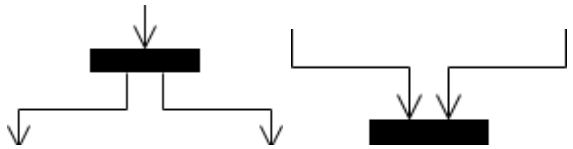
- Author: Ben Morrow
  Caption: Bicycle crash victim in Tucson
  Source: https://www.flickr.com/photos/billmorrow/5629782610
  License: Attribution 2.0 Generic (CC BY 2.0)
- Author: Dave Haygarth
  Caption: The bicycle thief
  Source: https://www.flickr.com/photos/minnellium/2865435779
  License: Attribution 2.0 Generic (CC BY 2.0)
- Author: Glory Cycles
  Caption: Look 795 Blade RS Seat Post
  Source: https://www.flickr.com/photos/glorycycles/33237568718
  License: Attribution 2.0 Generic (CC BY 2.0)
- Author: Skitterphoto
  Caption: Using phone smartphone internet hand screen
  Source: https://pixabay.com/photos-using-phone-smartphone-internet-3733354
  License: Pixabay License

6.2 Activity Diagram Legend

The diagram models the dynamic or behavioral aspects of the system. (Paradigm 2019)

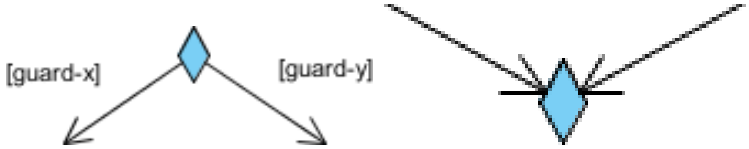
(a) Activity (b) Control Flow (c) Initial node (d) Final node

Activity represents a set of processes or actions. The entry and exit points of the logic is shown by the black circular nodes.



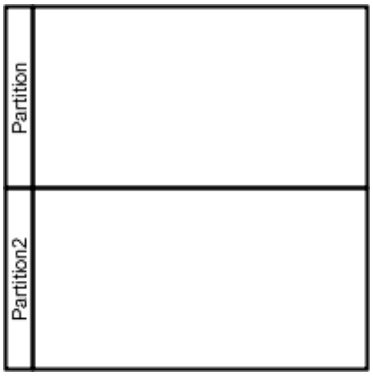
(a) Fork node (b) Join node

Forks and joins are used to split and bring-back concurrent activities.



(a) Decision node (b) Merge node

Decision and merge nodes are used to split and bring-back branches or conditionals in logic. The conditionals also called the guards are placed over the control flow lines.



(a) Swim Lane

Swim lane outlines the area of responsibility for the actors involved in initiating and handling the activities.

6.3 Word Count

```
File: biju_doc.tex
Encoding: ascii
Words in text: 1933
Words in headers: 118
Words outside text (captions, etc.): 203
Number of headers: 54
Number of floats/tables/figures: 37
Number of math inlines: 0
Number of math displayed: 0
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  0+2+0 (1/0/0/0) _top_
  0+1+0 (1/0/0/0) Chapter: Introduction} \input{chapters/01.introduction
  110+2+0 (1/0/0/0) Section: Project Motivation
  30+2+0 (1/0/0/0) Section: Problem Statement
  31+1+0 (1/0/0/0) Section: Aim
  0+1+0 (1/0/0/0) Section: Objectives
  72+1+0 (1/0/0/0) Subsection: Technical
  47+1+0 (1/0/0/0) Subsection: Academic
  35+1+0 (1/0/0/0) Subsection: Personal
  0+1+0 (1/0/0/0) Chapter: Justification} \input{chapters/02.justification
  263+2+15 (1/1/0/0) Section: Rich Picture}\label{rich_picture
  65+2+19 (1/5/0/0) Section: Components required
  0+2+0 (1/0/0/0) Chapter: The Build} \input{chapters/03.build
  122+2+12 (1/1/0/0) Section: Working Mechanism
  20+2+6 (1/1/0/0) Section: Block Diagram
  0+3+12 (1/2/0/0) Section: Software Tools Required
  0+1+6 (1/1/0/0) Section: Libraries} \label{lib
  131+4+0 (1/0/0/0) Section: Procedure to get started
  0+2+5 (1/1/0/0) Section: Circuit Layout
  0+1+5 (1/1/0/0) Section: Schematics
  0+2+4 (1/1/0/0) Section: PCB Diagram
  0+2+7 (1/1/0/0) Section: Pin Configuration} \label{pin
  0+2+0 (1/0/0/0) Section: The Sketch}\label{sketch
  15+3+7 (1/1/0/0) Subsection: Variable \& Instance declaration
  21+8+8 (4/4/0/0) Subsection: Setup Function
  251+26+52 (7/9/0/0) Subsection: Loop function
  0+2+12 (1/3/0/0) Section: Final Prototype
  0+2+16 (1/1/0/0) Subsection: Tracking Result
  9+2+0 (1/0/0/0) Subsection: Video demonstration
  0+1+0 (1/0/0/0) Chapter: Troubleshooting} \input{chapters/04.troubleshooting
  62+1+0 (1/0/0/0) Section: Troubleshooting}\label{shoot
  43+2+0 (1/0/0/0) Subsection: Accelerometer Issues
  59+3+0 (1/0/0/0) Subsection: GPS Module Issues
  94+3+0 (1/0/0/0) Subsection: GSM Module Issues
  28+3+0 (1/0/0/0) Subsection: Arduino UNO Issues
  0+2+0 (1/0/0/0) Chapter: Other Applications} \input{chapters/05.other_applications
  105+2+0 (1/0/0/0) Section: Other Application
```

```
0+2+0 (1/0/0/0) Section: Future Work
25+3+0 (1/0/0/0) Subsection: Areas of Improvement
39+3+0 (1/0/0/0) Subsection: Areas of Extension
73+2+0 (1/0/0/0) Section: Lessons Learnt
0+1+0 (1/0/0/0) Chapter: Appendix} \input{chapters/05.appendix
92+2+0 (1/0/0/0) Section: Image Credit}\label{imgcr
91+3+17 (1/4/0/0) Section: Activity Diagram Legend}\label{actlgn
0+2+0 (1/0/0/0) Section: Word Count
```

Bibliography

Ada, Lady (2019). *Arduino Test — Adafruit FONA — Adafruit Learning System*. URL: https://learn.adafruit.com/adafruit-fona-mini-gsm-gprs-cellular-phone-module/arduino-test (visited on 05/23/2019).

Arduino.cc (2019). *SoftwareSerial Library*. URL: https://www.arduino.cc/en/Reference/SoftwareSerial (visited on 05/23/2019).

Blum, Jeremy (2018). *Exploring Arduino: Tools and Techniques for Engineering Wizardry*. OCLC: 1008900456. Canada: Wiley & Sons Canada, Limited, John. ISBN: 978-1-119-40537-5.

DePriest, Dale (2019). *NMEA data*. URL: https://www.gpsinformation.org/dale/nmea.htm#nmea (visited on 05/23/2019).

Hart, Mikal (2019). *TinyGPS++ — Arduiniana*. URL: http://arduiniana.org/libraries/tinygpsplus/ (visited on 05/23/2019).

Javed, Adeel (2016). *Building Arduino Projects for the Internet of Things 2016*. OCLC: 951006293. Berkley: APress. ISBN: 978-1-4842-1939-3.

NMEA.org (2019). *Marine Electronics — Boating Electronics — About National Marine Electronics Association*. Organizational Profile. URL: https://www.nmea.org/content/about_the_nmea/about_the_nmea.asp (visited on 05/14/2019).

Olsson, Tony (2012). *Arduino wearables*. OCLC: 872670661. Berkley, California: Apress. ISBN: 978-1-4302-4359-5.

Paradigm, Visual (2019). *What is Activity Diagram?* URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/ (visited on 05/23/2019).

Schwartz, Marco (2016). *Internet of Things with Arduino Cookbook*. 1st ed. OCLC: 1020708470. Birmingham, UK. ISBN: 978-1-78588-331-6.