ESE 4009: EMBT PROJECT

Lambton College in Toronto

Instructor: Takis Zourntos

Student names: Amandeep Singh (C0765434)

Goutham Reddy Alugubelly (C0747981)

Parth Patel (C0764929)

Vishal Hasrajani (C0761544)

INDEX

ESE 4009: EMBT PROJECT

Project Concept

In today's world car is considered to be the safest vehicle on road, our approach is focused on making day to day commuting vehicles like bicycle safer for the riders. So, our team is planning to develop a 'Smart Bag' that would have two cameras, one located at the front and the other located at the back to provide a 360° view in the recorded videos. Also, it'll have a GPS sensor and an accelerometer to make sure that the location of user is kept in records while the accelerometer would be used to detect the any sudden changes in the trajectory of the path in which the bike is moving, while leaving a scope to improve the project if our team ever intends to make any kind of incremental upgrades to the mod.

The main motive behind this dual camera setup is to collect a footage that could be used for future investigations in case of an accident, theft, mugging etc. The way our project works is it'll start recording a new video when there's a sudden shift in axes and g-force which might not be a result of casual or normal peddling behaviour of the rider, so in short there will be two videos, the first one which was being recorded before the change in the data provided by the accelerometer namely 'Normal-Video**.avi**' and the second footage which started recording after the accelerometer was triggered namely 'ACC-Crash**.avi**' (fig-1).

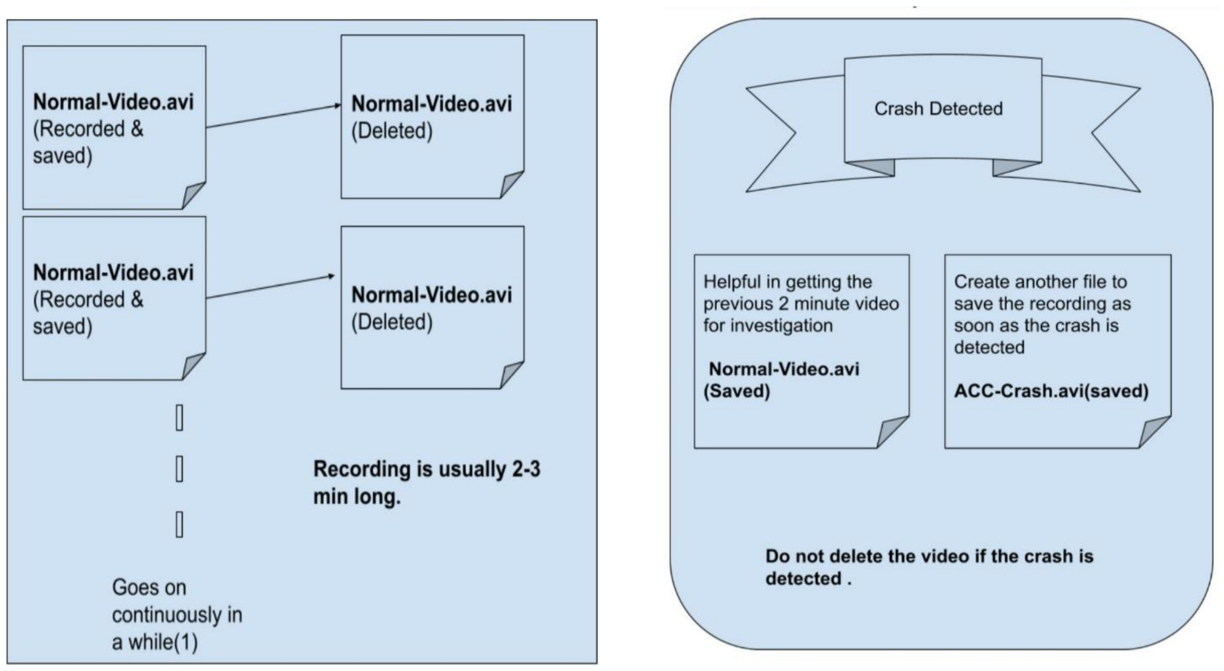

Normal-Video.avi (Recorded & saved)
Normal-Video.avi (Deleted)
Normal-Video.avi (Recorded & saved)
Normal-Video.avi (Deleted)
Recording is usually 2-3 min long.
Goes on continuously in a while(1)
Crash Detected
Helpful in getting the previous 2 minute video for investigation
Normal-Video.avi (Saved)
Create another file to save the recording as soon as the crash is detected
ACC-Crash.avi(saved)
Do not delete the video if the crash is detected .

Fig-1 (How the videos are made)

Since we are making a smart bag, it'd be a shame if it couldn't charge itself while the rider enjoys his journey. So, we decided to add a dynamo to our project that'd simple be used to generate enough power to charge our setup via the micro-USB cable, and since wires can be a little frustrating to deal with at time, we decided to place the charging port on the right side of the bag (Fig 2), which would allow the rider to charge the bag without taking out the components out of the bag. Once the bag is done charging this micro-USB cable can we used to charge a power bank or phone allowing a little bit of flexibility.

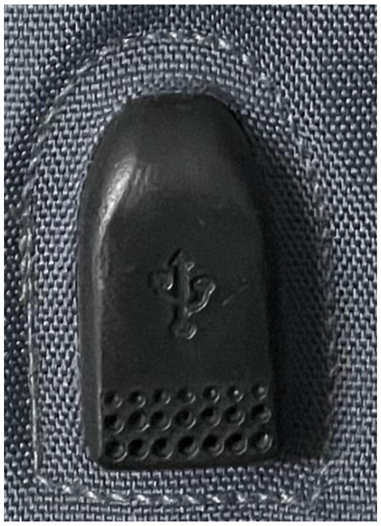


Fig 2 Micro USB-A port for charging

Since, our smart bag is going to record the video in case of an incident, we thought it'd be really helpful if the bag could store the location of the incident as well. In order to provide the coordinates, we can either create a log or try to incorporate the location data within the video which is possible via OpenCV. Once there is a sufficient amount of the data on the external micro-SD card of our raspberry pi 4, we can transfer all of that data to an external hard disk which the user can then take our and save the video/snaps directly to his home computer or android device.

NOTE: One additional feature that we might provide with the incremental upgrade is the ability to share these video and live location via AWS-SNS to a selected group of people.

The final products will look something like this:

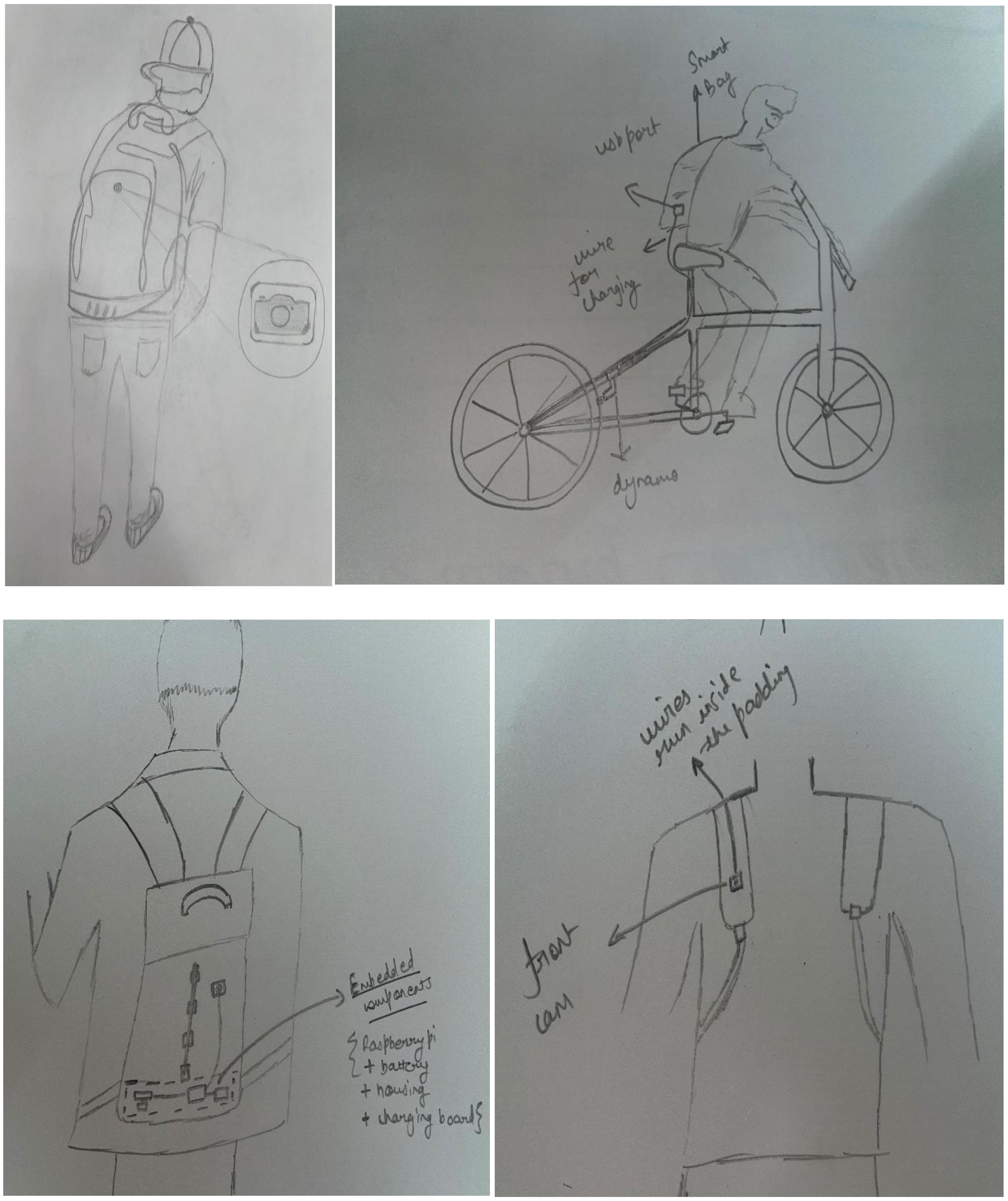


Fig 3 Visualizing the product

System Level Architecture

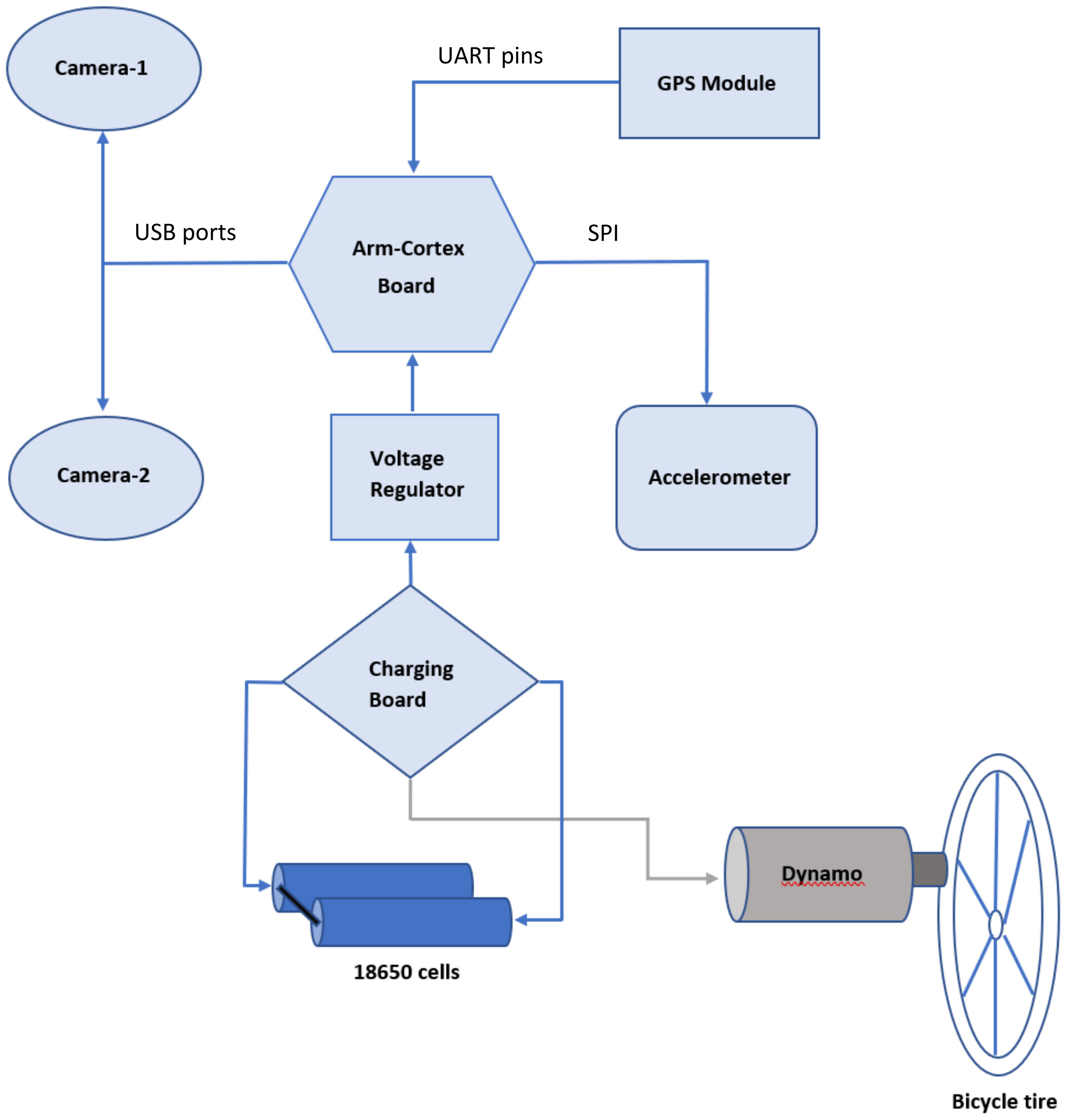
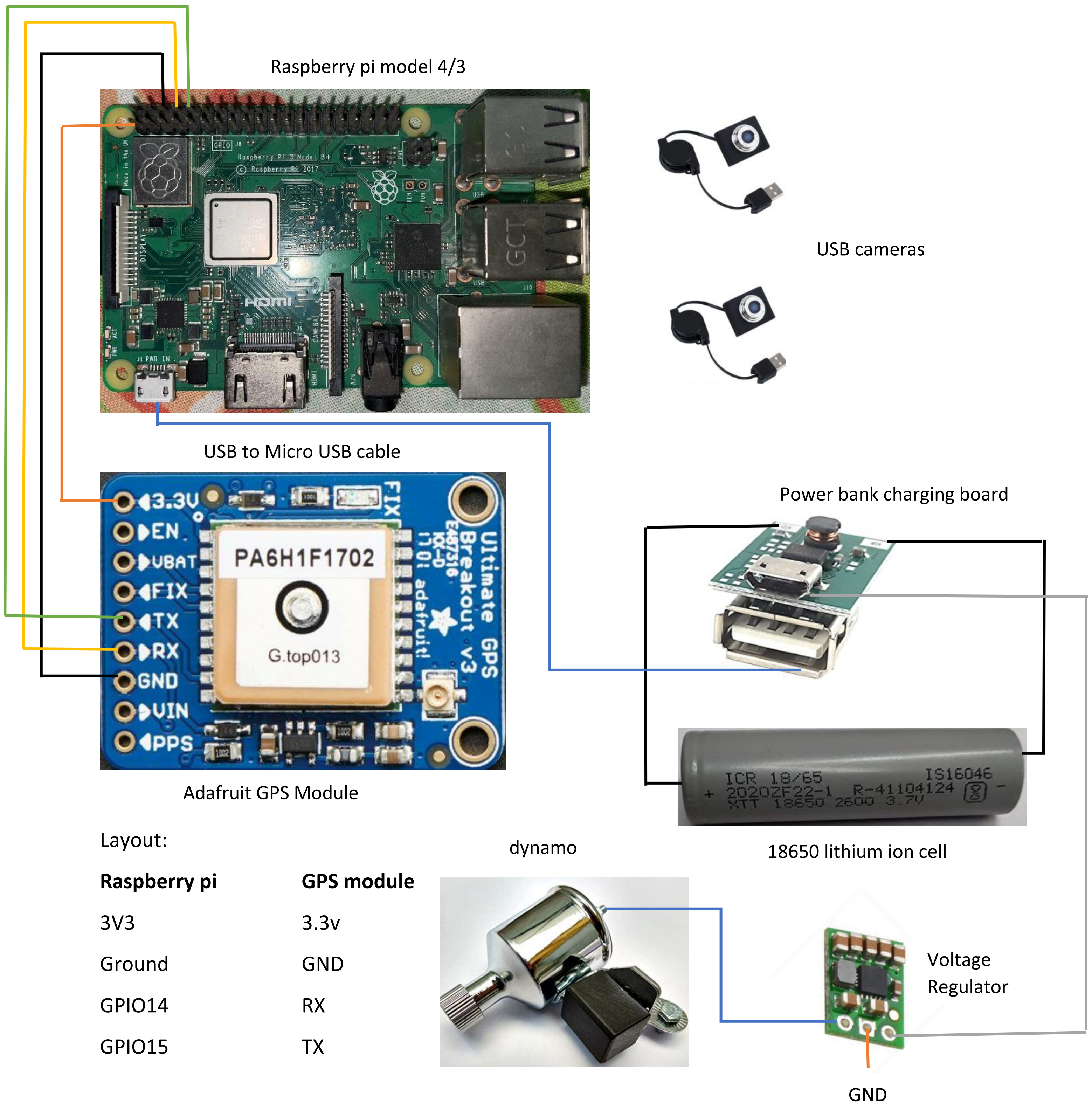


Fig 4 System level architecture

Hardware design



Layout:

Raspberry pi	GPS module
3V3	3.3v
Ground	GND
GPIO14	RX
GPIO15	TX

Note: the cameras will be connected via the USB ports so these won't be needed to solder and this allows us to replace the cameras if they ever stop working.

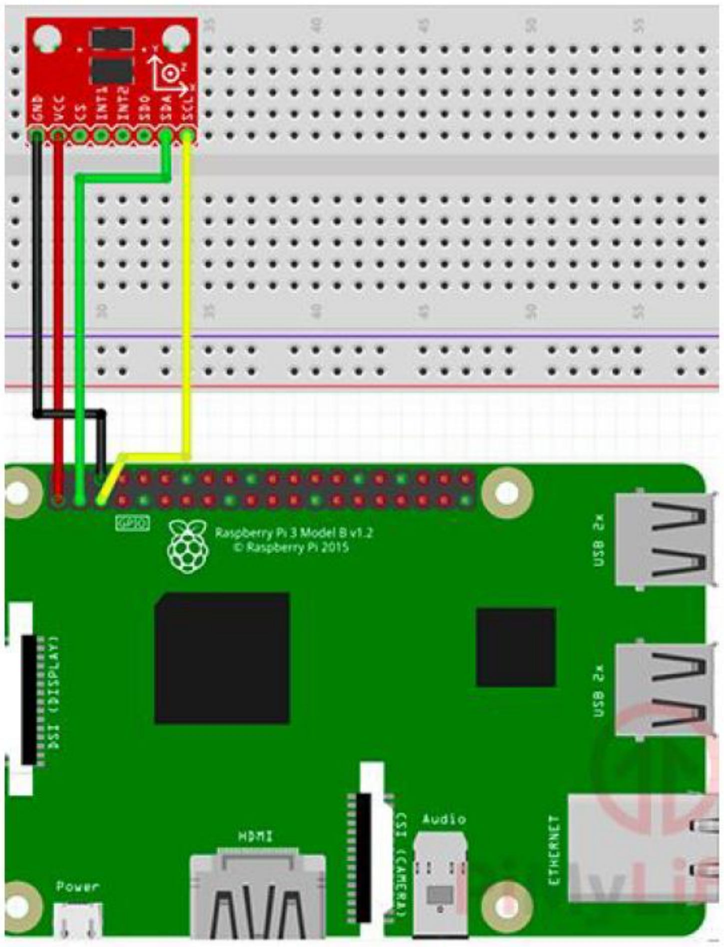


Fig 5 ADXL345 interfacing with the raspberry pi

- Wire the GND pin of the Accelerometer to Physical Pin 6 (GND) on the Raspberry Pi.
- Wire the VCC pin of the Accelerometer to Physical Pin 1 (3v3) on the Raspberry Pi.
- Wire the SDA pin of the Accelerometer to Physical Pin 3 (SDA) on the Raspberry Pi.
- Wire the SCL pin of the Accelerometer to Physical Pin 5 (SCL) on the Raspberry Pi.

To begin with, we will first look at the different components that we are using.

- **Dynamo:** The dynamo that we'll be using is a 12v 6w nickel plated iron dynamo which uses the mechanical rotation of the wheels of a bicycle to generate DC power as shown in the fig 4.

- **Lithium-Ion Cells:** These 18650 cells will be used to store the energy generated by the Dynamo. Which will be used for charging the phone or providing the power to the raspberry pi.

- **Regulator:** The S7V7F5 step-up/step-down voltage regulator provides 5v of constant voltage when the input voltage lies between 2.7v-11.8v which is perfect for our use case scenario. Also, this regulator does so will being 85-90% efficient thus preventing excessive power loss.

- **Camera module 1 & 2:** These cameras will be attached to the USB port of the raspberry pi. It will help us provide a 360° footage.

- **Raspberry pi 4:** We are using raspberry pi 4 as it allows us to implement Open-CV libraries that we need for our project.

- **GPS module:** Adafruit ultimate GPS breakout with 66 channels needs 5v and only 20mA of current to work which is sufficient for out product.

- **Accelerometer:** This is the core part of our embedded project which will help us detect a sudden fall or change in the g-force allowing us to trigger our recordings.

Software design

In this section, we will describe the flowchart of our program and different libraries that we are going to use:

- **#include <opencv2/highgui.hpp>**

- **#include <ctime>**
  This function is used to change the given time to machines local time and then in the form of a character.

- **#include <cstdio>**

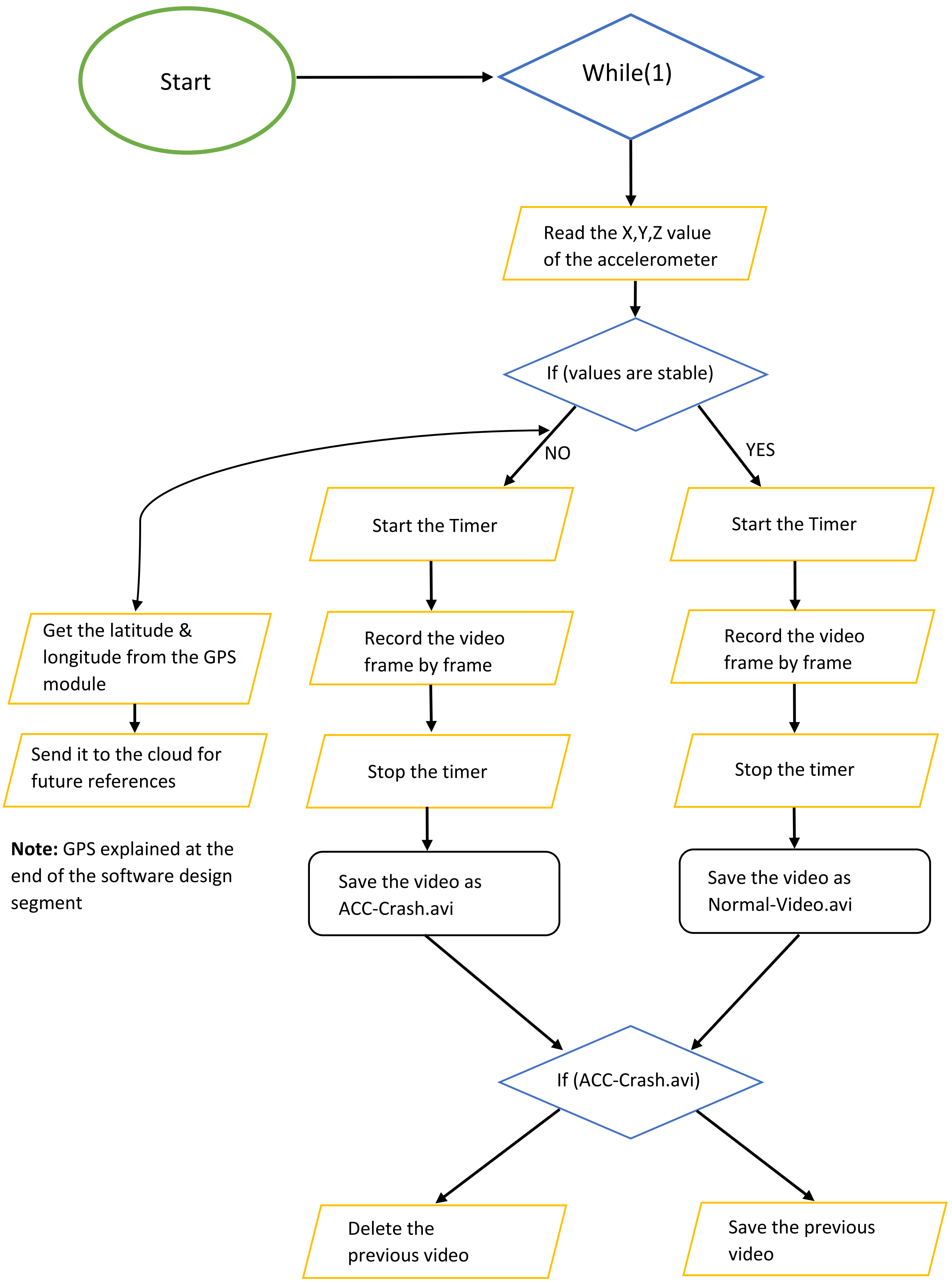
  Cstdio.h Is same as stdio.h in c language, also 'stdio' stands for standard input/output. This library is used to access physical devices like keyboard, printers etc using streams.

- **#include <time.h>**
  The **time.h** header file is used to get date and time information and then this information can be used to manipulate it.

- **#include <cstdlib>**
  This header defines several general purpose functions, including dynamic memory management, random number generation, communication with the environment, integer arithmetics, searching, sorting and converting.

- **#include<fstream>**
  This function is used to open a file, provided a filestream object is already created. We can use ofstream for writing and ifstream for reading from a file.

Flowchart


Start
While(1)
Read the X,Y,Z value of the accelerometer
If (values are stable)
NO
YES
Start the Timer
Record the video frame by frame
Stop the timer
Save the video as ACC-Crash.avi
Start the Timer
Record the video frame by frame
Stop the timer
Save the video as Normal-Video.avi
Get the latitude & longitude from the GPS module
Send it to the cloud for future references
If (ACC-Crash.avi)
Delete the previous video
Save the previous video

Note: GPS explained at the end of the software design segment

Fig 6 Software design flowchart

Some insights of the C++ code:

We will be using the predefined class from Open-CV Library

- To open a video camera:

```
// To Open the video camera-1 and camera-2
VideoCapture cap (0); //cam-1
VideoCapture cap (1); // cam-2
```

- To get frame Size:

```
//get the width of frames of the video
int frame_width = static_cast < int >(cap. get (CAP_PROP_FRAME_WIDTH));

//get the height of frames of the video
int frame_height = static_cast < int >(cap. get (CAP_PROP_FRAME_HEIGHT));
Size frame_size(frame_width, frame_height); //Frame size
int frames_per_second = 10; //FPS
```

- We took a reference of the 'VideoWriter' class mentioned in the **Reference(2)**. This class can be used by adding the following library:

```
#include <opencv2/highgui.hpp>
```

- //Create and initialize the VideoWriter object

```
oVideoWriter(recordfile.avi, VideoWriter :: fourcc('M', 'J', 'P', 'G'),
            frames_per_second, frame_size ,true );
```

  1. The 1st parameter here is the location where u want to store the file .
  2. 2nd parameter is to specify the 4 byte code used for video codec i.e fourcc ('M' , 'J' , 'P' , 'G') So here we are using **motion JPG.**
  3. The 3rd parameter is FPS.
  4. The 4th parameter is to pass the frame size.
  5. The 5th parameter is to pass colour=TRUE.

- Then to get the actual frame we have to follow the steps given below:

  1. `Mat frame; //read a frame`
```
bool isSuccess = cap . read (frame); // read a new frame from the video camera
```
  2. `oVideoWriter. write (frame); //write the frame to a file.`
  3. Do this in a loop to capture the frames.

- **GPS Sensor**

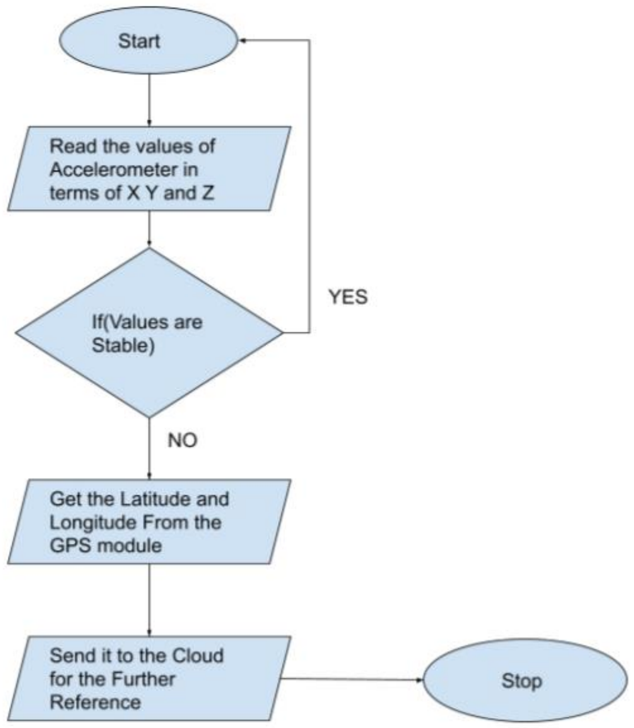

Start
Read the values of Accelerometer in terms of X Y and Z
If(Values are Stable)
YES
NO
Get the Latitude and Longitude From the GPS module
Send it to the Cloud for the Further Reference
Stop

Fig 7 GPS sensor Flow Chart

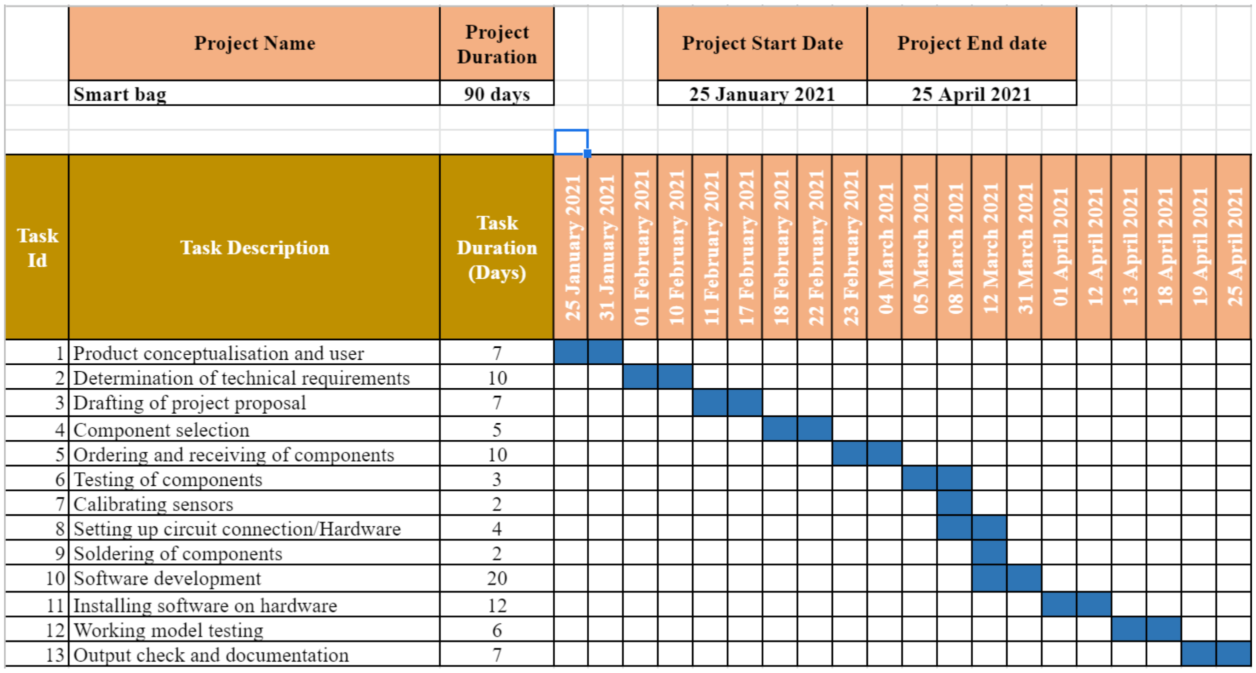
This flowchart illustrates that the values of the GPS sensor will be saved when there is a sudden spike or change in the values provided by the accelerometer, thus saving the latitude and longitude of the owner which can be directly inserted in google maps to get the current location of the user. Also there will be a log file provided for all such instances.

Code Snippet for the GPS sensor (link on the reference page):

```
#include "Navio/Ublox.h"      //in this header file the SPI communication is defined inside the ublox
                              //class that features a separate scanner and parser

#include "Navio/Util.h"       //this header file is used to convert PPM(Pulse position modulation) to
                              //PWM(pulse width modulation) decoder with Navio

std::vector<double> data;     //this vector stores data after the gps.decodeSinglemessage() has been
                              //decoded

Ublox gps;                    //creates an object for the Ublox class


int main (int argc, char *argv[])

{

    If(gps.testconnection()) // this is used to test the connection of the gps

    {

            printf(“GPS is now connected.\n”);

            While(true)

            {

                    if (gps.decodeSingleMessage(Ublox::NAV_POSLLH, data) == 1)

                    //checks if the message was decoded or not

                    {

                            printf("Longitude: %lf \t", data[1]); //prints the longitude
                            printf("Latitude: %lf\n", data[2]);  //prints the latitude

                    }

                    else

                    {

                            Printf(“\n Error: The message was not decoded successfully.\n”)

                    }

    }

return 0;

}
```

GANTT Chart

https://drive.google.com/file/d/1NZivqGIU7Ac0CHu0vOxLWNkGYCGnoDJB/view



Project Name	Project Duration	Project Start Date	Project End date
Smart bag	90 days	25 January 2021	25 April 2021

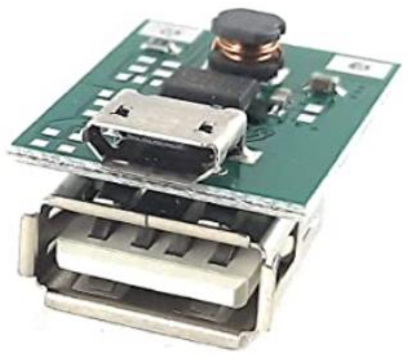
Task Id	Task Description	Task Duration (Days)	25 January 2021	31 January 2021	01 February 2021	10 February 2021	11 February 2021	17 February 2021	18 February 2021	22 February 2021	23 February 2021	04 March 2021	05 March 2021	08 March 2021	12 March 2021	31 March 2021	01 April 2021	12 April 2021	13 April 2021	18 April 2021	19 April 2021	25 April 2021
1	Product conceptualisation and user	7	■	■																		
2	Determination of technical requirements	10			■	■																
3	Drafting of project proposal	7					■	■														
4	Component selection	5							■	■												
5	Ordering and receiving of components	10									■	■										
6	Testing of components	3											■	■								
7	Calibrating sensors	2												■								
8	Setting up circuit connection/Hardware	4												■	■							
9	Soldering of components	2													■							
10	Software development	20													■	■						
11	Installing software on hardware	12															■	■				
12	Working model testing	6																	■	■		
13	Output check and documentation	7																			■	■

Fig 8 GANTT Chart

Appendix

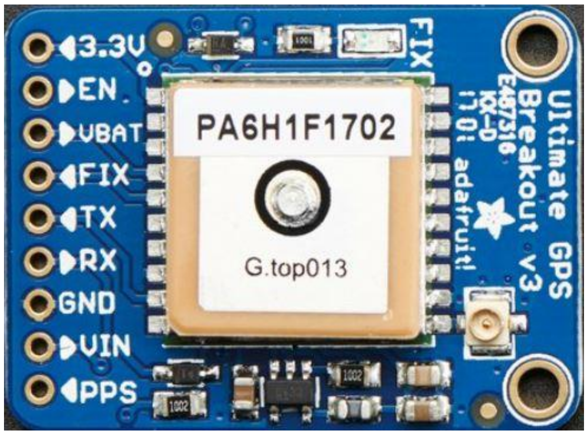
- Links for all the components required to make this project

1. https://www.dnatechindia.com/5-volt-1-ampere-usb-power-bank-module.html



2. https://robu.in/product/1-4-cmos-640x480-usb-camera-with-collapsible-cable-for-raspberry-pi-3/?gclid=CjwKCAiAmrOBBhA0EiwArn3mfGs4fwdKKER-qR1KJzH292uHy-25muqEx3AAm81pY0BQJ4qeNl2SjRoCe0YQAvD_BwE

3. https://www.adafruit.com/product/746



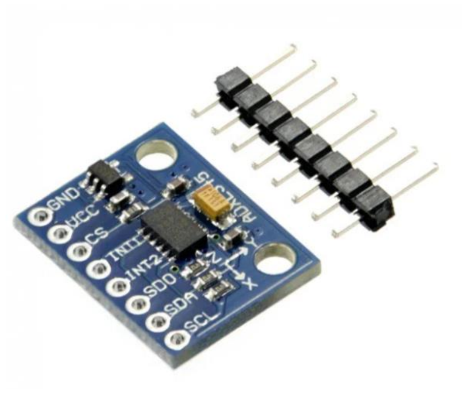
4. https://www.amazon.in/GI-Bicycle-Dynamo-Generator-12V/dp/B01CTBILWG/ref=sr_1_1?dchild=1&keywords=dynamo&qid=1613566381&sr=8-1

5. https://www.ineltro.ch/media/downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf

6. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

7. https://www.pololu.com/product/2119

8. https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf



Reference

- Ahmed Elkhatat. "Solar power V dynamo circuit", issued by Qatar University, Feb 2019 https://www.researchgate.net/figure/Solar-power-circuit-V-DYNAMO-CIRCUIT-A-dynamo-generates-a-sinusoidal-waveform-The_fig3_332143178

- https://docs.opencv.org/3.4/dd/d9e/classcv_1_1VideoWriter.html

- Patricio Gonzalezvivo. "Make your own GPS device with Tangram-es and RaspberryPi" https://github.com/tangrams/PI-GPS/blob/master/src/gps.cpp

- https://www.example-code.com/cpp/sns_publish_send_message.asp

- Shouji Usuda: "Introduction of circuit design for Lithium-Ion Battery", issued by Nikkan Kogyo Shinbun, Ltd., 2011.

- Shouji Usuda: "Introduction for Easy-to-Understand Utilization of Power Supply Circuits," issued by Nikkan Kogyo Shimbun, Ltd., 2009.

- Shouji Usuda, Takashi Matsui and Yuuki Mizobata: "Experiment for charging of Lithium-Ion Battery by Bicycle Dynamo", issued by Ohmsha, SHINDENKI, August and September 2013.

- https://www.programiz.com/cpp-programming/library-function/cstdio/remove

- https://matrix-io.github.io/matrix-documentation/matrix-hal/examples/imu/

- https://github.com/emlid/Navio/blob/master/C%2B%2B/Examples/PPM-decoder/PPM.cpp