

QINETIQ

QinetiQ Young Engineers' Scheme 2017-18

Lucas Fowler-Dewe, Tom Vernon, Jasmine Jain,
and Khoa Bang Tran



Contents

- 1) Introduction (p. 3)**
- 2) Project Aims (p.3)**
- 3) User Requirements (p.3-4)**
- 4) Research (p. 4-8)**
 - a) Triangulation**
 - b) Trilateration**
 - c) Doppler Effect**
 - d) Signal Strength**
 - e) Internet Examples**
- 5) Job Allocation/ Planning (p. 8)**
- 6) Software (p. 8-12)**
- 7) Hardware (p. 13-14)**
- 8) Materials and Costs (p. 14)**
- 9) Evaluation of Design (p. 14-15)**
- 10) Evaluation of Functionality (p. 15-16)**
- 11) Conclusions and Implications (aesthetic design) (p. 16)**
- 12) Recommendations (p. 16-17)**
- 13) Final Prototype Code (p. 17-19)**
- 14) References (p. 20-21)**

1) Introduction

The *Qinetiq Young Engineers' Scheme of 2017-18* at Malvern College consists of being set a task by the 'customer,' and as a team, going through a start to finish process to design and create a solution to the 'customer's' problem.

The project is about working as a team and embarking on a process to create a solution to the given problem, as well as using each of the team members' abilities to their full potential. Planning and being able to equally divide up the work between each of the team members is vital to a project like this.

We were given the task of making an Electronic Warfare device, which will track the position of a "target" that emits radio signals. Such devices will be used in military and business security applications, discovering and allowing authorities to eliminate malicious rogue signals. The device has to be able to deploy rapidly, be able to autonomously report the position of the target, and be relatively inexpensive to allow for a ubiquitous system. We must not use any pre existing devices for our finished product, and the design must be completely of our own. This is expected to span across the timeframe of four to six months.

2) Project Aims

We decided to design and build a portable device which uses radio signals to report the target device's position to the user, with our aim and end goal being to have a fully functional device. To meet this goal, we had to consider many aspects of the process, which included planning the project, identifying requirements, doing background research, and creating a logical solution to the given problem.

3) User Requirements

Creating a list of user requirements was vital at the beginning of the design process as they allowed us to be able to start designing towards our end goal and would fulfill the customer's requests.

<u>Requirements</u>	<u>Explanation</u>	<u>Test</u>
Deploy Rapidly	Being able to begin functioning quickly and effectively will be an extremely important factor on its performance when using this device in the field.	Time how long it will take for it to power on from a full shut down. It will pass if it is under a minute.
Shows position of target	Displaying the target's position should help the user to find the target more easily.	Look at the screen to see if the target is displayed on it.

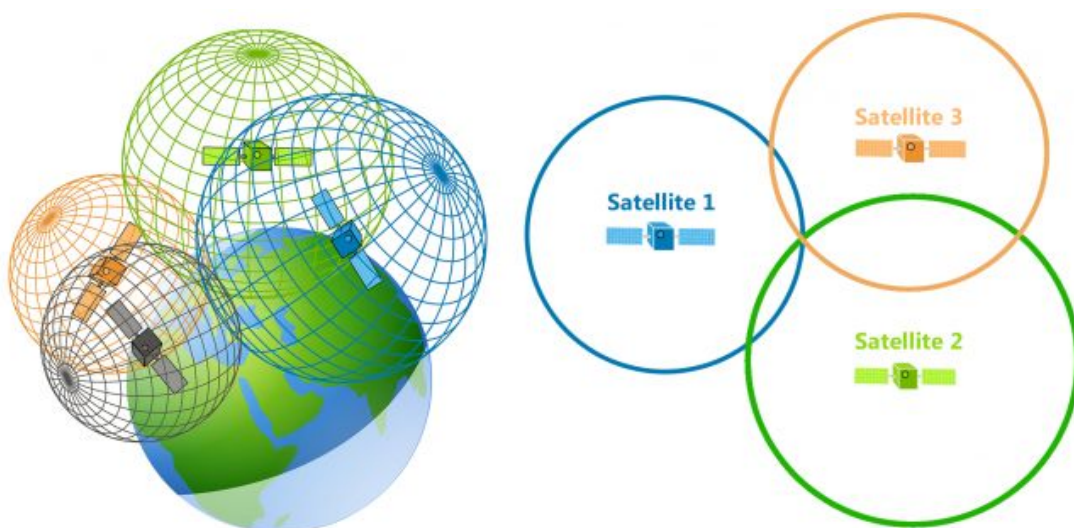
Cheap as possible	The budget we have been given is £300, but our aim is to use as little of it as we can	Calculate the sum of all of the components used. It'll pass if it is under £300
Lightweight and portable	Using this device in the field will be more effective and more efficient if it is light in weight and can easily be moved from place to place	Weigh it with a scale, should be less than 5kg; Give it to someone and observe how easily they can maneuver it
Locates target as quickly as possible	To achieve an exact location instantaneously would be very difficult, but our goal is to get as close to that as possible	Test how long it takes for a user to locate a hidden mobile hotspot. It'll pass if the device is located within 10 minutes
Sturdy and robust	The device should be strong enough that it does not break after dropping it by accident	Drop it several times from 1.5m and see if it is still fully functional.

4) Research

One of the first things that we researched was trilateration which is quite a basic method for pinpointing the location of an object.

This is done by using the time of arrival of a signal from known points to pinpoint the location of an object. **Figure 1 below** explains this visually. Trilateration is commonly used by geologists to find the location of Earthquakes, and is also used for GPS.

Figure 1

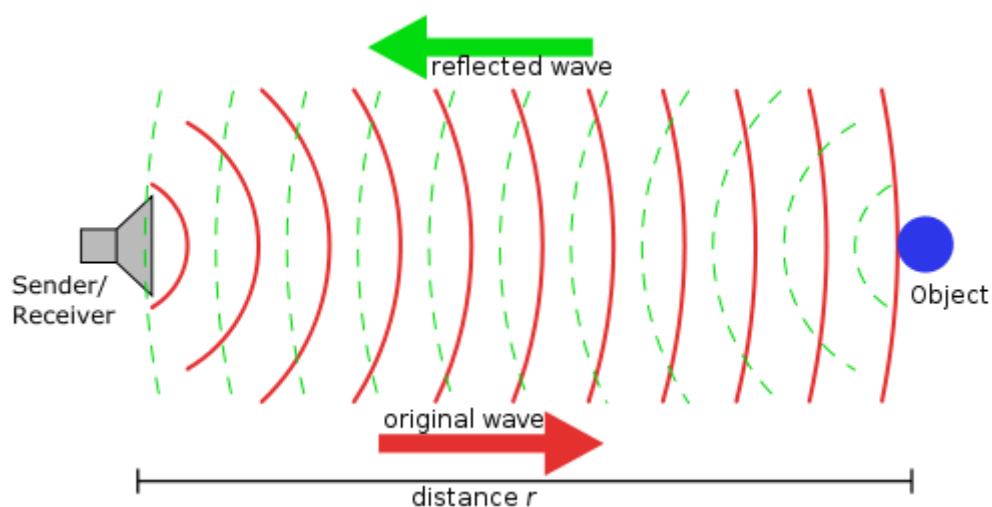


Trilateration is used to calculate the location of an object on a map or grid, using three known points. The three points have to be fixed relative to the map or grid for trilateration to work. Trilateration uses the principle of time of arrival between the three points to calculate the object's exact location. For example, GPS works this way, and even the location of earthquakes also uses this method.

However, what we thought was that it might be possible to reverse the process and make it work. Our idea was that we would use 3 antennas attached to the device and use it to monitor the airwaves and look for identical signals that it receives. Using the time differences between the receiving of the signal we could calculate a bearing and distance on the source. However, after doing some more research we decided to scrap this idea as it would require the antennas to be very far from each other - in order to reduce uncertainties and interference - making the device hard to set up and unportable and the code behind it would also need to be very sophisticated, which would be very time consuming to write.

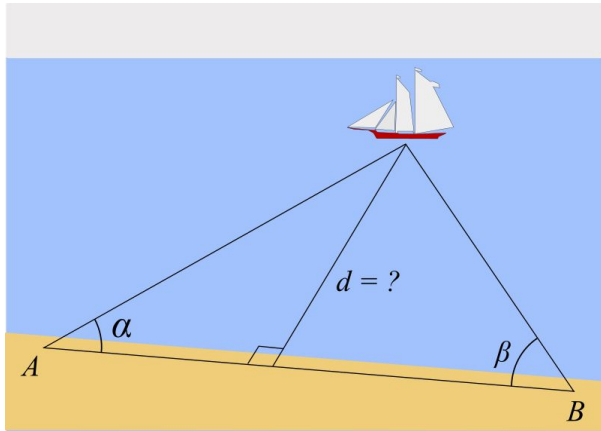
We also looked into radar, although after our research we deemed it unsuitable for use in our product. Radar is used to detect the angle and distance away of an object from a single known point. It does this by rotating and pinging a signal towards various objects. When the signal reflects off the object and reaches the known point again, it measures the time between the emitted signal, and the received signal after it has reflected off the object. Using this information, the distance of the object away from the known point can be calculated. The angle from the known point is already known from the ping being transmitted via a sweeping motion. It is commonly used for naval and aviation navigation. The picture below illustrates the radar. In most cases of radar, the sender/receiver in the diagram rotates at a constant speed 360°.

Figure 2



The reason it would be unsuitable is it relies on the fact that there are no objects in between the known point and the object, and it cannot be used to locate transmitters, because it relies on the fact that the object does not transmit any signal, only reflects.

Figure 3



Triangulation requires two known points and an object. The two known points and the object can be used to form two right angled triangles. Using simple trigonometry, the triangles can be used to calculate the distance between the object and the centre of the two known points. This is illustrated in the diagram to the left. We decided this would be the best approach for the project, as it is the simplest in terms of calculations and implementation.

We also looked into a sport called “Fox Hunting” in which a VHF transmitter is hidden within a certain area, and people go out to find said transmitter in a competitive fashion. The sport either uses “Null or Peak signal” to find the direction of the transmitter. In the case of peak signal finding, the operator is looking for the highest signal direction to find the direction. This can be quite unreliable due to the signal bouncing of landmarks, and buildings, which can intensify the signal and lead the operator off course. Since this is possible, null finding is another solution where you have to find the smallest signal result, and then rotate yourself 180°, and theoretically you should be pointing in the right direction.



Antennae

We looked in to many an antenna comparing size, gain, and price.

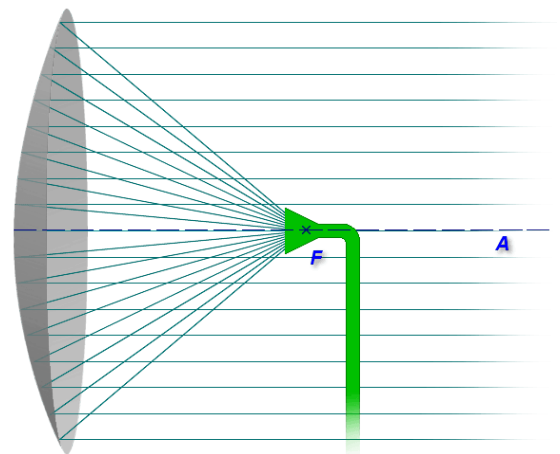
The first antenna that we looked into was the Yagi Antenna. This antenna uses a combination of directors, a reflector, and a driven element to allow the antenna to send and receive signals at very high gain. A good example of these are TV antennae. We built our own antenna, but in testing we found that the gain wasn't high enough, and there was too much interference to produce an accurate direction to the Wireless Access Point (WAP).



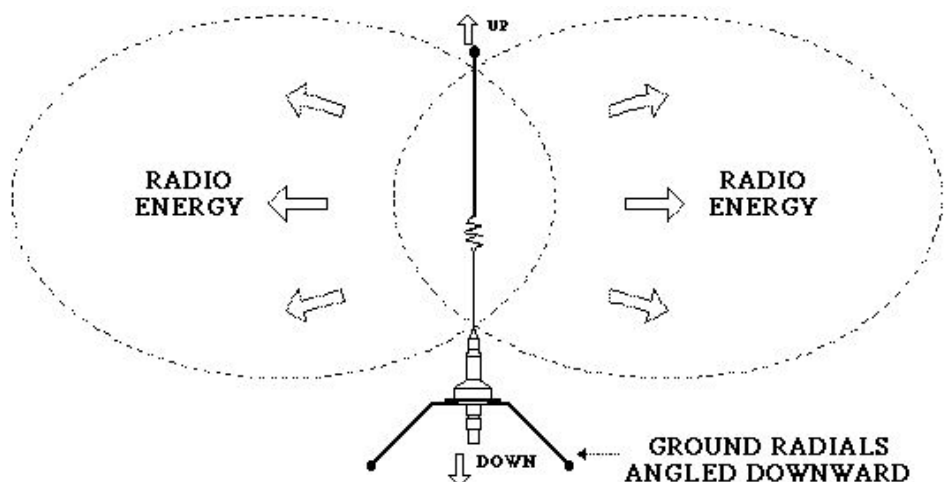
Another antenna we briefly touched upon was the can antenna. This is an antenna which is constructed typically with a coffee can (but other cans can be used). It is very cheap and easy to make, and you can also buy pre-built ones. However, we never experimented with this antenna because we found that in research that it's gain was relatively low compared to what we wanted in our device.



The next antenna that we looked at was the parabolic antenna. It uses a parabolically shaped dish to direct waves towards a receiver/transmitter, as illustrated in the diagram to the left. As a result, it has very high gain. This comes with a drawback though, which is that the dish has to be relatively large to have high gain. We decided that we would use this antenna, because it had the largest gain of all of the antennas that we researched. Although it is very large, we have implemented various methods to help with portability and usability.



The last antenna, the most common of all, is the omnidirectional antenna. It can send and receive signals from all directions hence “omni” being part of its name. It is used in all mobile phones, game consoles, computers and laptops alike to allow wifi connectivity. Although the parabolic antenna would be useful for long to medium range detection of WAPs, in our testing we found that it was less effective at short range. We tested the omnidirectional antenna built into our Raspberry pi to see how effective it was at allowing the user to find the WAP at close ranges, and found that the distance to the WAP and the signal reading correlated well enough for this task.



5) Job Allocation/ planning

At the start of this project we split the different tasks into separate sections, and then allocated a team member to each of the tasks, depending on what their skills were, so that

we were splitting up the workload and tailoring tasks to a certain member. We did a lot of planning for the future of the project to ensure that we stayed on schedule. To ensure this we setup a “Monday” account which is an internet calendar app that the whole team could access and update their positioning on any task that was required of them. For the actual spread of tasks, we tried to give everyone an equal amount of sections so that we were working at a productive and efficient speed.

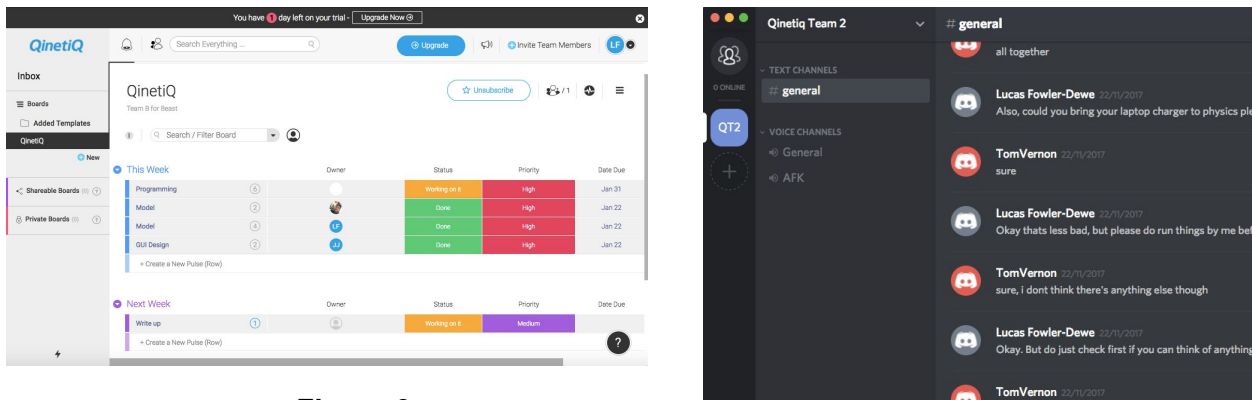


Figure 6

Jasmine was in charge of Research and Planning, Khoa Bang worked on programming, Tom worked on Hardware, Prototyping and the actual build of the product, and Lucas on Design Ideas, Prototyping, and the actual build of the product.

Obviously, there would be some overlapping when one person or another required assistance from another team member, and that was fine, because it meant that we were all keeping in touch with each other about different parts of the project without having to do a weekly discussion.

We organised a meeting at 4:30pm every Monday to go over what we had been doing during the week and to give us a chance to to do parts of the project together. For normal communication we set up a Discord group chat so that we could update each other on our progress and ask the team any questions that we might have.

For storing of files we set up a group Google Drive so that we could all edit and read everybody else's work, and comment on it if need be. We would then be informed on discord of anything that has been put onto the Google Docs by the editor themselves.

6) Software

Functionality and coding

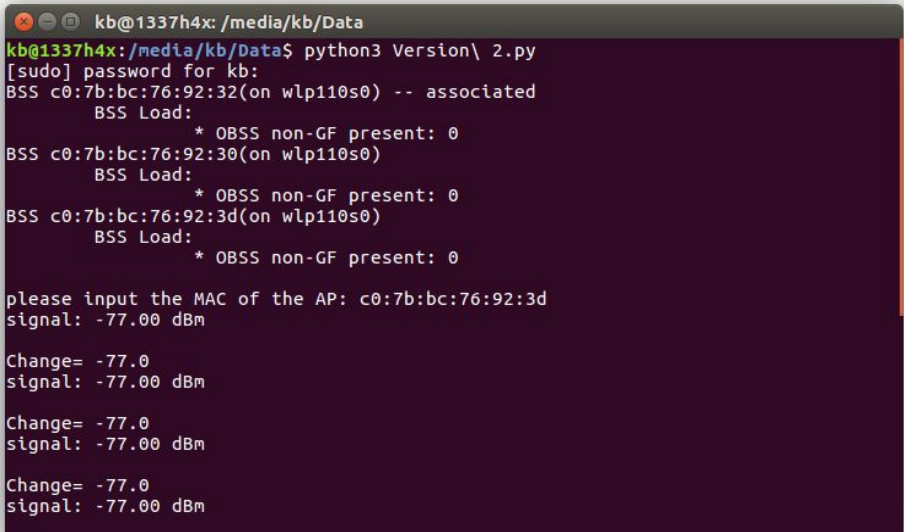
There were a number of ways in which we could tackle our problem. We could work on the GUI (Graphical User Interface) first, the scanning code first or the design of the interface. We decided to focus on the scanning first because it is its primary function, making it vital for the project.

We decided to create our program in Python 3 because, from our research, it is recommended as one of the best languages to get started in Linux app development (Ask Ubuntu, N.d.) and it is also the programming language that the team is most comfortable working with. With those initial parameters agreed on, we focused on creating our first

prototype to display the signal strength of a single access point. This is a screenshot of the program running:

```
import subprocess
import time

print(subprocess.check_output('sudo iw wlp110s0 scan | grep "BSS"', shell=True,
universal_newlines=True))
BSSID=input("please input the MAC of the AP: ")
old=0
while True:
    temp=subprocess.check_output('sudo iw wlp110s0 scan | grep -A 5 '+BSSID+'', shell=True,
universal_newlines=True)
    Current=float(temp[temp.find("signal")+8:temp.find("dBm")])
    print(temp[temp.find("signal"):])
    print("Change=", Current-old)
    time.sleep(1)
```



```
kb@1337h4x: /media/kb/Data
kb@1337h4x:/media/kb/Data$ python3 Version\ 2.py
[sudo] password for kb:
BSS c0:7b:bc:76:92:32(on wlp110s0) -- associated
    BSS Load:
        * OBSS non-GF present: 0
BSS c0:7b:bc:76:92:30(on wlp110s0)
    BSS Load:
        * OBSS non-GF present: 0
BSS c0:7b:bc:76:92:3d(on wlp110s0)
    BSS Load:
        * OBSS non-GF present: 0

please input the MAC of the AP: c0:7b:bc:76:92:3d
signal: -77.00 dBm

Change= -77.0
signal: -77.00 dBm

Change= -77.0
signal: -77.00 dBm

Change= -77.0
signal: -77.00 dBm
```

One of the biggest difficulties that we ran into while making this prototype was compatibility. We tried to, but could not, get networking modules to work since almost all of them were written for Python 2 (Pythonhosted.org, 2017). Further complicating things, although there were documentation for the handful of Python 3 modules, they were not very good and were hard to understand, which rendered the modules useless. Some of them were duplicates of the incompatible Python 2 modules, but were labelled with "Python 3 version". In the end, to scan for the WiFi networks we had to rely on using the subprocess module that was built into Python. The function allowed us to open a hidden terminal to run Linux commands and grab data from. However, the process was very finicky as the module was meant for short commands with simple outputs, not complicated multi line outputs typical of the "iw scan" command. This made the program considerably slow but at least it was functional.

The other compatibility problem we had was with the operating system. When we began working on the software, we chose to utilise Ubuntu Mate for the Raspberry Pi. When we tried running the prototype program on the it, the program gave us a few unexpected errors about the subprocess module. We spent several weeks trying to solve this and managed to solve it accidentally by switching to Raspbian. For reasons unknown to us, Raspbian could run the same script without any errors.

```

Open ▾  FI
Version 3.py
import subprocess
import time

print(subprocess.check_output('sudo iw wlan0 scan | grep "BSS"', shell=True,
universal_newlines=True))
BSSID=input("please input the MAC of the AP: ")

Signals=[]
Max=-10000

while True:
    temp=subprocess.check_output('sudo iw wlan0 scan | grep -A 5 "BSSID+"', shell=True,
universal_newlines=True)
    Current=float(temp[temp.find("signal")+8:temp.find("dBm")])
    print(temp[temp.find("signal"):])
    if Current > Max:
        Max=Current
        print()
        print("New max")
        time.sleep(3)

```

```

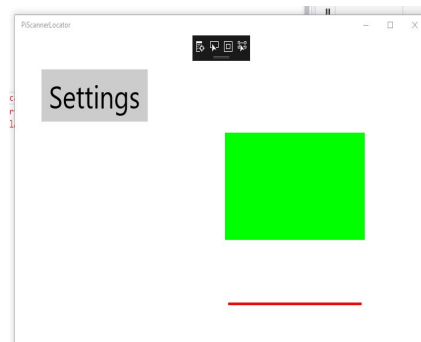
BSS c0:7b:bc:76:ff:22(on wlan0)
BSS Load:
* OBSS non-GF present: 0
BSS c0:7b:bc:76:ff:2f(on wlan0)
BSS Load:
* OBSS non-GF present: 0
please input the MAC of the AP: c0:7b:bc:76:ff:2f
signal: -79.00 dBm
New max
signal: -79.00 dBm
signal: -79.00 dBm
signal: -79.00 dBm
signal: -79.00 dBm
signal: -79.00 dBm

```

The final issue we had was with driver compatibility. While it would be better to use audio input for the WiFi receivers and utilise software defined radio - giving the program much greater flexibility - we thought that this would add too much work for our limited time, so we opted for a USB network interface. By utilising a common interface, we should be able to cut down on the amount of programming needed and reduce testing times. At first, our receiver didn't work with the pi so we had to use some unofficial Linux drivers. However, several weeks later, a Linux kernel update had stopped the driver from working. As a result, we had to spend more time and resources acquiring another WiFi NIC (Network Interface Card) that was compatible with Linux.

To avoid any further compatibility issues, we decided to switch to using Windows IoT (Internet of Things) Core for Raspberry Pi. Windows should help to alleviate these problems as it has its own library for WiFi built into its main programming language that is very well documented by Microsoft. Additionally, Windows is a lot less fragmented than Linux - with Linux there are a large number of distros that have subtly different behaviours and documentation is not always helpful as they may refer to a different version of the operating system. For example, the documentation that the team had used for the subprocess functions worked with Ubuntu 16.04 and Raspbian but didn't work with Ubuntu Mate.

The only real issue with switching would be that the program would have to be rewritten in another language as Windows IoT Core does not support Python. Windows IoT Core requires all of its apps to be UWP (Universal Windows Platform), which are written in C#, C++, Javascript or Visual Basics (Microsoft Corp., 2017). Nevertheless, we made a prototype app with the basic graphical user interface to test out C#'s capabilities (see screenshot below). However, we ran into a major problem during the switch. When we were installing Windows on the Raspberry Pi, we managed to get it to boot exactly once. After that the Pi refused to boot. We suspected that it has something to do with our hardware, or our school's network corrupting the windows image. Either way, with time running short we decided to go back to Raspbian for this prototype and work on Windows in the next revision. If we were making a second prototype, we would then purchase all the parts that are confirmed to be working with Windows IoT from a verified list to avoid the problems we had with driver incompatibility issues (Microsoft Corp., 2017).

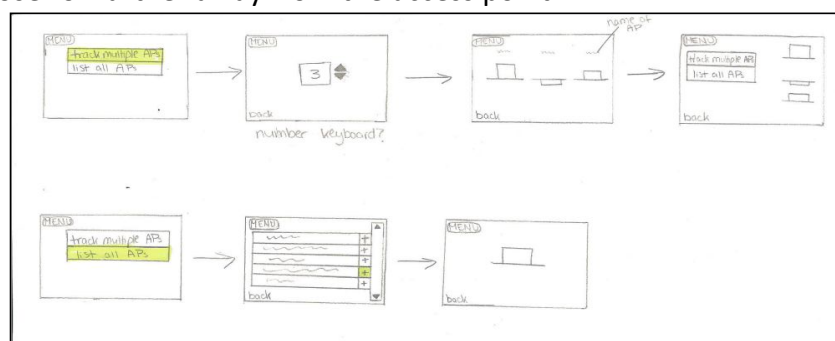


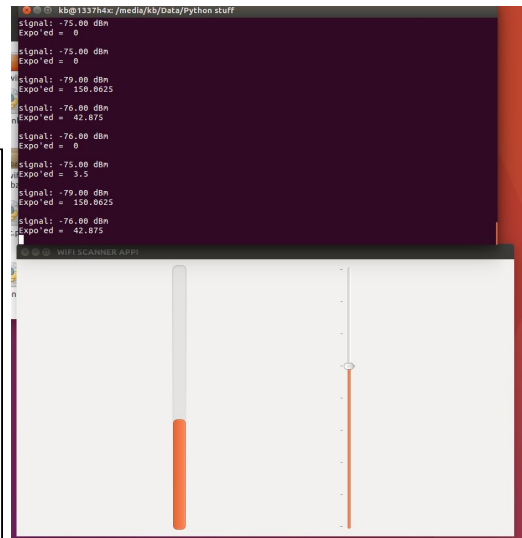
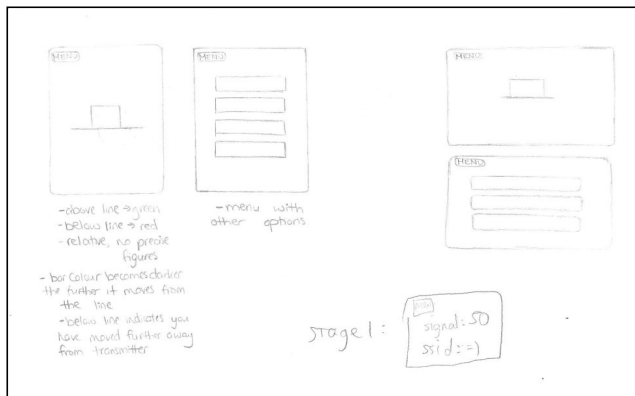
With the focus back on the Python version, we decided to start working on the Python user interface. To do this, we used the module PyQt as it is recommended as one of the best modules for creating graphical apps.

We spent several weeks going through the tutorial (PyQt was a lot less intuitive than C#) before we realised that we couldn't utilise rectangles in the same way we did in C#. In C#, we drew rectangles to build the basic interface, changing the variable of the rectangle's height when we needed to. On the other hand, with PyQt rectangles couldn't have their variables changed retrospectively with a single line of code. Instead, we have chosen to use progress bars in our prototype program as its variable could be changed retrospectively. One annoyance with PyQt was that if the program was busy doing something (like scanning the current available WiFi networks), the GUI would become unresponsive. To circumvent this, we implemented basic multithreading into our program to allow the GUI to function independently from the scanning and not become unresponsive.

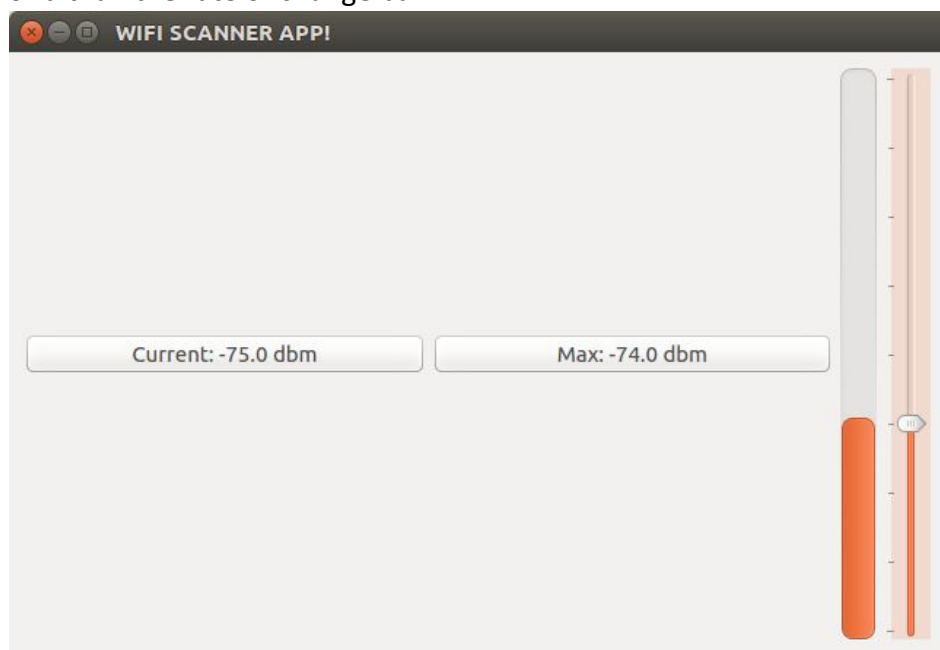
User Interface

During meetings, we also discussed and planned how the interface will look like and also what features we want it to have in the end. We planned out the layout of the interface by using drawings and once we were happy with the layout, we proceeded to add some colour into it. We needed a user interface that was simple and easy to understand, so we adopted a minimalistic design philosophy. We planned for the program to only have a bar that displayed the rate of change of the signal strength because we thought that the users would be confused if it was not clear how close they were to the access point. The rate of change bar should help clear up the confusion as it should be clearer for the user to see that they are getting closer or further away from the access point.



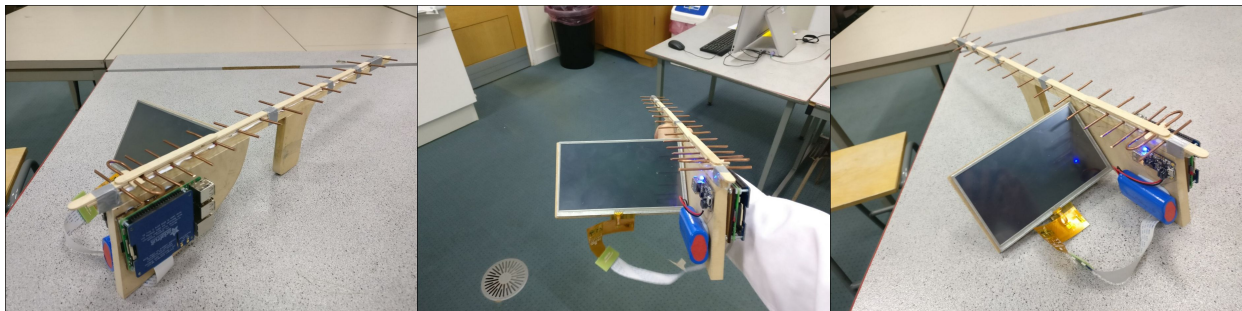


During a trial run of this version of the app (which only had a bar that displayed the rate of change of the signal strength and a slider that lets the user adjust the sensitivity of the bar), we realised that the rate of change bar only worked in theory. Due to our network interface card's slow refresh rate and interference from various background sources, the bar often jumped around, making it very hard to use. Because of this, we had to abandon our minimalistic design philosophy and added two boxes that showed the current signal strength and the maximum signal strength recorded (which can be reset by tapping on it). After another round of tests, we confirmed that these two boxes performed much better in the real world than the rate of change bar.



7) Hardware

First we did research into what antenna we should use. We looked at previous projects where people used directional antennas, such as fox hunting and other forms of radio location. From this, we looked at various directional antennas, and settled on the Yagi antenna. We decided to build our own. Although Yagi antennas themselves are directional enough, the one we built was not, and its gain was not high enough. As a team, we decided to scrap the ideas for the Yagi antenna, and go for a parabolic antenna, that we would buy. This would not affect the software or the hardware that we would use, as the main principles of our design would stay the same. Only the antenna and therefore aesthetics would change. Below are pictures of our first prototype.



We decided on the parabolic antenna, although the higher the gain, the larger the antenna. As a team, we wanted as high as gain possible, so we chose the huge 100cm by 60cm TP link parabolic antenna.

As for the components of the device itself, our design is required to be portable (from our set user requirements), so we chose a 6600mAh battery along with a battery controller to allow it to recharge via micro USB. We also chose a 7" touch screen display and a special hat which connects to the General Purpose Input/ Output (GPIO) pins on the Raspberry pi. This allows the screen to be powered off the pins and thus reduce power consumption.

We started software development on Khoa Bang's computer. As the we were going to use Ubuntu on the Raspberry Pi, we also made sure Khoa Bang did the development in Ubuntu. Unfortunately, we had some errors with the display and some Python errors, so we switched the Raspberry Pi's OS to Raspbian. This also happened to be based on Linux debian, like Ubuntu, so there were no compatibility issues. While the GUI is in the making, we also composed a simple shell script which lists the SSIDs (Service Set Identifier) of all the detected WAPs and their signal strengths, so we can still test our Antenna's functionality if it is built and the GUI isn't finished yet. We later switched to Khoa Bang's Python script which could also display MAC addresses of the WAPs and the change of the maximum of the signal strength. After installing Raspbian, we found that the display issues weren't unique to Ubuntu, but were also present in Raspbian. The issue was that the display worked, but the touch screen didn't. We eventually got it to work by modifying the python driver files by adding and removing a few lines. However, the next week, when putting it all together, the touch screen showed the same errors as if we didn't fix it at all.

As for ergonomics, we recycled the end of a crutch. Lucas cut off the top of the crutch using a saw, and mounted it securely to our parabolic antenna's grid array, using bespoke bolts, and a tap and die set. We also replaced the plastic grip with a tennis grip, to add comfort for

the user. Soft foam padding was added to the area of the grid array which contacts the user's hand, for protection. We mounted the screen and all the main components onto a board, at about head height to the grid array, using velcro so it can be removable. This is so if the user wants to switch to the omnidirectional antenna which is built into the main board, they can remove the board and carry it by hand. This way, they don't have to carry around the large and heavy parabolic antenna which isn't being used at that time.

8) Materials and Costs

We were given a budget of £300, but one of the goals for our project was to use as little of that money as we could, while still carrying out the same quality of its functionality and aesthetics. We initially tried to see if we could get a WiFi antenna to work with a mobile device, to test our proof of concept with our ideas. To do this, we bought some USB-C to USB-A adaptors, but after some testing we discovered we'd have to root an Android device to allow USB WiFi Network Interface Cards (NICs) to work.

In terms of electronics, Tom happily lent the team some of his spare Raspberry Pi equipment - The Pi itself, a Li-Ion Battery, a battery charger/controller, a 7" TFT display and a display hat to allow the display to connect to the Pi. However, the display and the hat mysteriously stopped working over a half term holiday, so the team had to buy a different, smaller 3" Thin Film Transistor (TFT) display. A TP-link USB WiFi NIC was bought to allow connectivity between the Pi and our chosen antenna. However, due to compatibility issues, this did not work with the Pi. As a result, we bought another Blueway USB WiFi NIC that was compatible with the Pi. The antenna we bought was a huge 100x60cm Parabolic dish, and connected nicely with our Blueway WiFi NIC. Various cables were also bought for connectivity between components.

Other materials we used were: a wooden board we cut our ourselves to attach all of our electronics to, small pieces of MDF to mount onto each separate electronic board and to allow easier mounting to the main wooden board, a modified crutch handle to allow the user to hold the antenna, and a tennis grip to apply around the handle of the crutch to give the user more comfort when holding the device.

Altogether the project has cost us £117.

9) Evaluation of Design

What we decided to do for the final product was to use a parabolic antenna to detect the WiFi sources. The pros of this antenna are that it has a high gain, and is lightweight; which means that it is covering two of our requirements which are that it will show us the target as accurately as possible with the antennas available, and it is easy to transport. The cons of this are that for the antenna to be accurate it has to be quite large, which means that it could possibly be hard to manoeuvre in small, tight spaces.

We overcame this problem by using the antenna built into the Pi. This solution is made possible because when the device is used in cramped conditions the user will be close enough to the target that we do not need the high gain of the large antenna to detect the WiFi signal.

The use of a crutch to hold the antenna, which was inspired by the design of the riot shield. This has been very effective, and we have made it more comfortable to hold by adding a grip to the handle, and some foam to act as a hand guard, so that it is more comfortable.



10) Evaluation of Functionality and Features

With the prototype hardware and software in a functional state, we devised a test to evaluate the overall functionality and effectiveness of our prototype solution. We did this by hiding a phone with its personal hotspot on in a certain area, and used the device to find it. To stop the phone turning off, we had a laptop connect to the hotspot. A video of the test can be found using the following link: https://drive.google.com/open?id=11vITrWY7k-6HHQAezLnjbITWRIOLuw_0

We also then performed all of the tests mentioned in the user requirements section.

<u>Requirements</u>	<u>Test Result</u>	<u>Evaluation</u>
Deploy Rapidly	We ran the test from when the device was off, and stopped when the device was booted and the application was fully set up. Overall, it took just under a minute	Since it is under a minute, it passed our test, albeit just barely. This was because we spent half the time waiting for the operating system to boot
Shows position of target	We hid a phone with its personal hotspot on in a certain area, and used the device to find it. The device displayed the signal strength, which can be used to infer the target's position.	Due to technical limitations, our device could not fully display the device's location. However, it was still functional without this feature.
Cheap as possible	We totalled the costs, which was £117.	Since it is under £300, it passes the test.
Lightweight and portable	On a digital scale, we got 6kg for the entire device. The stripped down	6kg is a little bit too heavy, but we think it is worth the tradeoff for increased signal

	version of it with only the Pi and a battery only weighed 1.5 kg.	strength, and the user can use the stripped down version when they need portability
Locates target as quickly as possible	From our tests, we managed to locate the hotspot within 10 minutes	While 10 minutes is good, it could be even better if it could locate the hotspot within 5 minutes.
Sturdy and robust	We dropped it 5 times onto ceramic floor.	It was still fully functional afterwards, though a few loose components fell off.

11) Conclusions

Our design for the product was limited due to the parabolic antenna, meaning that we had to use the parabolic dish as our aesthetic and eye catching feature. This ended up well due to our idea of holding the antenna like a riot shield, connoting ideas of military weaponry.

The network interface card also limited the capabilities of the device as it had a very low refresh rate, meaning that we could not get a refresh rate high enough to use a rate of change bar, making the interface less intuitive.

The making of the device turned out to be quite simplistic due to us having planned for everything to be attached and revolved around the parabolic dish. The challenge was how to get the arm support mounted to the dish. We resolved this problem by sacrificing the adjustability feature of the support, and using the already fabricated holes to use as feeders for the custom bolts that we made which were fixed to the dish via large washers and bolts, and then secured to the inside of the support with the use of a tap and die set to create threaded holes for the bolts.

12) Implications and Recommendations

The device is functional, able to find rogue signal emitters. The implications of this is that the military and businesses could easily fund such a project. This project has proved that such a device is possible, further increasing the likelihood of more advanced projects. In the future, it is likely that such devices will be very common in the future as it aids security to do their job.

While the prototype device could not precisely locate an access point and show its position on a map - a feature that we had hoped to achieve - a user with some basic training or instructions should be able to operate it and locate an access point within 10 minutes. Incremental updates to the hardware, such as a faster network interface card, could help reduce this time. However, in order to achieve near instant tracking and map position display, a different design for the device will be required - due to the inherent limitation of basing the tracking process on one antenna's signal strength reading, requiring it to move to scan in all directions.

As for the software, the current setup is very unstable. If we had more time we would look at migrating the project to Windows IoT Core to improve stability. Windows should also help with productivity as more stable code and better documentation meant less time spent debugging.

13) Final prototype code

```
# !/usr/bin/env python3
# coding: utf-8
```

```
import subprocess
import math
```

```
import sys
from PyQt4 import QtGui, QtCore
```

```
global old, current, change, sensitivity, max
change = 0
current = 0
sensitivity = 1.5
max = -1000
```

```
class Window(QtGui.QWidget):
    def __init__(self, parent=None):
        super(Window, self).__init__(parent)
        self.setGeometry(300, 200, 900, 500)
        self.setWindowTitle("WIFI SCANNER APP!") # FearFactor ++
        self.home()

        self.ScanThread = Scanning()
        self.ScanThread.start()
        self.connect(self.ScanThread, QtCore.SIGNAL('SCAN'), self.update_num)

    # Flips the bar to show a decrease or increase
    def update_num(self, change):
        change_expo = math.pow(sensitivity, change)
        if current > old:
            # Making it exponential makes it easier
            # for the user to see the difference
            self.change_progress.setInvertedAppearance(False)
        elif current < old:
            self.change_progress.setInvertedAppearance(True)
        else:
```

```
    change_expo = 0
    print("Expo'ed = ", change_expo)
    self.change_progress.setValue(change_expo)

    global current
    self.cur_button.setText("Current: "+str(current)+" dbm")

    global max
    self.max_button.setText("Max: "+str(max)+" dbm")

def update_sensitivity(self):
    global sensitivity
    sensitivity = float(self.sl.value()/2)
    print("New sensitivity:", sensitivity)

def home(self):
    layout = QtGui.QHBoxLayout()
    self.setLayout(layout)

    global current
    cur_str = "Current: "+str(current)+" dbm"
    self.cur_button = QtGui.QPushButton(str(cur_str), self)
    layout.addWidget(self.cur_button)

    global max
    max_str = "Max: "+str(max)+" dbm"
    self.max_button = QtGui.QPushButton(str(max_str), self)
    self.max_button.clicked.connect(self.update_max)
    layout.addWidget(self.max_button)

    # Progressbar to show the change in strength
    self.change_progress = QtGui.QProgressBar(self)
    self.change_progress.setOrientation(QtCore.Qt.Vertical)
    self.change_progress.setTextVisible(False)
    layout.addWidget(self.change_progress)

    # Lets users adjust the sensitivity
    self.sl = QtGui.QSlider(QtCore.Qt.Vertical)
    self.sl.setMinimum(2)
    self.sl.setMaximum(10)
    self.sl.setValue(3)
    self.sl.valueChanged.connect(self.update_sensitivity)
    self.sl.setTickPosition(QtGui.QSlider.TicksLeft)
    self.sl.setTickInterval(1)
```



```
layout.addWidget(self.sl)
```

```
self.show()
```

```
def update_max(self):  
    global max  
    max = current  
    self.max_button.setText("Max: "+str(max)+" dbm")
```

```
class Scanning(QtCore.QThread):  
    # Uses threading, allows the scanner to work without  
    # causing the GUI elements to become unresponsive  
    def __init__(self, parent=None):  
        super(Scanning, self).__init__(parent)  
  
    def __del__(self):  
        self.wait()  
  
    def run(self):  
        old = 0  
        current = 0  
        temp = "lorep ipsum"  
        while True:  
            # Sometimes the interface card may be busy and spits out an error,  
            # this prevents a crash if it happens  
            try:  
                temp = subprocess.check_output('sudo iw '+name+'  
                                                ' scan | grep -A 5 '+BSSID+',  
                                                shell=True,  
                                                universal_newlines=True)  
            except:  
                temp = temp  
            global current, old, change, max  
            old = current  
            current = float(temp[temp.find("signal")+8:temp.find("dBm")])  
            # Finds the target signal from the output  
            if current > max:  
                max = current  
            print("")  
            print(temp[temp.find("signal"):temp.find("dBm")+3])  
            change = abs(current-old)  
            self.emit(QtCore.SIGNAL('SCAN'), change)
```

```
def turn_off(self):
    sys.exit()

def run():
    app = QtGui.QApplication(sys.argv)
    GUI = Window()
    sys.exit(app.exec_())

# Temporary method of selecting the adapter and ssid,
# mainly for debugging purposes
print(subprocess.check_output('iwconfig', shell=True, universal_newlines=True))
name = input("please enter the network adapter's name: ")

print(subprocess.check_output('sudo iw '+name+' scan | egrep "BSS|SSID"',
                             shell=True, universal_newlines=True))
BSSID = input("Please input the MAC of the AP: ")

run()
```

Presentation Feedback

After we had finished the project, We presented our design, and findings to our mentors and some other people affiliated with QinetiQ. The overall feedback was good, and they liked the fact that we had improvised with the antenna so as to make it more easily portable. One of the mentors wanted to try out the product itself, which they did and reported that they were pleased with the balance and ease in which they could hold the relatively heavy device. Their main issue with the device was that it takes too long to boot up with the hardware that we had to hand. If we were to change this in the future we could most likely get a much faster speed. Overall it seemed that the mentors were thoroughly impressed with what we had managed to achieve and produce in the short amount of time we had.

14) References

- Ask Ubuntu. (n.d.). *How do I get started creating an Ubuntu Desktop App?*. [online] Available at: <https://askubuntu.com/questions/49849/how-do-i-get-started-creating-an-ubuntu-desktop-app> [Accessed 4 Jun. 2018].
- Biotele (2011). *Easy to Build WIFI 2.4GHz Yagi Antenna*. [online] Instructables.com. Available at: <http://www.instructables.com/id/Easy-to-Build-WIFI-24GHz-Yagi-Antenna/> [Accessed 6 Jun. 2018].
- Buckley, I. (2017). *How To Make a Wi-Fi Antenna Out Of a Pringles Can*. [online] MakeUseOf. Available at: <https://www.makeuseof.com/tag/how-to-make-a-wifi-antenna-out-of-a-pringles-can-nb/> [Accessed 6 Jun. 2018].
- Canonical Ltd (n.d.). *Ubuntu Manpage: NetworkManager - network management daemon*. [online] Manpages.ubuntu.com. Available at: <http://manpages.ubuntu.com/manpages/bionic/en/man8/NetworkManager.8.html> [Accessed 6 Jun. 2018].
- Canonical Ltd (n.d.). *Ubuntu Manpage: Wicd - Wired and Wireless Network Connection Manager*. [online] Manpages.ubuntu.com. Available at: <http://manpages.ubuntu.com/manpages/bionic/en/man8/wicd.8.html> [Accessed 6 Jun. 2018].
- Wikipedia. (n.d.). *Free-space path loss*. [online] Available at: https://en.wikipedia.org/wiki/Free-space_path_loss [Accessed 6 Jun. 2018].
- Gite, V. (2012). *8 Linux Commands: To Find Out Wireless Network Speed, Signal Strength And Other Information*. [online] nixCraft. Available at: <https://www.cyberciti.biz/tips/linux-find-out-wireless-network-speed-signal-strength.html> [Accessed 4 Jun. 2018].
- Microsoft Corp. (2018). *Developing foreground applications - Windows IoT*. [online] Docs.microsoft.com. Available at: <https://docs.microsoft.com/en-us/windows/iot-core/develop-your-app/buildingappsfor-iotcore> [Accessed 12 Jun. 2018].
- Microsoft Corp. (2017). *Hardware Compatibility List - Windows IoT*. [online] Docs.microsoft.com. Available at: <https://docs.microsoft.com/en-us/windows/iot-core/learn-about-hardware/hardwarecompatlist> [Accessed 12 Jun. 2018].
- Pythonhosted.org. (2017). *Python API to talk to NetworkManager — python-networkmanager 2.0.1 documentation*. [online] Available at: <https://pythonhosted.org/python-networkmanager/> [Accessed 6 Jun. 2018].

- RasHAWK Distributed EM Situational Awareness Based on Raspberry Pi and REDHAWK. (2014). [ebook] Baltimore, Maryland, USA: Geon Technologies, LLC. Available at: http://www.geontech.com/download/white_papers/AOC_Challenge_GEON.pdf [Accessed 4 Jun. 2018].
- Redhawksdr.github.io. (n.d.). *Home*. [online] Available at: <https://redhawksdr.github.io/Documentation/> [Accessed 4 Jun. 2018].
- Robinson, S. (2015). *python-wifi*. [online] PyPI. Available at: <https://pypi.python.org/pypi/python-wifi> [Accessed 4 Jun. 2018].
- rtl-sdr.com. (2015). *An RTL-SDR Phase Correlative Direction Finder - rtl-sdr.com*. [online] Available at: <https://www.rtl-sdr.com/an-rtl-sdr-phase-correlative-direction-finder/> [Accessed 4 Jun. 2018].
- rtl-sdr.com. (2015). *Signal Direction Finding with an RTL-SDR, Raspberry Pi and REDHAWK*. [online] Available at: <https://www.rtl-sdr.com/signal-direction-finding-with-an-rtl-sdr-raspberry-pi-and-redhawk/> [Accessed 4 Jun. 2018].
- SpenchWiki. (n.d.). *SDRDF*. [online] Available at: <http://wiki.spench.net/wiki/SDRDF> [Accessed 4 Jun. 2018].
- Wifi.readthedocs.io. (2012). *Managing WiFi networks — wifi 0.2.0 documentation*. [online] Available at: <https://wifi.readthedocs.io/en/latest/scanning.html> [Accessed 6 Jun. 2018].
- Wikipedia. (n.d.). *Indoor positioning system*. [online] Available at: https://en.wikipedia.org/wiki/Indoor_positioning_system [Accessed 4 Jun. 2018].
- Wikipedia. (n.d.). *Transmitter hunting*. [online] Available at: https://en.wikipedia.org/wiki/Transmitter_hunting [Accessed 6 Jun. 2018].