Cover page….

# Abstract

The problem of relaying battlefield information is at the heart of all modern warfare, getting crucial, reliable data from one area to another, or from the battlefield back to the chain of command as fast as possible.

To help mitigate the problem, we developed a small, lightweight, gun mountable sensor array with wireless capabilities and sufficient battery life to record and transmit vital information off the battlefield. In addition, an off-site server will process and display the information, and could be used to alert necessary personnel if need be. The system could also play a role off the battlefield, simplifying the tracking of weapons, automatically calling for backup in sudden life-threatening situations, or to assist training exercises.

# תקציר

בעיית העברת מידע אמין  משדה הקרב היא בעיה מרכזית בשדה הקרב המודרני, העברת מידע אמין וחיוני מאזור קרבות אחד למשנהו, או משדה הקרב חזרה לשרשרת הפיקוד במהירות האפשרית. כדי לנסות ולמזער את הבעיה פיתחנו חיישן קטן מימדים וקל משקל להתקנה על הנשק, בעל יכולות אלחוטיות וחיי סוללה מספקים על מנת להקליט ולשדר מידע חיוני משדה הקרב. בנוסף לכך, שרת הנמצא מחוץ לאזור הקרבות יעבד ויציג את המידע המתקבל ויכול לשמש להתרעת כוחות שונים במידת הצורך. בנוסף לכך יכולה מערכת זו לבוא לידי שימוש גם מחוץ לשדה הקרב, לפשט את המעקב אחרי נשקים, הזעקת גיבוי באופן אוטומטי במצבים של סיכון חיים, או לסייע באימונים.

# Table of Contents

[Abstract 2](#_Toc469165165)

[תקציר 2](#_Toc469165166)

[Table of Contents 3](#_Toc469165167)

[Table of figures 4](#_Toc469165168)

[Table of symbols and abbreviations 5](#_Toc469165169)

[1 Introduction 6](#_Toc469165170)

[1.1 System purpose 6](#_Toc469165171)

[1.2 Physical Unit 6](#_Toc469165172)

[1.3 Server 7](#_Toc469165173)

[2 Specifications 8](#_Toc469165174)

[2.1 Physical Specifications 8](#_Toc469165175)

[2.2 Electrical specifications 8](#_Toc469165176)

[2.3 Server specifications 9](#_Toc469165177)

[3 Workflow 10](#_Toc469165178)

[3.1 Hardware 10](#_Toc469165179)

[3.1.1 Component choices 10](#_Toc469165180)

[3.1.2 Electrical design (schematic) 11](#_Toc469165181)

[3.1.3 PCB design (layout) 12](#_Toc469165182)

[3.1.4 RF and antenna design 12](#_Toc469165183)

[3.1.5 Mechanical design 13](#_Toc469165184)

[3.2 Firmware 14](#_Toc469165185)

[3.2.1 Orientation algorithm 14](#_Toc469165186)

[3.2.2 Magnetometer Calibration 14](#_Toc469165187)

[3.2.3 Issues with I2C, Wi-Fi startup and z axis magnetometer data 15](#_Toc469165188)

[3.2.4 Get request 16](#_Toc469165189)

[4 Software (server) 17](#_Toc469165190)

[4.1 Architecture 17](#_Toc469165191)

[4.2 AWS 17](#_Toc469165192)

[4.3 Ubuntu 17](#_Toc469165193)

[4.4 Application 17](#_Toc469165194)

[4.5 sqlite3 18](#_Toc469165195)

[4.6 Apache\* 18](#_Toc469165196)

[4.7 django 18](#_Toc469165197)

[4.8 google maps api 18](#_Toc469165198)

[4.9 xml gateway 19](#_Toc469165199)

[4.10 server issues at deployment 19](#_Toc469165200)

[4.11 19](#_Toc469165201)

[5 Extensions and additions 20](#_Toc469165202)

[6 Source code 20](#_Toc469165203)

[7 Appendix 21](#_Toc469165204)

[7.1 Appendix A: Full electrical schematics 21](#_Toc469165205)

[7.2 Appendix B: PCB 24](#_Toc469165206)

[7.3 Appendix C: Full mechanical drawings 25](#_Toc469165207)

# Table of figures

[Figure 1: Picatinny rail on a paintball pistol 6](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165208)

[Figure 2: Device physical dimensions 8](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165209)

[Figure 3: Hardware block diagram 10](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165210)

[Figure 4: CC3200 module 10](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165211)

[Figure 5: CC3200 development kit 10](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165212)

[Figure 6: ADA18182 ceramic patch antenna 11](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165213)

[Figure 7: The device PCB in EAGLE 12](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165214)

[Figure 8:Wi-Fi antenna close-up 12](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165215)

[Figure 9: GPS test antenna 13](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165216)

[Figure 10: GPS antenna close-up 13](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165217)

[Figure 11: Device enclosure 13](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165218)

[Figure 12: Firmware flowchart 14](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165219)

[Figure 13: Electrical schematic page 1 of 3 (main MCU and GPS) 21](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165220)

[Figure 14: Electrical schematic page 2 of 3 (sensors and SD card) 22](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165221)

[Figure 15: Electrical schematic page 3 of 3 (LED indicators) 23](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165222)

[Figure 16: Full PCB for manufacturing 24](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165223)

[Figure 17: Mechanical diagrams 25](file:///C:\Users\Rafi\Google%20Drive\Projects\IoTsightEmbedded\WorkspaceV2\IoTsightV3\Project%20Report.docx#_Toc469165224)

This needs to be updated manually after adding images **with captions**. Under references->Update table: Ctrl+A F9

# Table of symbols and abbreviations

GPS – Global Positioning System

GLONASS – Global Navigation Satellite System (Russian GPS system)

Li-ion – Lithium ion battery

Wi-Fi – Wireless fidelity (wireless network connection)

PCB – Printed Circuit Board

RF – Radio Frequency

MCU – Microcontroller unit

SPI – Serial peripheral interface

IC – Integrated circuit

I2C – Inter-Integrated circuit communication protocol

SMD – surface mount device

LGA – Land grid array

BGA – Ball grid array

I/O – Input and output

MEMS – Micro-electrical-mechanical system

VCC – Supply voltage

GND – Ground (voltage return)

PLA – Polylactic acid (a plastic)

TI – Texas Instruments

ATM – Automatic teller machine

AWS – amazon web services

ec2 – elastic cloud computing

HTTP – hypertext transfer protocol

SSH – secure shell

OS – operating system

# Introduction

## System purpose

The system consists of two separate sections, a physical unit, distributed to each soldier and mounted to the weapon, and a server to collect and process the data.

The unit is always running, but remains dormant most of the time to conserve battery life. When a soldier fires his weapon, the weapon experiences a sudden change in acceleration along its axis, known as ‘*recoil’*. The device’s accelerometer captures this sudden change in acceleration and causes an interrupt to wake up the processor. The processor then takes a snapshot of all current sensor data and preprocesses it to extract the devices orientation, heading, altitude and position, and the time of the occurrence. This information, along with the devices unique ID number, is sent via Wi-Fi to the main server, which displays the information on an interactive map. Depending on requirements, the server could be setup for multiple purposes. In one instance, it could be used to take multiple shots’ information to predict where enemies might be and automatically notify other soldiers in the area that assistance is necessary. The server could also be setup as part of training exercises, and would be used to analyze how many shots each participant fired and how accurate each shot was. The device could also be mounted to steel plate targets to detect when a ‘hit’ occurred.

## Physical Unit

The physical unit is a small plastic enclosure not much bigger than a can of soda and weighing ~???g. It can be mounted to nearly any weapon via a standard *‘Picatinny rail’* mounts which are commonly used for mounting accessories to a weapon such as scope sights, lasers, and flashlights (see Figure 1). These are present on nearly all military issue weapons ranging from pistols to large machine guns.

Figure 1: Picatinny rail on a paintball pistol

The unit is a sensor array consisting of the following:

* Accelerometer
* Gyroscope
* Magnetometer
* Barometer
* GPS receiver

Together, they give the ability to read out orientation, altitude, and location in the form of roll, pitch and yaw angles, GPS coordinates and ambient air pressure. The accelerometer also gives the ability to send out an interrupt on sudden changes in acceleration along any axis, which is used for the shot detection function.

The plastic enclosure houses the battery and main PCB, and can be sealed against water and dust while still providing a pressure relief port to take accurate air pressure measurements.

The battery is a standard sized Li-ion cell, which can be swapped out for charging. The idle battery life is just over 21 days, and each shot detection and transmission taking up to 30 seconds expends the battery life equivalent of up to 6 minutes of idle time.

## Server

The server is a virtual machine provided by amazon as part of AWS. In the service, it is often called an instance of ec2. That means that the consumer is provided with a computation power and internet connectivity without having to buy hardware, Thus enabling a low cost server farm to be used while enjoying robustness huge scalability and few locations redundancy for mission critical solutions.

The virtual machine is configured to run Ubuntu a distribution of linux that is used both in commercial and private settings. The http server is apache, it is the means by which the HTTP requests are served so that users could connect to the webpage. The webpage itself has a dynamic rendering that creates an html view according to the queried parameters. The language used to create the dynamic interaction is python, namely the django library of the language. Data from devices is stored in sqlite3. Map rendering is made with a google maps API in javascript. There is also an xml static page served for the devices to read periodicaly to change their state to close a control loop.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| xml | Images | Css | javascript | Html |
| Static files | | Django app | | |
| Apache web server | | | sqlite3 | |
| Ubuntu linux | | | | |
| AWS amazon cloud compute | | | | |

# Specifications

Figure 2: Device physical dimensions

## Physical Specifications

|  |  |  |
| --- | --- | --- |
| PARAMETER |  | UNIT |
| Length | 110 | mm |
| Width | 36 | mm |
| Height | 35 | mm |
| Weight | 120 | g |
| Operational Temperature range | -40 to +85 |  |

## Electrical specifications

|  |  |  |
| --- | --- | --- |
| PARAMETER |  | UNIT |
| Battery Type | 18650 Li-ion | - |
| Battery Voltage | 3.5 to 4.4 | V |
| Battery Capacity | 2200 | mAh |
| Idle current | 4.3 | mA |
| Idle battery life | 21.3 | day |
| Transmitting current | 54 | mA |
| Continuous transmission battery life | 40 | hour |
| Peak current | 250 | mA |
| Maximum time to transmission | 30 | second |
| Roll accuracy | ±1.0 | degree |
| Pitch accuracy | ±1.0 | degree |
| Heading accuracy | ±2.0 | degree |
| Position accuracy | 10 | m radius |

## Server specifications

The server is a t2.micro instance with the following specs:

|  |  |
| --- | --- |
| Vcpus | 1 |
| RAM | 1Gb |
| HDD | 8Gb |
| OS | ubuntu |

We are using apache2. Python 2.7 . Django 1.9 . slqite3.

# Workflow

## Hardware

MCU

[CC3200]

Accelerometer

[ADXL345]

Wi-Fi

[CC3200]

Magnetometer

[HMC5883L]

Gyroscope

[ITG3200]

GPS

[NEO-6M]

SD card

Support Hardware

Figure 3: Hardware block diagram



### Component choices

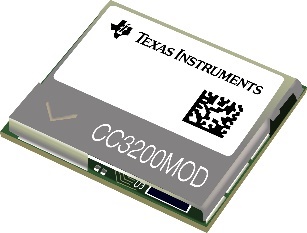
The first major component is the MCU, which serves as the processor for the hardware. For this purpose, the CC3200 chipset from TI was chosen. This was because it has built-in Wi-Fi capabilities which require no external components aside from an antenna. This greatly reduces the amount of design work that falls into the RF spectrum which requires a great deal of additional experience. The CC3200 is available in two forms, the first, a SMD chip, which needs various support components such as decoupling capacitors, pullup resistors, crystal resonators, and an SPI flash memory chip. The second form is a module which has all the above components build into a SMD LGA package of 17.5mm x 20.5mm. Due to cost and ease of use, the chip form was used during development while the module was used on the final PCB. The MCU also feature a hibernate mode where current consumption drops down to 4uA while still allowing for external interrupts to ‘wake up’ the MCU. This choice also eliminates the need for an external wireless chipset which also conserves power since readily available Wi-Fi modules such as the ESP8266 use more power even when in hibernation modes.

Figure 4: CC3200 module

Figure 5: CC3200 development kit

For an accelerometer, the ADXL345 was chosen for its low power consumption, easy to solder package (LGA as opposed to BGA) and its dual configurable interrupt outputs which were used for the ‘shot detection’ feature. The interrupts are configurable to only capture motion along a single axis and to fire only when the acceleration exceeds a configurable magnitude. The data can then be read out over an I2C interface.

The HMC5883L was chosen as the magnetometer the device for the same reasons that it’s very popular in the hobbyist community. It has low power consumption, simple I2C interface, and easy to solder SMD package.

A gyroscope chip, the ITG3200 was also included in the design, although it isn’t necessary for the intended application of the device and the component does not need to be inserted during manufacturing. Its pads however are present on the PCB, which enables various other applications and uses for the device as will be mentioned later.

Because of time constrains and availability, the NEO-6M was chosen as the GPS module for the device, since it was already on hand.

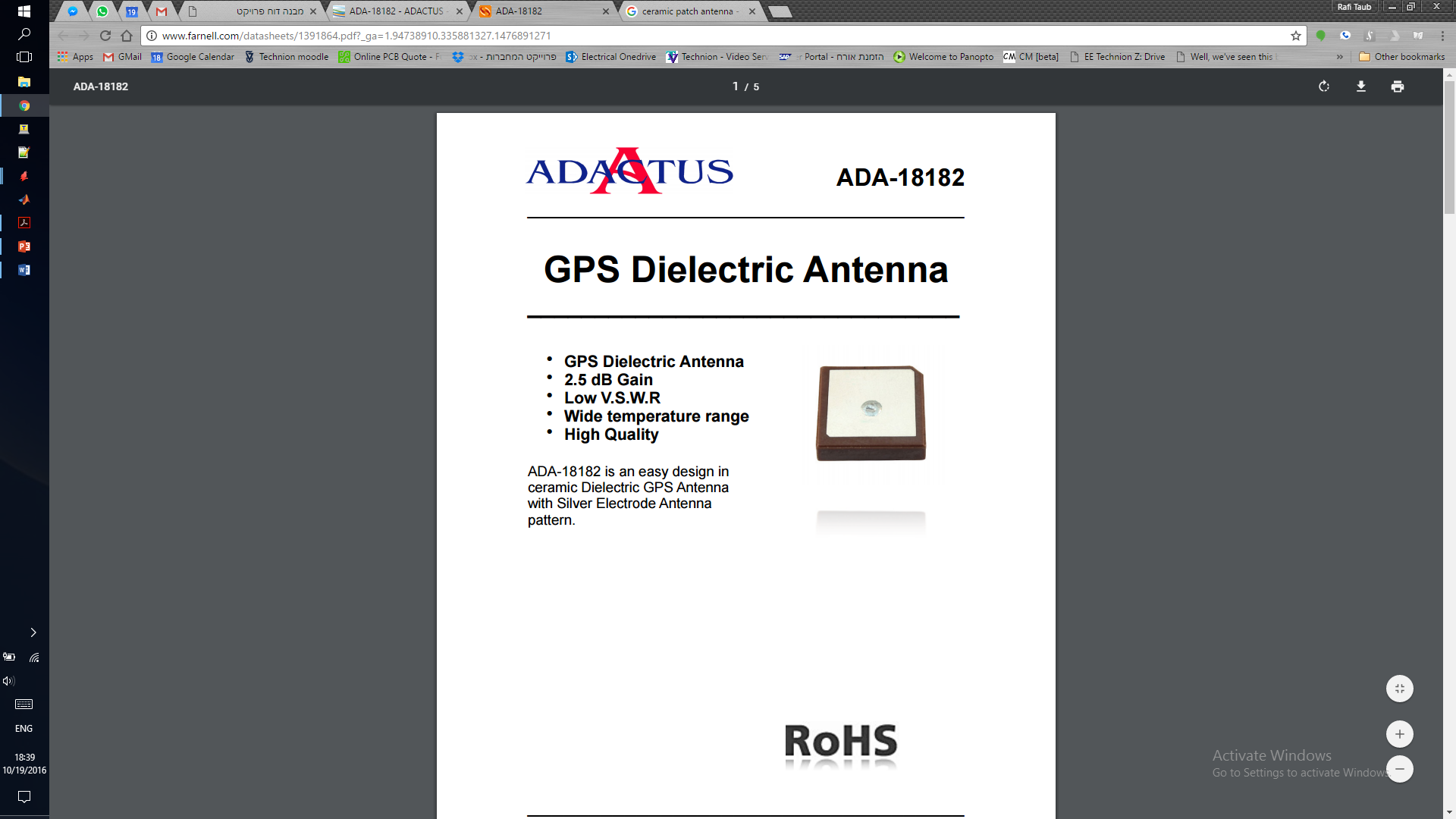
 Since the unit will be oriented in a specific direction and the GPS satellites only ever being above any point on earth, the antenna for the GPS could be a directional antenna, which would provide gain to the GPS signal, and in turn, greater accuracy in position measurements. Since the output impedance of the CC3200 is 50Ω, it was easiest to find an antenna which also had an impedance of 50Ω. In addition, the device required a rather thin profile which lead to the choice of a ceramic patch antenna. The ADA18182 was chosen for this since it is easy to solder and for its small size of 18mm x 18mm and 2mm height.

Figure 6: ADA18182 ceramic patch antenna

Again, due to time constraints and availability the MPL115A was chosen as the barometer for the device to give more accurate altitude measurements than the GPS system can provide. The ‘A2’ version of the chip was used since it interfaces via I2C as the other sensors do and therefor doesn’t require additional I/O pins.

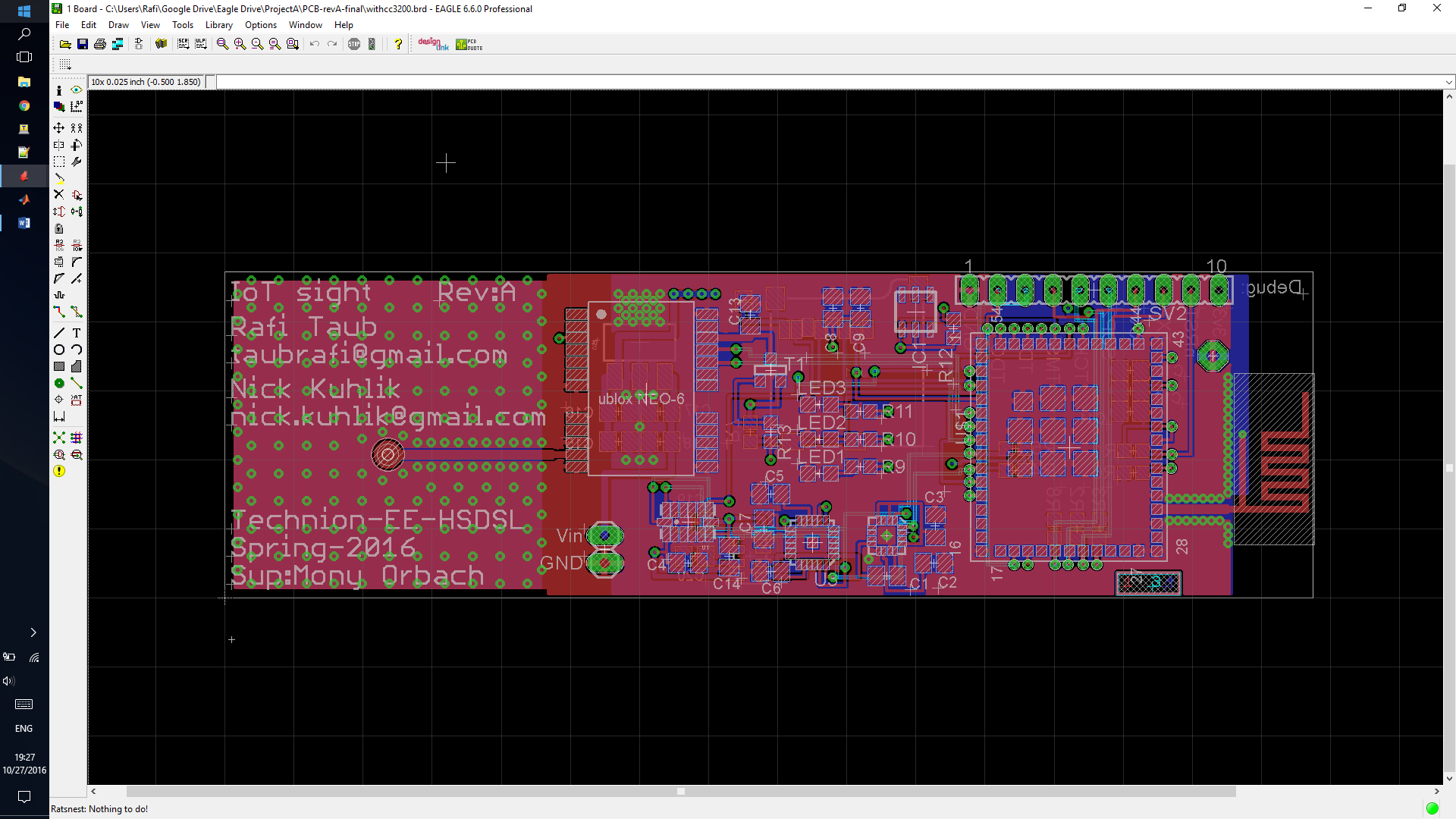
The last major component is the DS2401+ which is a silicon serial number. It provides a unique 64-bit identification number hard coded into the device (which gives 18 billion-billion unique ID numbers). Since no two chips have the same number this enables individual tracking of every device and the ID number can be matched with every issued weapon and individual using it. This device also has a *‘OneWire’* interface which only requires one additional I/O pin. (The chip only has two pins, ground, and data. There is no positive voltage line, instead the chip charges an internal capacitor from the pullup resistor on the data line and uses that stored charge to continue operating when the data line is low.)

In addition to the above, various other components such as capacitors, resistors, LEDs, a transistor, voltage regulator and a microSD card socket were included in the design. The microSD card connector was included for future expandability and as with the gyro, does not play a role in this application.

### Electrical design (schematic)

The schematic design is rather straightforward, all the integrated circuits were connected with the necessary external components as per the manufacturer’s specifications. In addition, decoupling capacitors were placed near each IC. A removable solder jumper was added between the DAT1 pin of the SD card and GND. This enabled the switching of SD card protocols during the debug stage. In the end, the *‘SPI mode’* of the SD card was chosen, meaning the jumper can be left as is (connected to GND).

To conserve power, a PNP transistor was connected between VCC and the GPS receiver’s main power input pin while the backup battery input was left connected to VCC. This enables the shutdown of the GPS receiver to be controlled by the microcontroller while keeping the GPS data in memory. This ensures a faster ‘*hot fix*’ when trying to regain the location data from the GPS system.



### PCB design (layout)

The PCB was designed using EAGLE which is a combined schematic and layout software which is supported by most PCB manufacturers.

The PCB has four layers which are (from top to bottom): Signal layer with ground plane flood, supply voltage plane, signal layer, signal layer with ground plane flood. Although the system performs as expected the two internal layers should have been voltage supply and ground, with all signals kept on the outer layers. This would increase the distributed capacitance across the entire board keeping noise to a minimum. It would also keep a solid ground plane below signal traces keeping impedance constant and minimizing wave propagation artifacts caused by reflections. However, since all signals (excluding the RF areas which have no internal layers) in the device are rather slow (less than 1MHz) we did not experience any issues.

Figure 7: The device PCB in EAGLE

Three LED indicators were added at this stage (as can be seen in the center) and were connected to three spare I/O pins on the CC3200. They were meant as aids during debugging and are not mean to serve any function in the final version. That said, as with the gyro and SD card could be used in possible future applications for the device.

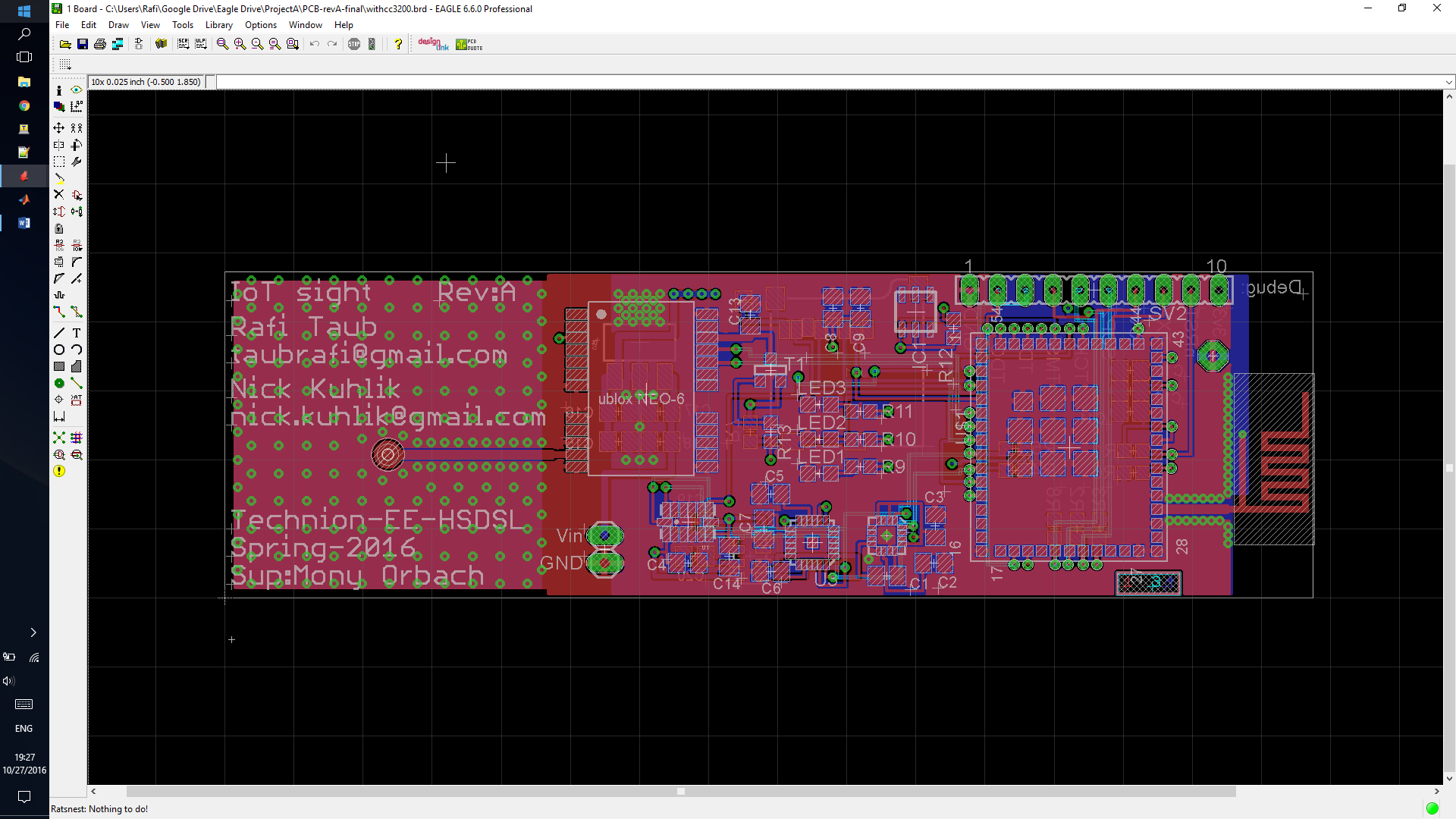
The three MEMS sensors were placed so that all their axes are aligned to keep data processing simple, and a debug header is also present to enable programming of the CPU without the need for desoldering. It also provides a serial terminal for printing debug messages and data (again, for use during debugging). Two battery voltage input pins are also present.

Lastly, ‘*open* *vias*’ we placed on all signal pins to enable probing, and cutting-resoldering of traces as needed when working on hardware. To aid in probing, one more large ‘*via*’ was added and connected to the ground plane as a reference for oscilloscopes and logic analyzers

### RF and antenna design

The last two key components on the PCB require special attention. The Wi-Fi antenna was designed as part of the board while the GPS antenna is a ceramic patch antenna. Both require specific dimensions in both the traces that connect them and the ground plane below them.

Figure 8:Wi-Fi antenna close-up

The design for the Wi-Fi antenna was taken from an application note[[1]](#footnote-2) from Cypress semiconductors on Wi-Fi and Bluetooth antenna design. The PCB antenna choice was made to since it doesn’t require a matching network to convert to a 50Ω trace. It also provides reasonable directionality, sending the signal out along the plane of the PCB instead of normal to the PCB, which would be directed into the battery. This effectively adds gain to the signal being sent back to the router which is receiving the signal adding range to the unit’s operating radius. In addition, the trace connecting the antenna to the CC3200 is of a specific width with the ground plane coming to a specific distance away from the trace. This trace setup is known as a ‘*coplanar wave guide*’ and is commonly used for signals in the few up to tens of GHz range.

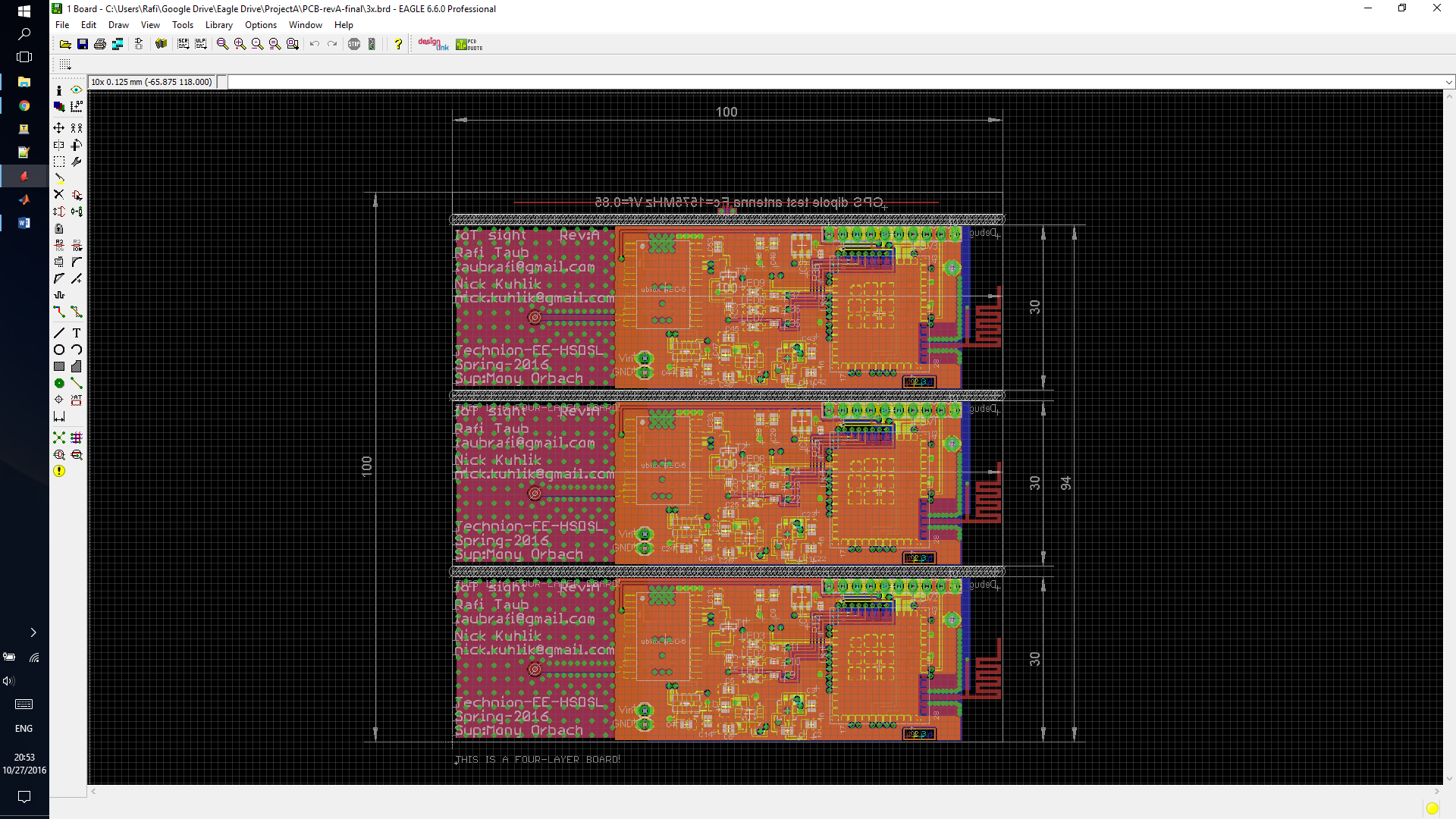
For the GPS antenna, the dimensions for the ground plane were taken from the device’s datasheet. The ground plane gives then antenna directionality and therefore gain (of about +5dB) in the upward direction normal to the plane of the PCB. This improves positional accuracy, likelihood of GPS lock, and time to lock for the GPS receiver. The trace connecting the GPS antenna to the receiver is also a ‘*coplanar wave guide*’ and has the same dimensions as the Wi-Fi trace since the GPS signals are in the same band being ~1.5GHz.

Figure 9: GPS test antenna

Figure 10: GPS antenna close-up

During development not all the components had arrived, so for testing a dipole antenna for the GPS was desinged and added as part of the PCB sent to manufacturing. Although it does work, the ciramic antenna is a better choice for its directionality and low losses.

### Mechanical design

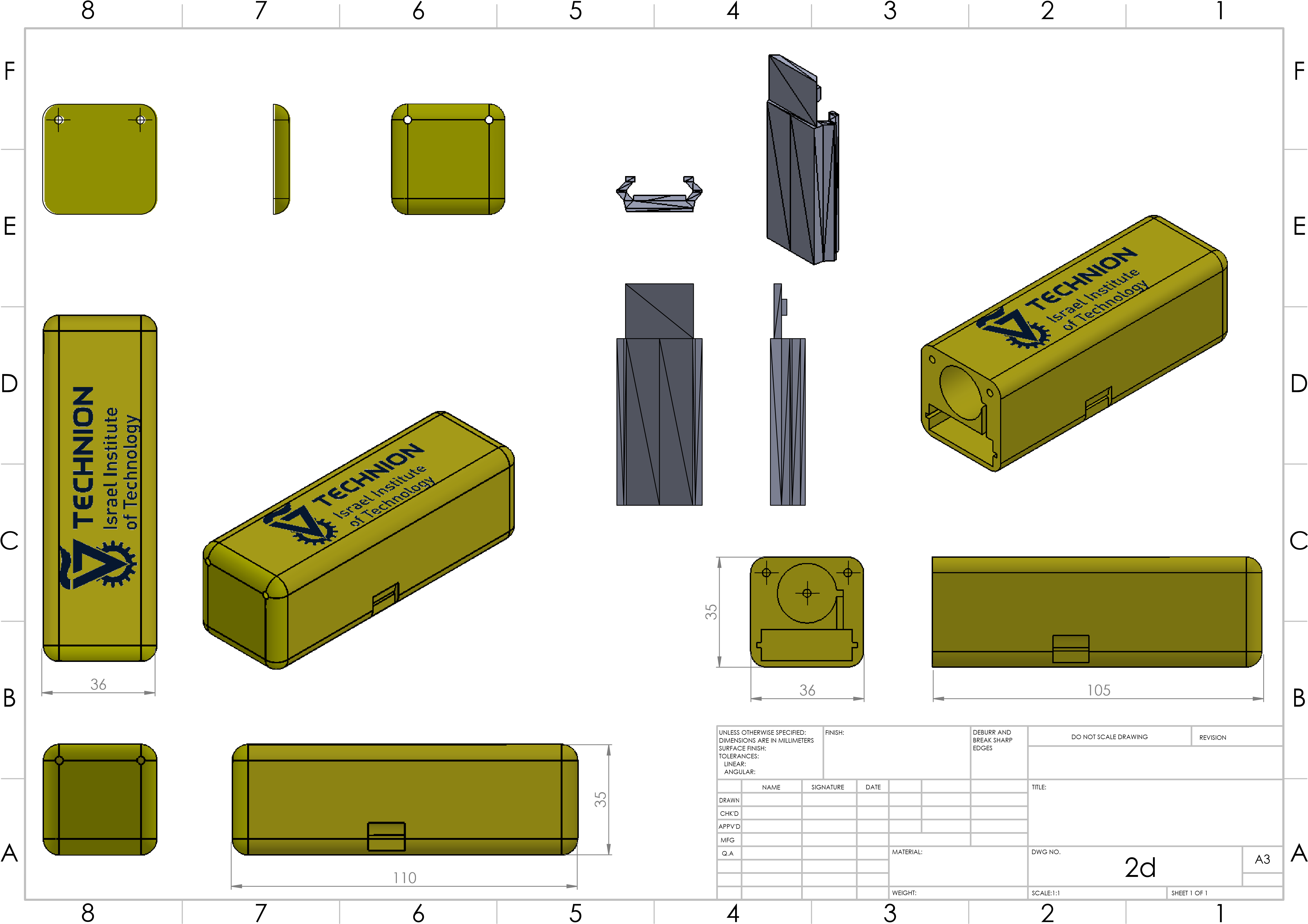
The devices prototype enclosure is a 3D-printed PLA which was designed in SolidWorks. It features a slot for the PCB and a compartment for the battery. The battery was chosen to be an 18650 Li-ion rechargeable battery with a nominal voltage of 3.7v-4.2v and a capacity of 2200mAh. The slot along the side of the battery cutout is for the wire running to one of the terminals. Both wires run down into the lower compartment to the main PCB. In addition, a slot on the side of the enclosure gives access to the microSD card and lines up with the socket on the PCB. The alignment was maintained by taking measurements from the PCB and bringing them into SolidWorks to make the required cuts. A rubber cap could be designed to cover this hole to provide weatherproofing while still allowing changes in air pressure through to the internal barometer. The cover attaches to the case by two #4-40 screws which rotate freely in the cover and screw into the slightly undersized hole in the main body of the enclosure to ensure a solid connection. The last part of the enclosure not show in this diagram is a mount to connect the device to the Picatinny rail system for mounting on a weapon.

Figure 11: Device enclosure

## Firmware

Figure 12: Firmware flowchart

Calibration routines

Power on sensors

Read sensor data

Calculate altitude & attitude

Get position from GPS

Power off sensors

Accelerometer triggered interrupt

Send data to server

### Orientation algorithm

Once a shot is fired the accelerometer triggers an interrupt on the MCU which prompts it to turn on the GPS module and proceed to take readings from the various sensors. From the corrected magnetometer data and the accelerometer data .

First the roll angle is calculated as:

(Note the function provides the inverse of where the signs of and are maintained, giving a result on instead of .)

Next, the pitch angle is calculated as:

(Not since the pitch angle should lie on .)

Lastly, the yaw (heading from magnetic north) angle is calculated as:

Together these three angles give a complete description of how the device (and therefore the shooter) is oriented at the time of the occurrence.

### Magnetometer Calibration

As mentioned, the magnetometer data is used to calculate the orientation angles of the device and the equations rely on the magnetometer data however, this data is not usable directly as magnetic offsets are a major problem with magnetometers. Any piece of ferrous metal (steel, iron, etc.) all have slight magnetizations and will offset the magnetometer readings.

If each data reading from the magnetometer were to be plotted on a graph while the device was rotated to every possible orientation, the points would form an ellipsoid with axis lengths of in the and directions respectively and the center of the ellipsoid would be at . Ideally, the center of the ellipsoid would be at (0, 0, 0) and the axis lengths would all be equal (1, 1, 1). However, small magnetized objects will both shift the center of the ellipsoid along the axis the magnetic material points, and lengthen or contract the respective axis depending on the alignment of the material. This could be potentially catastrophic as a flat north-facing orientation, which would ideally give a magnetometer reading of (1,0,0) would be could be shifted and scaled to give any possible reading (a reading of (-1,0,0) would mean south). To counter this, a simple calibration algorithm was added to the software, which works as follows:

* Six static values are stored which represent the lowest and highest recorded values (the ‘extreme values’) on each of the magnetometer’s axis respectively.
* On each reading, the current value is compared with the stored extreme values, and the extreme values are updated as needed.
* We then define: , . Since an ellipsoid is symmetric with regards to any of its axes, its center must be halfway between its two ends. Thus, we know where the center of this offset ellipsoid lies.
* Next, we define: which represent the lengths of each of the axes of the offset ellipsoid.
* Lastly, This shifts the ellipsoid back to (0, 0, 0) and brings then axis lengths back to (1, 1, 1).

This algorithm is essential in maintaining good orientation data accuracy. One of its main benefits over pre-calibrating each device is that this algorithm is a very simplistic ‘machine learning’ algorithm. To initially calibrate the unit, one simply needs to rotate the unit a few times in each direction, and once mounted to the weapon the offsets would change since the weapon itself contain many metal pieces. Even the bullets or magazines could be magnetic and since they are swapped out regularly, it would mean any possible pre-calibration would be useless. But with this algorithm, one needs to just power cycle the device (by swapping the batteries) and rotate the device while attached to the weapon to recalculate the calibration values.

### Issues with I2C, Wi-Fi startup and z axis magnetometer data

During the development stage, several interesting issues were uncovered which are worth mentioning since their cause was not due to any development error on our part but rather on the manufacturers of some of the components we used. Two such issues were found in the CC3200 manufactures by Texas Instruments. The CC3200 was a relatively new device at the time when development started and as with many new devices, it had some issues that had not yet been found. The first of which was had to do with the assignment of I/O pins used for I2C communications. To save pins on the device package, the two I2C pins can be mapped to several physical pins on the device, but only two such physical pins can be assigned for this purpose. It turned out the device had a predetermined assignment for these pins which was not noted in the datasheet and the assignment had to be undone before reassignment could be performed. A few weeks after we found the issue, a relevant section was added to the CC3200 errata sheet on TI’s website. A similar issue was found in the startup code for the Wi-Fi communications on the CC3200 and was later corrected with a library update by TI.

The last notable issue was not a software bug but rather a hardware defect. As it turns out, the HMC5883L chips which we purchased for use in the early design stage were not manufactured by Honeywell, but instead were manufactured by some copy facility in china. These ‘copy chips’ were not as robust as the originals and the Z axis sensor head inside the chip would fail and only gave constant data. Once the problem was discovered, some genuine chips were ordered which worked flawlessly.

### Get request

The very last step in the data path for the device is the ‘Get request’ sent over the Wi-Fi network connection to the server. The request is formatted to have separate fields in a predefined protocol setup for this system and transmits: the current date and time, the devices latitude and longitude, the ambient air pressure (used in conjunction with weather servers to calculate altitude), the device’s orientation angles, and the device’s unique ID number.

# Software (server)

## Architecture

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| xml | Images | Css | javascript | Html |
| Static files | | Django app | | |
| Apache web server | | | sqlite3 | |
| Ubuntu linux | | | | |
| AWS amazon cloud compute | | | | |

The architecture selected for the server was guided by the principles of simplicity, scalability and reliance on free open source software.

By implementing those principles it is guaranteed that the project will be easy to maintain and easily passed to future generations of students. The

TODO:

here goes explanation about the idea to make a mono-core of code that uses python.

Google maps api addition. Since we didn’t want to roll out our own maps.

## AWS

The aws platform gives a readily made instance of a computer that can be easily replicated to adjust for demand. The servers are located In data-centers separate from each other thus granting fault tolerance in case of a local disaster that disables one of the locations. Also, it is easier to rent a server on the web and route all traffic to it instead of aquiring a pc to be used in the technion network and having to open up connections to the internet and managing the hardware. A fault that was critical to another project in the HSDL lab.

In conclusion, the aws platform is both convenient for both small scale deployments that don’t necessitate the purchase of additional hardware and large scale mission critical deployments.

## Ubuntu

Ubuntu is free open source linux distribution targeted both for the private and corporate use. The distribution is well supported and well documented with a vast community of developers using the OS.

The ubuntu os takes care of the lower levels of the OSI hierarchy registering the pc on the network, creating sockets and collecting ip packets into a tcp protocol.

Also, the ubuntu machine runs the application and saves the database giving useful tools to manage and edit both.

## Application

the application is constructed upon the interaction of a database, a collection of static resources that are not need to interact with the server in a dynamic way , and a python application that controls the database workflow, fetches static files, renders dynamic views for the user and interacts with the thin client implemented on the tracked unit.

## sqlite3

sqlite is a relational database used to create read update and delete records in a SQL way that makes a standardized way to work with data. This way was chosen because it is easily used, although it may be switched for any preferred database with little effort.

The application generates the needed data tables in accordance to the models that are described in the application code. The data is then easily saved and retrieved and can be accessed both from the python wrapper for sqlite3 and a mysql like command line tool.

The sqlite3 is pointed back to the hosting machine, but in future versions can be shared with several machines hiding behind the DMZ in the internal firewall. This configuration can be useful for decoupling database and apps for robustness and scalability.

## Apache\*

The apache web server manages connections on the http level, after a connection has been established by the linux os, the http web server must direct the user to the appropriate location and gather the data needed from the application to return.

The apache web server was chosen for its free as in free beer ideology and great swath of support.

## django

Django is a full stack web framework written in python that is used to control the layer 7 application interaction with server. The framework is a set of modules written in python that make prototyping websites faster and easier by taking care of the stuff that usually all web servers must preform and giving capabilities to write code that will do the “Buisness Logic” of the application.

The application is created in the mvc design pattern which separates the problem into 3 components.

The central component of MVC, the *model*, captures the behavior of the application in terms of its problem domain, independent of the user interface.

* The *model* directly manages the data, logic, and rules of the application.
* A *view* can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
* The third part, the *controller*, accepts input and converts it to commands for the model or view.

In our app, the model -represent the shots class and its methods.

The view – is the /shots page that prints the output.

The controller is the url parser that directs the user to the page requested (view) showing the data selected (model).

## google maps api

Using the javascript api, the google maps api is useful for visualising the received points of interest and bearings to the user that shall interpret them. It is possible to use javascript additional functionality and create algorithms for crossfire enemy detection based on time and location, friendly fire alarms and training situations in which part of the sensors are designated static targets. Also we can show different slicing of data, by time period or location specific and even create a “recording” of events that shall be useful in debriefing missions or training excercises.

## xml gateway

The xml gateway is a useful format to push data back into the tracking device’s memory. Since each unit is arbitrarily woken up and may experience connection loss or be given a dynamic ip, there has to be a way to update the settings that govern the devices. The best way is to point them to a known location available when connection to transmit data has occurred. The xml page contains the id and data settings that need to be parsed by each device. The xml page in the prototype is a dummy place holder and has no receive functionality on the devices (as of writing of this project report it was deemed useless for the means of this configuration). The django app can and should create a the xml file in accordance with the logic decided upon.

## server issues at deployment

After creating the application on a local machine, the migration process was to upload the files into an ec2 instance from the github repository.

On the local machine an Internet port serving only the localhost is generated once the app is run. Once the app is deployed a web server (like apache) has to commandeer access to the requested ip and pass them to the app.

In the development process a problem was encountered that only part of the webpage would load without additional styling. Thus giving a plain look to pages that were stylized before the deployment.

Since it was chosen to host the static files locally, the app had to look up the files on the hard disk and send them to the user to generate an HTML. The problem has been diagnosed as a permissions denied response when requesting those static files. A number of fixes to try and change file permissions were tried, changing locations, owners, privileges and such.

The problem was at last solved by a technical document describing the proper apache configuration to enable a static folder to become world available and thus hyperlinked to the HTML file, rendering it completely.

## 

The main takeaway from this problem was that although there is great support for apache, a problem with django which was still young at the time was not identified correctly due to not informative errors and then hard to fix since the django-apache stack was new at the time.Performance tests

# Extensions and additions

This system has many more possible applications that the original intended purpose. The physical unit is essentially just a Wi-Fi connected sensor module with GPS and datalogging which could be used for inventory or vehicle tracking, amateur rocket experiments, long term datalogging, earthquake detection and much more. In addition, the I2C and SPI interfaces, along with the remaining I/O pins could be used to connect further external sensors or as outputs to trigger alarms, lights, motors, or anything else. Combined with the small form factor this device could even serve as the heart of an autonomous drone and could in theory support a low-resolution camera and transmit back over Wi-Fi.

One addition that was left out of the project due to time constraints and practicality is encryption. It wouldn’t be very smart for every soldier to transmit his location over a Wi-Fi connection, since any laptop running Wireshark (a free network traffic analyzer tool) could simply read out the data and track down the soldiers. Therefore, a layer of encryption would be necessary to hid the data from anyone who could try to read it. This encryption would have to be implemented on both ends, the device, and the server, and must be done in such a way that even if a potential enemy gets hold of one of these devices, he still would not be able to break the encryption. Solutions like this are quite common in the software industry and it’s how activating a piece of software with a license key is simple and yet it’s impossible to somehow find out what possible license keys the software would treat as valid even if you have the software installed.

Another feature that could be added to avoid the above issue is a remote self-destruct, where the microcontroller could wipe its own flash memory and reset itself (doing nothing from then on unless reprogramed) if the PCB area of the enclosure is opened or by remote signal (a technique used in cred card readers and ATMs to prevent bank fraud).

# Source code

Device firmware and project files: <https://github.com/taubrafi/IoTsightV3>

Server source code: <https://github.com/sudo-Whateverman/IOTsights>

# Appendix

## Machine generated alternative text: GNDGNDGND GND GND GND GND GNDGNDGND GND GND GND GND GNDGND GNDGND 10k 2.7k 100k 100k GNDGNDGND VDD VDD VDD VDD 2.7k 2.7k VDD VDD GND GND GND GNDGND 4.7uf 0.1uF0.01uF GND GND GND 100k GND GND VDD DS2401P GND 2.7k VDD 0.1uF ADA18182 GND MIC2920A-3.3WS-TR VDD GND 4.7uf 0.1uF 4.7uf 0.1uF BS250 VDD 2.7k 27_GND P27 26_ANTSEL2 P26 25_ANTSEL1 P25 24_SOP1 P24 23_SOP2 P23 22_JTAP_TMS P22 21_JTAG_TCK P21 20_NC P20 19_GPIO28 P19 18_JTAG_TDO P18 17_NC P17 28_GND P28 29_NC P29 30_GND P30 31_RF_BG P31 32_GND P32 33_NC P33 34_SOP0 P34 35_N_RESET P35 36_VBAT_DCDC_ANA P36 37_VBAT_DCDC_PA P37 38_GND P38 39_VDD_ANA2 P39 40_VBAT_DCDC_DIG_IO P40 41_NC P41 42_GPIO30 P42 43_GND P43 44_GPIO0 P44 1_GND P1 2_GND P2 3_GPIO10 P3 4_GPIO11 P4 5_GPIO14 P5 6_GPIO15 P6 7_GPIO16 P7 8_GPIO17 P8 9_GPIO12 P9 10_GPIO13 P10 11_GPIO22 P11 12JTAG_TDI P12 13_NC P13 14_NC P14 15_NC P15 16_GND P16 45_NC P45 46_GPIO1 P46 47_GPO2 P47 48_GPIO3 P48 49_GPIO4 P49 50_GPIO5 P50 51_GPIO6 P51 52_GPIO7 P52 53_GPIO8 P53 54_GPIO9 P54 55 P55 56 P56 57 P57 58 P58 59_GND P59 60_GND P60 61_GND P61 62_GND P62 63_GND P63 CC3200MOD U$1 UBLOX NEO-6 13_GND 14_MOSI/CFG_COM0 15_MISO/CFG_COM1 16_CFG_CPS0/SCK 17_RESERVED 18_SDA2 19_SCL2 20_TXD1 21_RXD1 22_V_BCKP 23_VCC 24_GND 12_GND 11_RF_IN 10_GND 9_VCC_RF 8_RESERVED 7_VDDUSB 6_USB_DP 5_USB_DM 4_EXTINT0 3_TIMEPULSE 2_SS_N 1_RESERVED R1 R2 R3 R6 R4 R5 C10 C11C12 R8 SV2 1 2 3 4 5 6 7 8 9 10 PIO GND IC1 DATA 2 13 R12 C13 ANT1 U$4 GND INOUT JP3 1 2 C15 C16 C17 C18 T1 R13 ANT NRST NRST SOP0 SOP0 SOP1 SOP1 SOP2 SOP2 SOP2 TXD TXD RXD RXD SCL SCL SDA SDA RX1 RX1 TX1 TX1 TIMEPULSE TIMEPULSE SD_DATA0 SD_CS SD_CMD SD_CLK JTAG_TMS JTAG_TMS JTAG_TCK JTAG_TCK JTAG_TDO JTAG_TDO JTAG_TDI JTAG_TDI SD_CARD_DET ACC_INT1 ID ID LED2 LED3 LED1 N$1 N$19 N$22 GPS_ON GPS_ON D Benjamin (Rafi) Taub THIS IS A FOUR-LAYER BOARD! Released under the Creative Commons Attribution Share-Alike 4.0 License https://creativecommons.org/licenses/by-sa/4.0/ Design by: Appendix A: Full electrical schematics

Figure 13: Electrical schematic page 1 of 3 (main MCU and GPS)

Machine generated alternative text:
ADXL345
HMC5883LSMD
ITG-3200ORIG
2908-05WB-MG
VDD
GND
VDD
GND
GND
VDD
GND
0.22uF
4.7uf
GND
0.1uF
GND
0.1uF
GND
0.1uF2.2nf
0.1uF
JUMPER-PAD-2-NC_BY_TRACENO_SILK
GND
10k
VDD
VDD
GND
4.7uf0.1uF
GNDGND
0.1uF
MPL115A2
1uF
1uF
VDD
GND
GND
GND
GND
GND
U1
VDD
1
NC
11
NC
3
GND
4
GND
5
VS
6
CS
7
INT1
8
INT2
9
GND
2
NC
10
SDO
12
SDA
13
SCL
14
U2
SCL
1
VDD
2
NC
3
S1
4
NC
5
NC
6
NC
7
SETP
8
GND
9
C1
10
GND
11
SETC
12
VDDIO
13
NC
14
DRDY
15
SDA
16
U3
CLKIN
1
VLOGIC
8
AD0
9
REGOUT
10
RESV-G
11
INT
12
SDA
24
SCL
23
CPOUT
20
GND
18
VDD
13
JP1
CD/DAT3
CD/DAT3
CMD
CMD
GND
GND
VCC
VCC
CLK
CLK
DAT1
DAT1
DAT0
DAT0
DAT2
DAT2
SHIELD
GND3
SHIELD
GND1
CD1
CD1
CD2
CD2
C1
C2
C3
C4
C5C6
C7
JP2
R7
C8C9
C14
U4
VDD
1
CAP
2
GND
3
SHDN
4
RST/CS
5
NC/DOUT
6
SDA/DIN
7
SCLK
8
C19
C20
SDA
SDA
SDA
SDA
SCL
SCL
SCL
SCL
ACC_INT1
ACC_INT2
MAG_DRDY
G_INT
SD_DATA0
SD_CLK
SD_CMD
SD_CS
SD_CARD_DET
D
Benjamin (Rafi) Taub
Released under the Creative Commons
Attribution Share-Alike 4.0 License
 https://creativecommons.org/licenses/by-sa/4.0/
Design by:


Figure 14: Electrical schematic page 2 of 3 (sensors and SD card)

Machine generated alternative text:
470R
470R
470R
GND
GNDGND
LED1
LED2
LED3
R9
R10
R11
LED1LED2LED3
D
Benjamin (Rafi) Taub
Released under the Creative Commons
Attribution Share-Alike 4.0 License
 https://creativecommons.org/licenses/by-sa/4.0/
Design by:


Figure 15: Electrical schematic page 3 of 3 (LED indicators)

## Appendix B: PCB

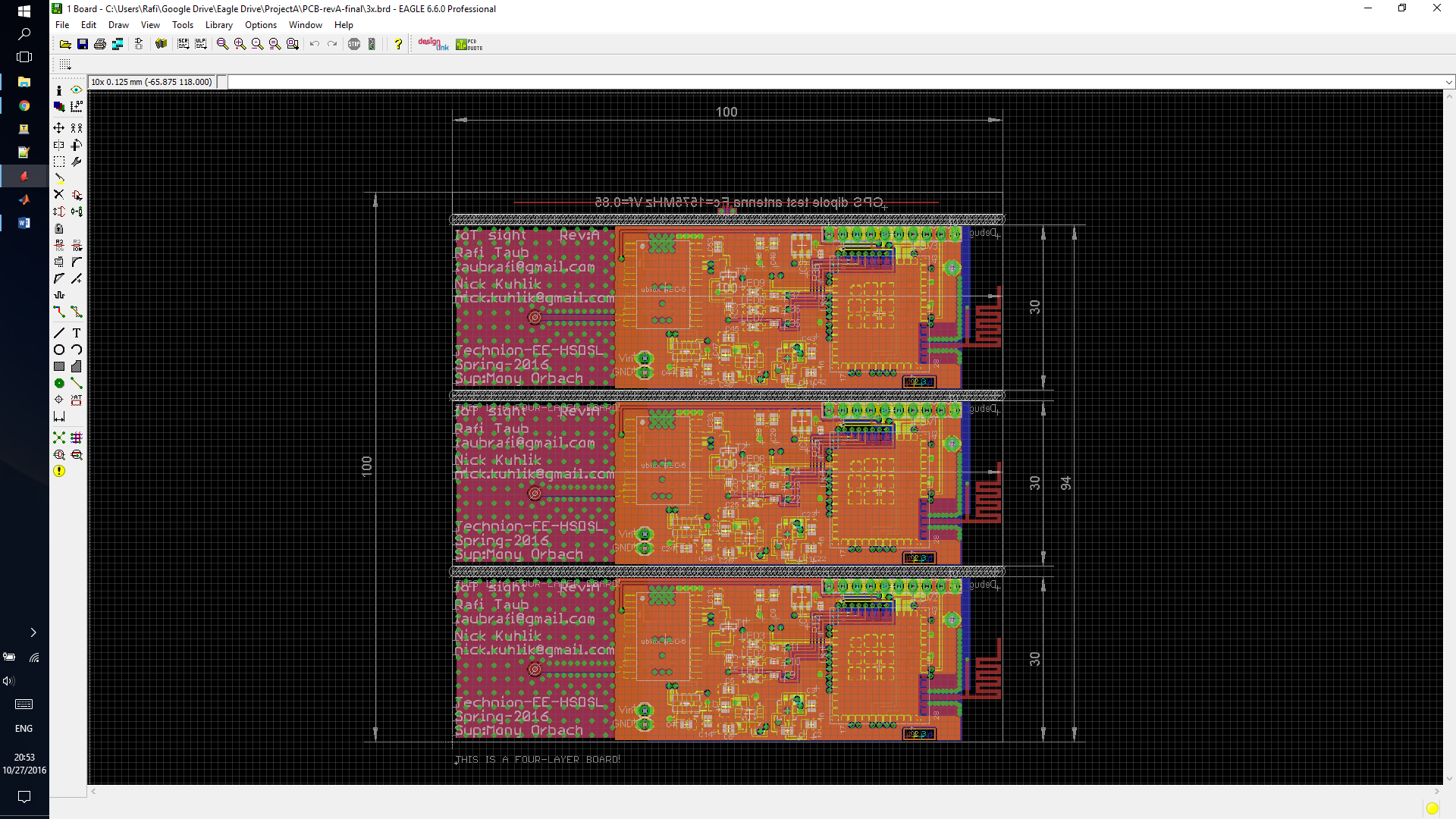


Figure 16: Full PCB for manufacturing

## Appendix C: Full mechanical drawings

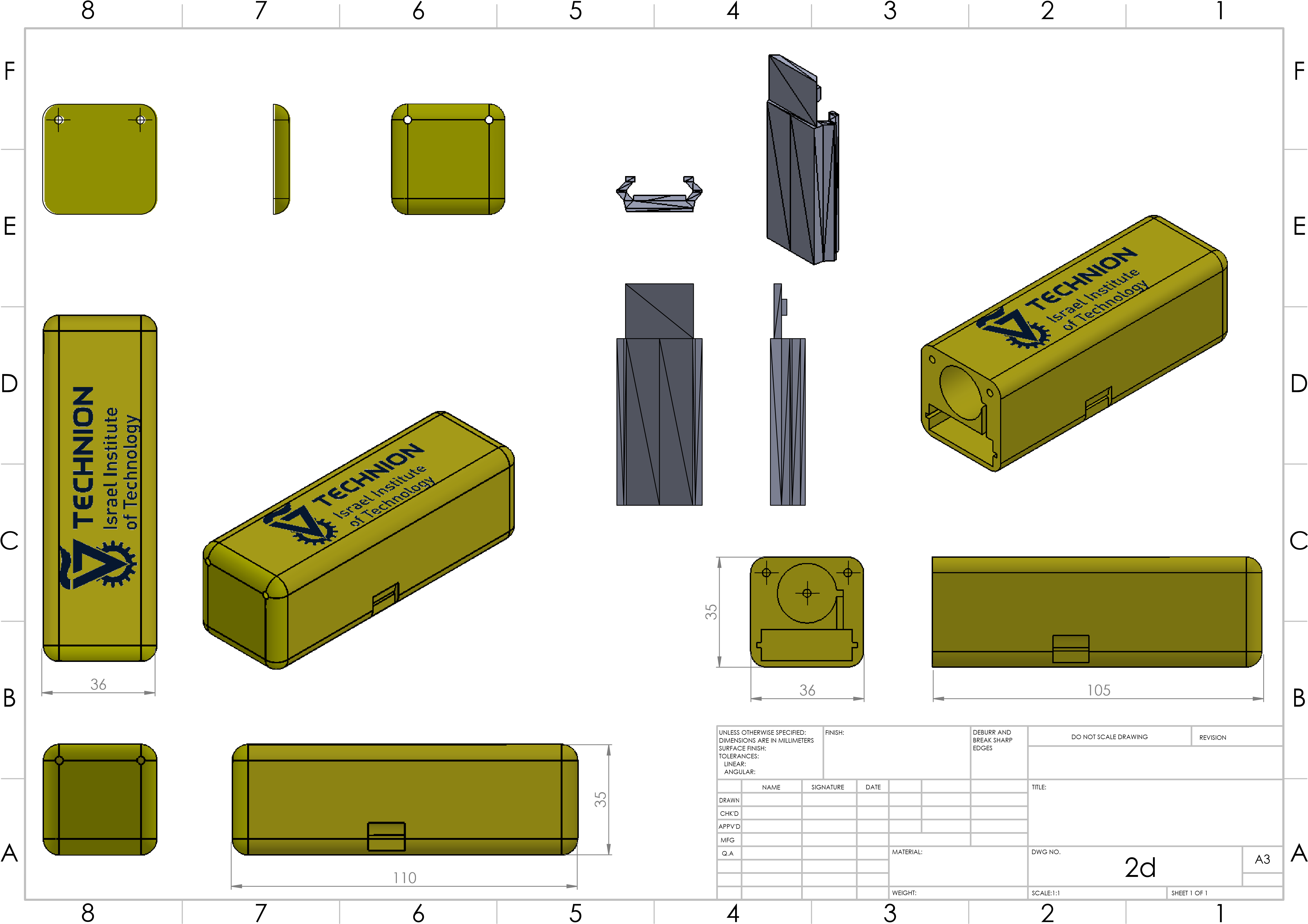


Figure 17: Mechanical diagrams

1. AN91445 from Cypress semiconductors [↑](#footnote-ref-2)