NaviGrip: A System for Indoor Positioning and Navigation

Marlborough College
Intel Corporation

Contents

1 PROBLEM BRIEF

1.1 PROBLEM

Intel is the world's largest semiconductor manufacturer, with over 200 different sites worldwide. Some sites/buildings are so huge they take over 20 minutes to walk from one end to the other, and it can be very challenging to find the desired location of a meeting room or work area. It can initially be difficult for new employees on their home site or especially difficult for employees travelling to a new unfamiliar site, wherever that may be in the world. Also the sites are dynamic and constantly changing and redeveloping. Much valuable time can be lost trying to navigate these huge complex buildings and this can make you late for meetings and reduce your efficiency. The European Headquarters in swindon caters for approximately 800 people and consists of 3 interlinked buildings spread over 3 floors, with multiple office areas, labs, meeting rooms, canteen, kitchens, games room and breakout areas, so can be very challenging to find your way around.

1.2 REQUIREMENTS

We must create an autonomous self-guided system that allows employees to easily navigate and locate a required destination in the Intel Swindon building. This project is designed to showcase one possible application of the Geniuno 101, and the Intel Curie module it is based on. Our key considerations are:

1. Ease of use
2. Low cost
3. Small size
4. Robustness

2 ORGANISATION

2.1 THE TEAM

The scheme was organised by Simon Flatres at Marlborough College, and the team consisted of six lower-sixth form students:

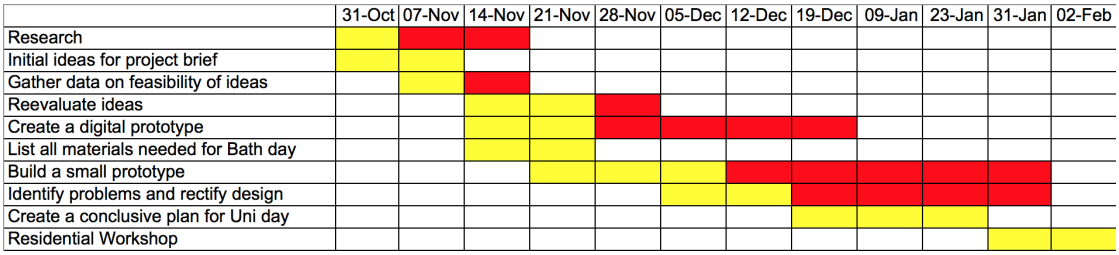
- Simran Chowdhry
- Jack Kirkwood
- Tuomas Laakkonen
- Isabella Nicholson
- Freddie Moorhead
- Alexander Rowe

We were assisted by five engineers from Intel Corporation:

- Matt Walker
- James Hawkes
- Ben Cummings
- Richard Goldman
- Connor Mcloughlin

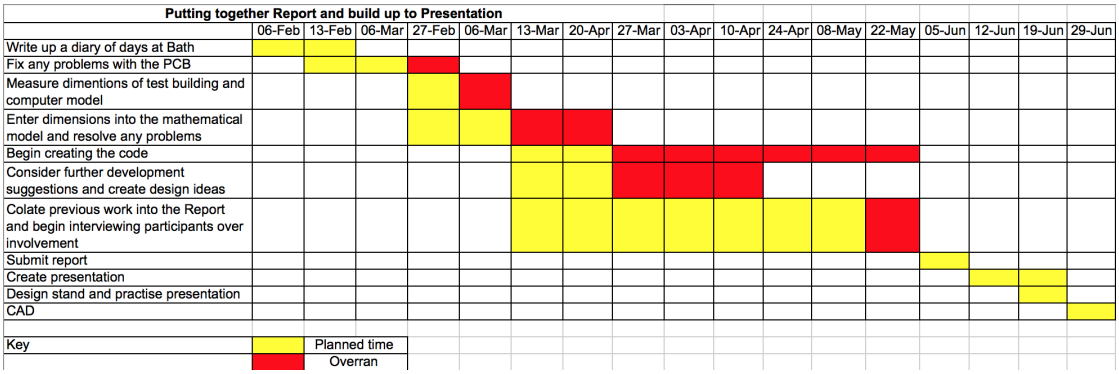
2.2 TIME PLAN

A Gantt chart was used in order to make sure that during this process the team remained on track.



	31-Oct	07-Nov	14-Nov	21-Nov	28-Nov	05-Dec	12-Dec	19-Dec	09-Jan	23-Jan	31-Jan	02-Feb
Research												
Initial ideas for project brief												
Gather data on feasibility of ideas												
Reevaluate ideas												
Create a digital prototype												
List all materials needed for Bath day												
Build a small prototype												
Identify problems and rectify design												
Create a conclusive plan for Uni day												
Residential Workshop												

Figure 2.1: The initial Gantt chart used for time management.



Putting together Report and build up to Presentation	06-Feb	13-Feb	06-Mar	27-Feb	06-Mar	13-Mar	20-Apr	27-Mar	03-Apr	10-Apr	24-Apr	08-May	22-May	05-Jun	12-Jun	19-Jun	29-Jun
Write up a diary of days at Bath																	
Fix any problems with the PCB																	
Measure dimentions of test building and computer model																	
Enter dimensions into the mathematical model and resolve any problems																	
Begin creating the code																	
Consider further development suggestions and create design ideas																	
Colate previous work into the Report and begin interviewing participants over involvement																	
Submit report																	
Create presentation																	
Design stand and practise presentation																	
CAD																	
Key		Planned time															
		Overran															

Figure 2.2: The second Gantt chart used for time management after we finished the residential.

During the course of the project the team kept closely to the chart and was often ahead of schedule. The only times the team was seriously behind schedule was whilst printing the first prototype, and whilst we were learning the 3D design programs that we used to create our digital prototype. The printing issues were due to the 3D printer being broken at school and it was necessary to spend a few days fixing it before printing could begin. The report could not be completed or started before the residential as much of what was being written took place at Bath University.

2.3 FINANCE

Marlborough College and Intel funded our project, it is fairly low cost, given the quality, quantity and nature of the components. They were bought primarily from Adafruit through Maplin, and Intel provided the team with Genuino boards for free which was the foundation on which the rest of the project was based.

Interim Financial Figures

					£
Total Capital Investment					
Production Costs	**Type**	Link	**Number**	**Unit Cost**	**Total Cost**
Wifi Module	ESP8266	Here		£2.00	£2.00
Screen	2.8" TFT / Cap Touch	Here		£15.00	£15.00
Compass	HMC5883L	Here		£9.00	£9.00
Barometer	BMP180	Here		£8.00	£8.00
Battery Charger	TPS61090	Here		£12.00	£12.00
Battery	2200mAh 3.7v LiPo	Here		£8.00	£8.00
PLA plastic			100cm^3	£1.98	£1.98
Shipping					
				£55.98	
Total Costs of Production					**£55.98**
Total costs					
Total profit					
Sales			**Number Sold**	**Sale Price**	
Navi Grip				£65.00	
Total revenue					
Total Income					

Figure 2.3: A breakdown of the prices for each component.

3 RESEARCH

3.1 INITIAL THOUGHTS

The first steps were to come up with ideas on how to track a person's position with enough accuracy that we could distinguish between floors and rooms. On this bases the traditional GPs tracking systems are no accurate enough for the application that we want. The initial ideas included:

- Wi-Fi positioning system
- RADAR
- RFID
- Bluetooth
- Accelerometer
- Barometers

3.2 FEASIBILITY

In order to decide which positioning systems was going to be used, research was done into each of the different types of tracking methods. From this a feasibility study was created for each option in order to narrow down the best, and consequently the ones that would be used. The tracking methods were broken into 3 different types.

3.2 EXACT POSITIONING

GPS

Range Global

Accuracy Works poorly within buildings.

Cost Low.

Wi-Fi

Range Less that 70m

Accuracy If properly calibrated using sophisticated triangulation methods, around 1m.

Cost Low.

Cellular Signal

Range Less than 45 miles.

Accuracy Time of flight methods are unusable due to the low resolution of the device, other methods produce very bad estimates (around 1 mile).

Cost High, as a SIM-card needs to be used.

RFID

Range About 1m.

Accuracy Can only tell if a device is within range, not how far it is.

Cost High, as recievers with a range as high as 1m are only available industrially.

Bluetooth

Range Less than 10m.

Accuracy Around the same as Wi-Fi if more than five metres from the source, otherwise about 10cm.

Cost Low.

Overall when considering the exact positioning methods, cost is a big factor in the feasibility study as well as the range of positioning. For those reasons Wi-Fi is the best exact positioning method to use.

3.2 VISUAL METHODS

Environment Markers

Range Pre-determined areas.

Accuracy Can identify which area you are in.

Cost Low.

QR Codes

Range 5m, if not obscured by people or other objects.

Accuracy Highly accurate when not obscured, about 10cm.

Cost High, as a camera is required.

Lighting

Range Pre-determined areas.

Accuracy Can determine which area you are in, assuming that the lighting is not obscured by other objects. The device must be held in a specific orientation for this to work. However, this will not work well at Intel's Swindon building as the lights are on a timer.

Cost Low.

Depth Images

Range About 10m.

Accuracy About 20cm, although only if the environment is constant - people and other objects will severly impact accuracy.

Cost High, an IR camera and projector are required.

Overall, people are a problem for most of the visual methods as they decrease accuracy and the environment is constantly changing. However, QR are very certain, but they require permanent codes, which would make the Navi Grip less transferable as a product.

3.2 MOVEMENT METHODS

Step Counting

Accuracy Relatively accurate (i.e 2m) if the person maintains a constant gait and the step size is accounted for. This only provides distance, not direction data.

Cost Low.

QR Codes

Accuracy Very accurate when isolated, but may be affected by nearby magnetic objects. Only provides direction data.

Cost Low.

Lighting

Accuracy Decreases with time from a precise starting position - provides both distance and direction data. Can also account for the vertical dimension if gravitational acceleration is ignored.

Cost Low.

Overall, the movement methods are very useful as an accompanying method of tracking position but they couldn't be used just on their own. They can provide a general direction and distance of a point relative to the previous position, but can't be used to pinpoint an exact location, as they lose accuracy with time and can only be as accurate as the previous position. However, these are useful, as they are not limited in range, but only in accuracy.

3.2 COMBINATION METHODS

In order to combine different positioning methods, a suitable method is required. The simplest of the methods is the naîve Bayesian method. This uses statistics to find which position is the most probable, given all the data we collect. The advantages of this method are that it is continuous - it does not require that we divide the building up into a grid - and that it is easily correctable - if it finds a wrong prediction which is then proven inaccurate with further data, it can correct its position using the new data. The disadvantage of this method is that it requires complicated decimal arithmetic, which is slow and may have rounding errors. The second method is the branching method, which maintains a list of possible locations of the user, and then eliminates some when they are proven impossible. This method is not continuous which means it can be implemented using only integer arithmetic, improving performance, but it also

uses more memory (a precious resource on a system with only 32kb of RAM), as it has to maintain a list of possible locations. It is also easily correctable, in the same way as the last method. The final and most efficient method is the Hidden Markov Model based method. This uses a Hidden Markov Model to find the most probable state following some data being collected. This works almost the same as the branching method, except it only keeps the most probable state at each iteration. While it does use decimal arithmetic, it calculates the probabilities more efficiently than the naîve bayesian method. The main disadvantage of this method is that if it makes a wrong decision at any point, it is impossible for it to go back and correct its mistake. As such, the best method to use is the naîve bayesian method, as it is easily correctable, but also passably efficient in terms of runtime, and very efficient in memory usage.

4 INITIAL DESIGN

4.1 HARDWARE

Design was focused on usability, functionality and with working on a timeline. It was primarily done using a variety of CAD programmes and graphic design programmes.

There were five different design ideas; a drone, a handheld device: either a touchscreen or a different ergonomic gripped one, light strips, a scannable code card and finally a self moving "car." Of these the car and the drone were unfeasable; too difficult to make and completely unpractical.

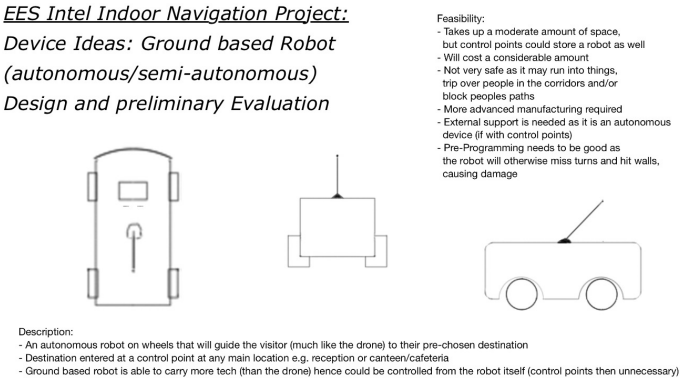


Figure 4.1: Initial robot based design

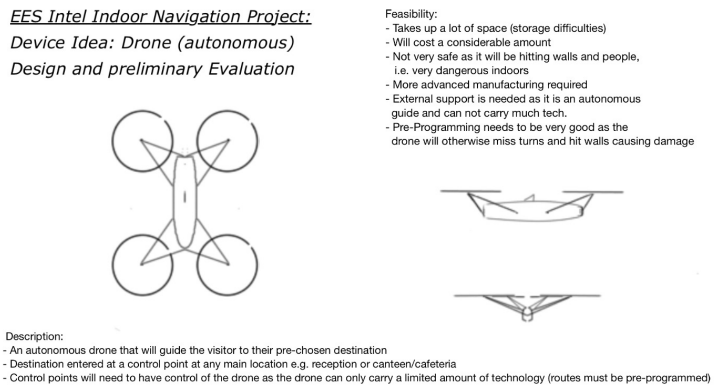


Figure 4.2: Initial drone based design.

The remaining options included the scannable QR code or a RFID card in conjunction with a screen based device which would give driections to a place in the building based off of where the scan areas were, this would allow for extremley good accuracy and would be very easy to set up for any building. However, it would be not be user friendly as it would be up to the person to find the scanner and then scan the code perfectly or hold the card up to the scanneras the range is less than a half meter. This would be cumbersome and would also cause people to possibly even have to wait as multiple people scan at one point.

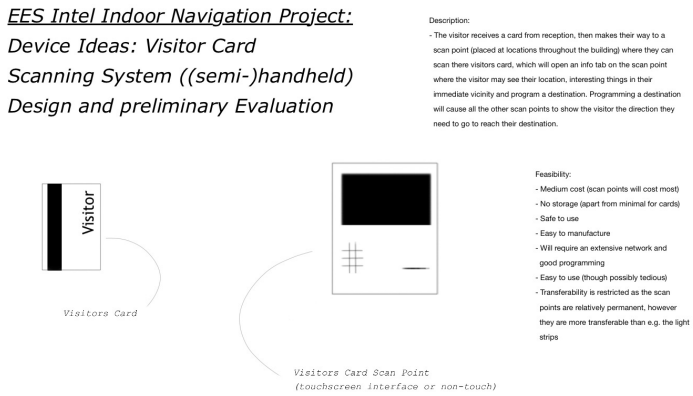


Figure 4.3: Initial scanner and card based design.

Another of the ideas which remained was a series of light strips either on the wall or on the floor which people could follow depending on the color of light they were told to follow. This proved not usuable as the combinations of colors for multiple people wouldn't work and the likelihood of people having to use the light strip at the

same time was too high. Multiple light strips could also be used but people would very easily get confused, especially at four-way hallway junctions. It would also guarantee collisions as people watched the lights and not where they were going.

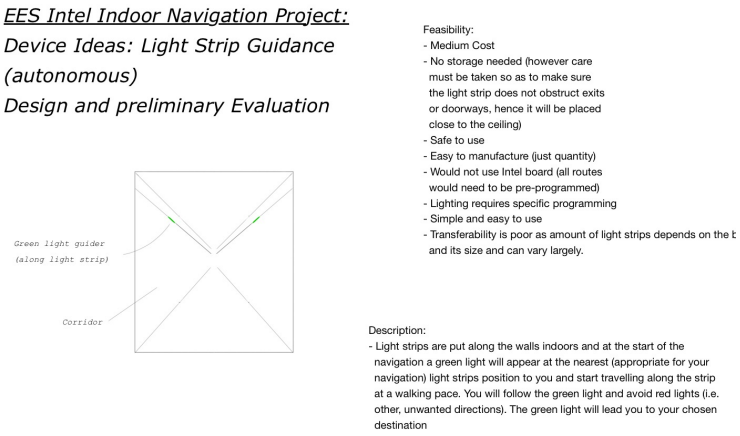


Figure 4.4: Initial light strip based design.

The final idea was the handheld device, working off of triangulation, either from Wi-Fi or bluetooth. This was far more difficult to make than the RFID/QR code, but it was much more user friendly and could be made to be very accurate - within a meter. This method would use a screen to display directions depending on the position of the user and could be reprogrammed on the go in case the user's destination changed.

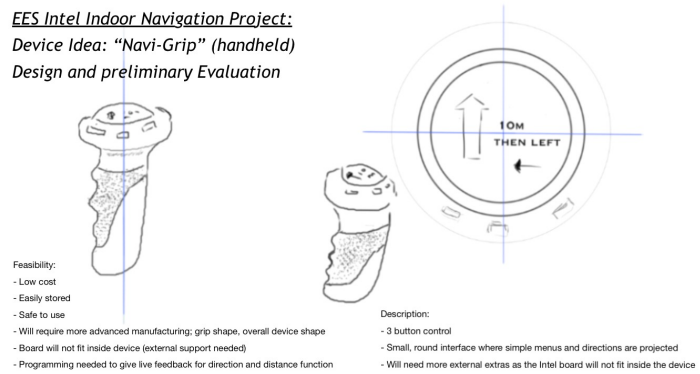


Figure 4.5: Initial handheld device based design.

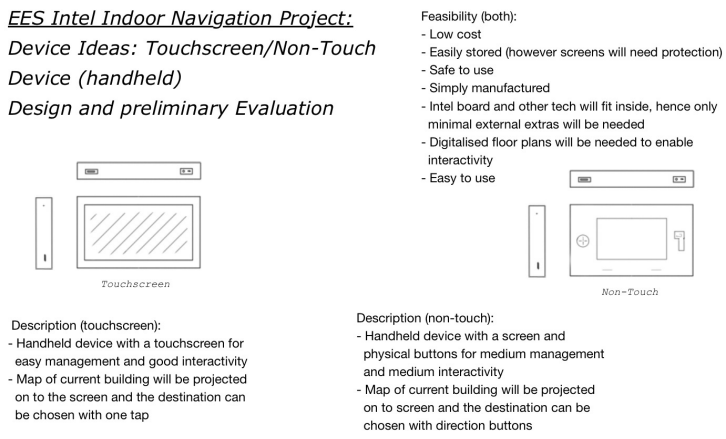


Figure 4.6: Designs showing operation with or without a touchscreen.

EES: Intel Indoor Navigation Project: Feasibility regarding different Device Ideas

Feasibility Criteria	**Device (autonomous)**			**Device (handheld (HH))**			
	Drone (airborne)	*Robot (ground)*	*Light Strip Guidance*	*HH Touchscreen*	*HH Non-Touch*	*HH "Navi-Grip"*	*Visitor Card (QR-Scan)*
Cost	- -	-	+ -	+ +	+ +	+	+ -
Storage	- -	-	+ +	+ +	+ +	+ +	+ +
Safety	- -	- -	+ +	+ +	+ +	+ +	+ +
Ease of Manufacture	- -	- -	-	+	+	+ -	-
Compatibility with Intel board	-	+	N/A	+	+	- -	N/A
Ease of Programming	- -	-	- -	+	+	+ -	+
Ease of Use	-	+ -	+ -	+ +	+ +	+ +	+ +
Overall complexity	-	-	-	+ +	+ +	-	+
Transferability	+ -	+ -	- -	+ +	+ +	+	-
Amount of external extras needed (routers, etc.)	-	+ -	+ -	+ +	+ +	+ -	- -
Possibilities Indoors	- -	-	+ +	+ +	+ +	+ +	+ +
Interesting	+ +	+ +	+ +	+ -	-	+ +	+ -

Key:

Meaning	Very Poor	Poor	Acceptable	Good	Very Good
Symbol	- -	-	+ -	+	+ +

Figure 4.7: Assessment of the feasibility of all our initial ideas.

4.1 HARDWARE DESIGN DECISIONS

In the end the handheld device proved to be the most appropriate. An accelerometer, barometer and a magnetometer were added in order to increase accuracy of the postioning. Finally, when the components had arrived, it was dicovered that the bluetooth component was not a reciever but an emitter, this does not allow it to connect to multiple bluetooth recievers at once, so for triangulation it would connect one at a time and there would be major time delay for both positioning and directions. As such, a Wi-Fi module was included instead, to give the same functionality as bluetooth. The updated designs for the NaviGrip were drawn in a graphic design app and also with 3D visualisation software. The touchscreen was incorporated into this later on.

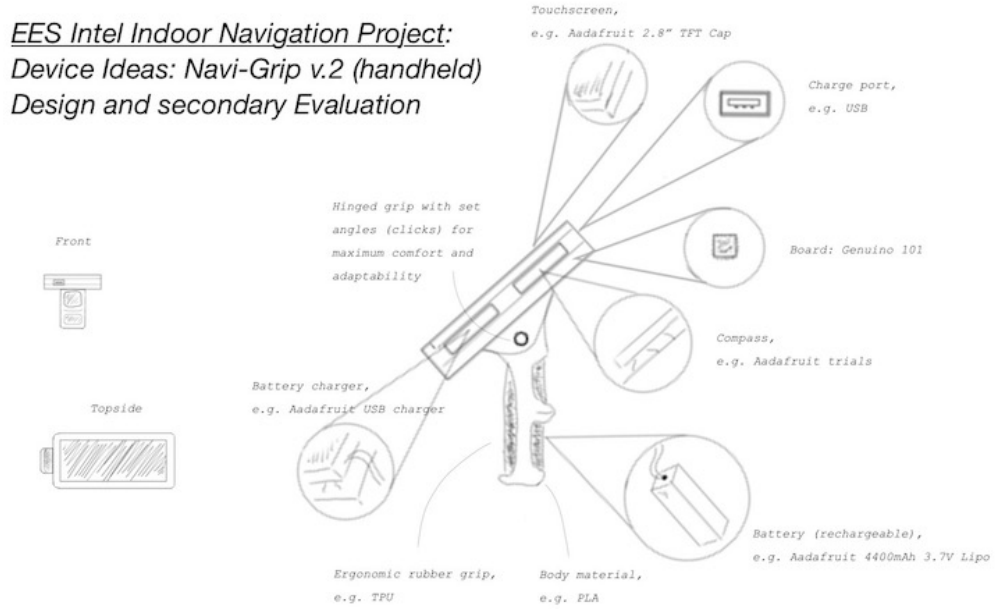


Figure 4.8: Final handheld device design.

4.2 MATHEMATICAL MODELS

Two different mathematical models were initially created, using classical and quantum electrodynamics. Eventually, it was decided that the classical method would be used, as both methods produced the same results, and some of the assumptions made in our quantum derivation would not hold at large scales.

4.2 QUANTUM METHOD

One of the initial mathematical models that had been used assumed the photon model of light, and calculated loss of signal using the linear attenuation coefficient, which is derived from Beer and Lambert's laws.
Lambert's Law states:

> Absorbance of a material sample is directly proportional to its thickness (path length).

Beer's Law states:

> Absorbance is proportional to the concentrations of the attenuating species in the material sample.

GENERAL SOLUTION

$$I(x) = I_0 e^{-\alpha x}$$

For:

$$\alpha = \mu\rho$$

where μ is the mass attenuation coefficient and ρ is the density of the material.

DERIVATION

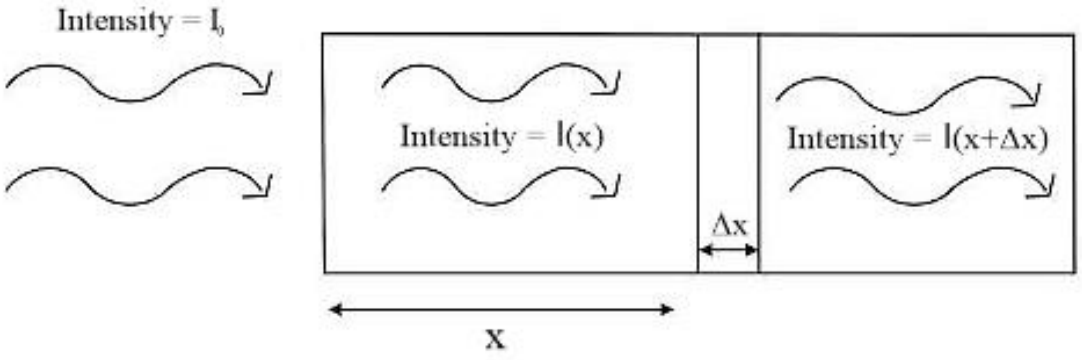


Figure 4.9: Signal attenuation in a species.

Assuming light of initial intensity I_0 passes through a material of length x, and a second material with horizontal thickness Δx (calculated in relation to x). The intensity

through the first material can be written as $I(x)$ and after second material as $I(x-\Delta x)$. Therefore,

$$I(x) - I(x - \Delta x) = \alpha I(x)\Delta x$$

Multiplying both sides by -1:

$$I(x - \Delta x) - I(x) = -\alpha I(x)\Delta x$$

Assuming with variation to x, Δx tends to zero:

$$\lim_{\Delta x \to 0} \frac{I(x - \Delta x) - I(x)}{\Delta x} = -\alpha I(x)$$

$$\frac{dI}{dx} = -\alpha I$$

Therefore, upon integration:

$$I = I_0 e^{-\alpha x}$$

and conversion to decibels:

$$I^{db} = I_0^{db} - 10 \log(e)\alpha x$$

LIMITATIONS

Though simple, this model could not be used as the Beer-Lambert law has certain limitations:

- At a high concentration, the mass attenuation coefficients deviate due to the electrostatic interactions between molecules in close proximity and scattering of light due to particulates in the sample.

- Moreover, this does not account or diffraction in visible light wavelenghts, nor does it include signal loss due to partial reflection and refraction. By simply taking the photon model of light, the properties of light that cause it to exhibit wave-like nature are completely ignored, hence not showing the most accurate representation of signal loss.

- Another assumption essential to the derivation of the equation is that Δx tends to zero, which is not true, for a wall, which is what the model is for.

4.2 CLASSICAL METHOD

To determine the location of a person inside a building, a combination of methods are used. For our Indoor positioning system, we are using a Barometer to detect pressure differences, which acts an indicator for which floor the device is on, an accelerometer to determine the distance moved by the device in the horizontal plane, and wifi interference to determine exact location.

Precise location can be calculated using wifi by applying triangulation in the horizontal plane. By taking three labelled hotspots and predicting their individual signal strengths at a certain point. This combination is unique for any given point on the map of a particular floor and location can be detected with a marginal error of about 2 feet.

In order to predict signal strength, attenuation of signal (signal loss) needs to be calculated, both over a free space (free space path loss) as well as through walls (using Fresnel's Equations as well as Maxwell's equations to understand the dielectric properties of the materials in the bricks of the wall). The loss of signal can be mechanically calculated by calculating dielectric loss, free space path loss and variable attenuation in irregular materials.

DIELECTRIC LOSS

Dielectric loss can be calculated using the following equation-

$$P = P_0 e^{-\delta k x}$$

where $k = \frac{2\pi}{\lambda}$ is the wave number, and λ is the wavelength of light used

In decibel form, this is:

$$\log_{10}(P) = \log_{10}(P_0) + \log_{10}(e^{-\delta k x})$$

$$P^{db} = P_0^{db} + \log_{10}(10^{-\delta k x \log_{10}(e)})$$

Therefore,

$$P^{db} = P_0^{db} - \delta k x \log_{10}(e)$$

where:

- e is Euler's constant (2.718)
- P is the final power of the signal
- P_0 is the initial power
- δ is the dielectric loss tangent

- P^{db} is the final power in decibels
- P_0^{db} is the initial power in decibels

δ can be measured experimentally using an oscilloscope, or can be calculated via weighted average of known values of δ for more complicated structures.

LOSS DUE TO CHANGE IN MATERIALS

This can be calculated using Fresnel's equations. By calculating refelctance, which is the ratio between the refracted (final) intensity and the incident (initial) intensity. This is particularly useful when the percentage composition of the irregular material is known, but the distrubtion within the material is not.

Reflectance can be calculated using;

$$R = \left| \frac{\sqrt{\epsilon_2 \mu_2} cos\theta_i - \sqrt{\epsilon_1 \mu_1} cos\theta_t}{\sqrt{\epsilon_2 \mu_2} cos\theta_t + \sqrt{\epsilon_1 \mu_1} cos\theta_i} \right|^2$$

where:

- μ_1 and μ_2 are the magnetic permeabilities of the two materials
- ϵ_1 and ϵ_2 are the electrical permittivities of the two materials
- θ_i and θ_t are the angles of incidence and transmittance respectively, to the normal in the plane of reflection.

SIGNAL LOSS THROUGH FREE SPACE

Free space path loss refers to the attenuation of signal when passing through air, due to factors such as scattering and diffraction.

$$FSPL = \left[\frac{4\pi df}{c} \right]^2$$

to express the equation in decibels:

$$\begin{aligned} &= 10 \log_{10} \left[\frac{4\pi df}{c} \right]^2 \\ &= 20 \log_{10} \left[\frac{4\pi df}{c} \right] \\ &= 20 \log_{10}(d) + 20 \log_{10}(f) + 20 \log_{10} \frac{4\pi}{c} \end{aligned}$$

where:

- d is the distance from the transmitter.
- f is the signal frequency
- c is the speed of light in air ($2.99792458 \times 10^8 \,\mathrm{ms}^{-1}$)

5 FINAL DESIGN

Our final device design has the following major components:

- A 240 by 320 pixel resistive touch screen for its user interface, from Adafruit, which has a micro-SD card reader mounted to the back of it, allowing the Genuino to store large files.
- A Bosch BMP180 barometer, an LSM303U combined accelerometer and magnetometer, and an ESP8266 Wi-Fi reciever to allow the device to position itself.
- A 2400mAh battery and a charging circuit (from Adafruit), powered by a micro-USB connector that allows the device to be charged without the removal of the battery.
- This is all powered by a Genuino 101 board, based on the Intel Curie module.

5.1 SUSTAINABILITY

The NaviGrip was not designed with sutainability particularly in mind. It is generally too small to be traditionally sustainable in the way that it is powered; a touchscreen and the navigation software need more power than a solar panel of apropriate size could provide, a larger panel may work to charge several while they were docked. It is also too small to contain a dynamo to generate is own power. It does use a lithium ion battery which is not sustainable but it does not really ever need to be replaced as it gets charged. The materials we used to make it were ABS and TPU for the body and copper and semi-conductors for the electrical components. The plastics not made from renewable sources but can be recycled or melted down and reused as they are not thermosetting. The PCB is made from plastic with a copper plating and, once etched,

is not reusable. Finally, it is a product that is intended to be used many, many times over the course of may years, not something which only works once so the product itself is reusable.

5.2 CASE DESIGN

Using our original designs as a base, we started to see what we could do with the modeling software, Space Claim.

5.2 TRIAL RUN

This design was created with the limits of the Makerbot Replicator 3D printer in mind. For the first few runs, just the handle was made.

Figure 5.1: The original trial handle design.

This was printed out of PLA with a wall thickness of 5mm. Such thin walls led to the problem of the handle flexing when gripped so it was decided that it would be solid apart from an area for the components to go.

5.2 MARK 1 AND 2

After the trial run efforts were focussed on building a model with the limits of the printer no longer taken into consideration as the printers at Bath University could print almost any conceivable shape.

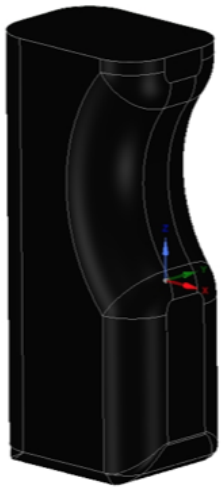


Figure 5.2: The second handle design, that is the base of all our further designs.

5.2 MARK 3

The mark 3 was developed directly from the mark 2. It now has a articulating lobe on the top for for the screen to move on.

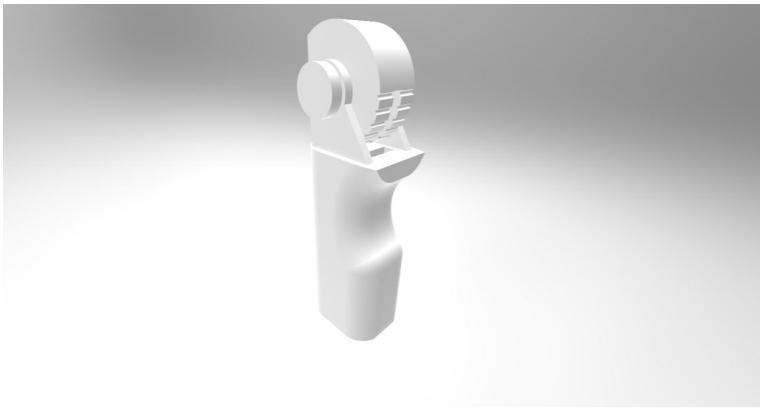


Figure 5.3: The first handle design with consideration for the mounting of the screen.

The image below shows the mark 3's handle cross section and base cover. This is how it was split up for printing.

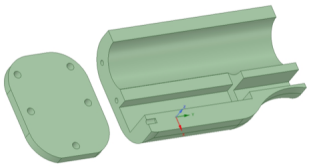


Figure 5.4: Expanded view of the Mark 3 handle.

5.2 MARK 4

The Mark 4 was the final attempt to bring everything together for printing. We added a dock and the screen housing, specifically designed to fit all the components. These were then printed and the results are below. The printed parts will then be used for the prototype NaviGrip.

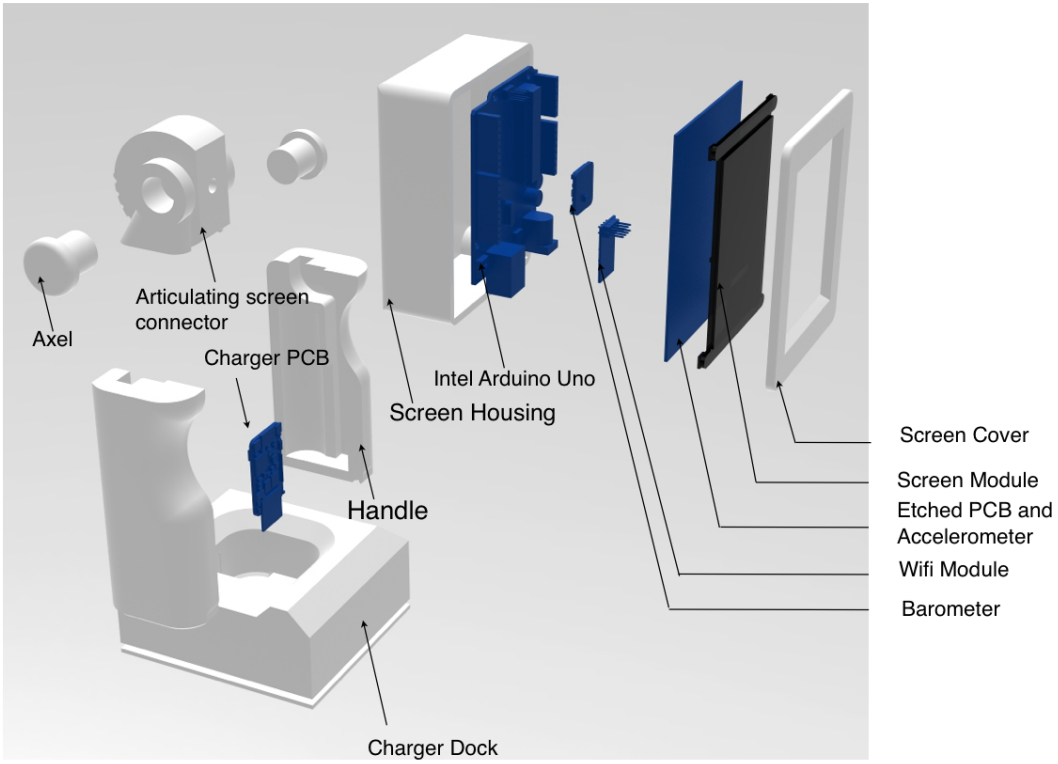


Figure 5.5: Expanded view of the whole Mark 4 design.

Figure 5.6: View of the Mark 4 showing screen articulation.

This trimetric view demonstrates the articulation of the screen as well as how the components are placed inside the device. The screen and the components inside that module are held in place by long screws which screw into the base. The power cable from then handle comes through as shown:

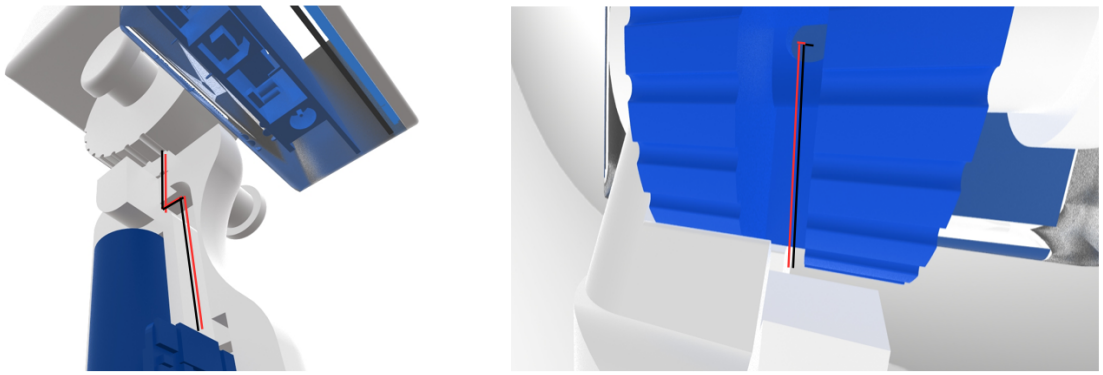


Figure 5.7: View of the power wire placement inside the screen mount.

5.2 CONCEPTUAL DESIGN

We wanted to demonstrate what a device like the NaviGrip could become in the future, if the internal components were made for the device rather than the device made for the components.

5.2 NAVIGRIP 2.0

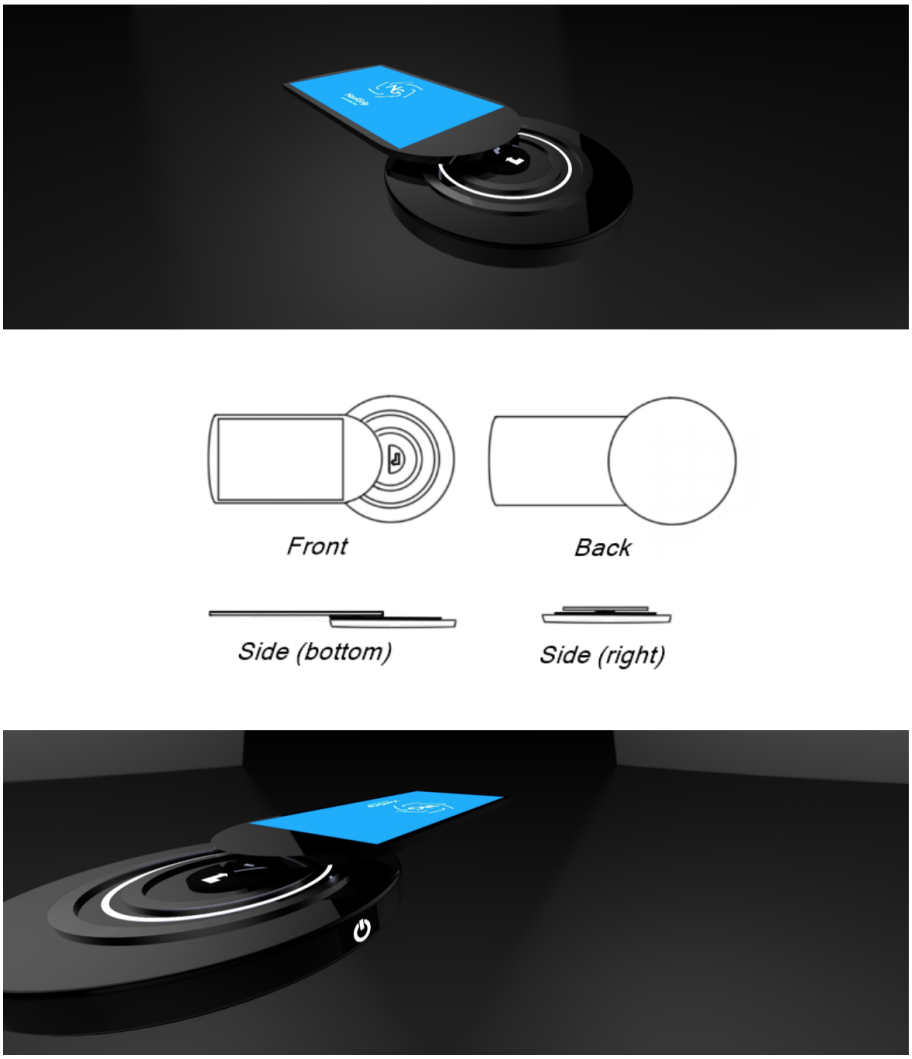


Figure 5.8: Design of the NaviGrip 2.0.

DESIGN AND DESCRIPTION

The NaviGrip 2.0 was actually the second iteration of the concept phase.

The device consists of a control pad of two ring selectors and an enter button. The rings are made of radial brushed aluminium and the enter button is made out of rubber.

A new way of interacting with the NaviGrip has been introduced with the two ring selectors. The outer selector is for wider control whereas the inner control is for the higher level selection. The light bar makes it easy to operate the device in the dark.

The wide LED screen, which is great for seeing your current direction at a glance, is mounted to the control pad through a piece of machined aluminium, anodized black, however, the rest of the housing is a glossy, black ABS plastic.

The slim design of the NaviGrip 2.0 enables it to be put away easily, e.g. into pockets, when it is not being used, as well as held with minimum effort, much like modern phones.

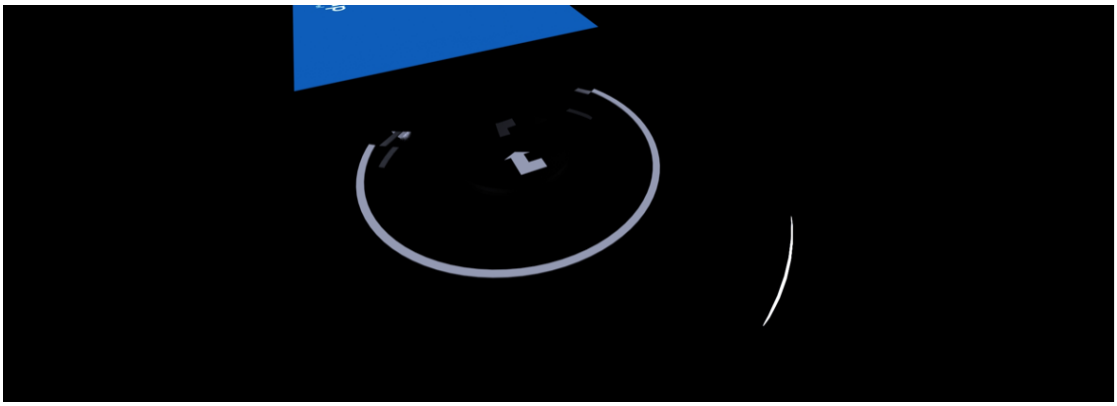


Figure 5.9: View of the controls on the NaviGrip 2.0.

The NaviGrip 2.0 would make the perfect device for any start-up company that would like to add flair and functionality to their business HQ. It is less expensive than the 2.0 Pro (next section) but there is no compromise on functionality and leaves nothing of need to be desired.

Just like the NaviGrip Pro, the NaviGrip 2.0 will be equipped with an interactive map which can easily be navigated using the ring selectors - one for vertical movement and one for horizontal.

A new way of interacting with the NaviGrip has been introduced with the two ring selectors. The outer selector is for wider control whereas the inner control is for the higher level selection. The light bar makes it possible to see the controls in a darker setting.

5.2 NAVIGRIP PRO

DESIGN

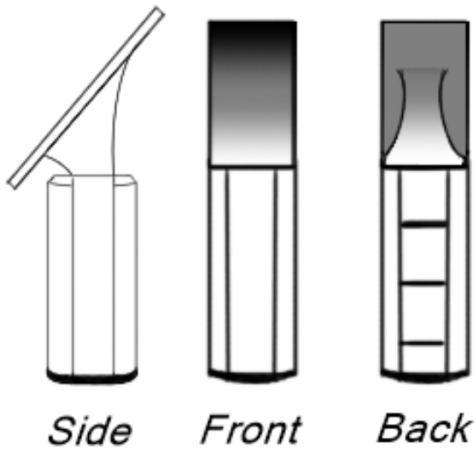


Figure 5.10: Diagram views of the NaviGrip Pro design.

The handle is constructed of milled aluminium and assembled in two halves to allow the components inside to be managed.

The neck is of the same material as the handle, however, milled from one piece, as is the screen support except with a space for the attachment of the OLED display.

The device is anodized with a colour of the clients specification, the default being 'Intel-blue' for the handle and 'Gold-Silver' for the remainder of the device.

DESCRIPTION

The NaviGrip Pro is an updated, fully revised version of its predecessor. The refined design provides the user with a functional tool that doesn't detract from its elegance.

The stunning OLED touch screen display allows for easy following of the navigation as well as the smooth use of an interactive map.

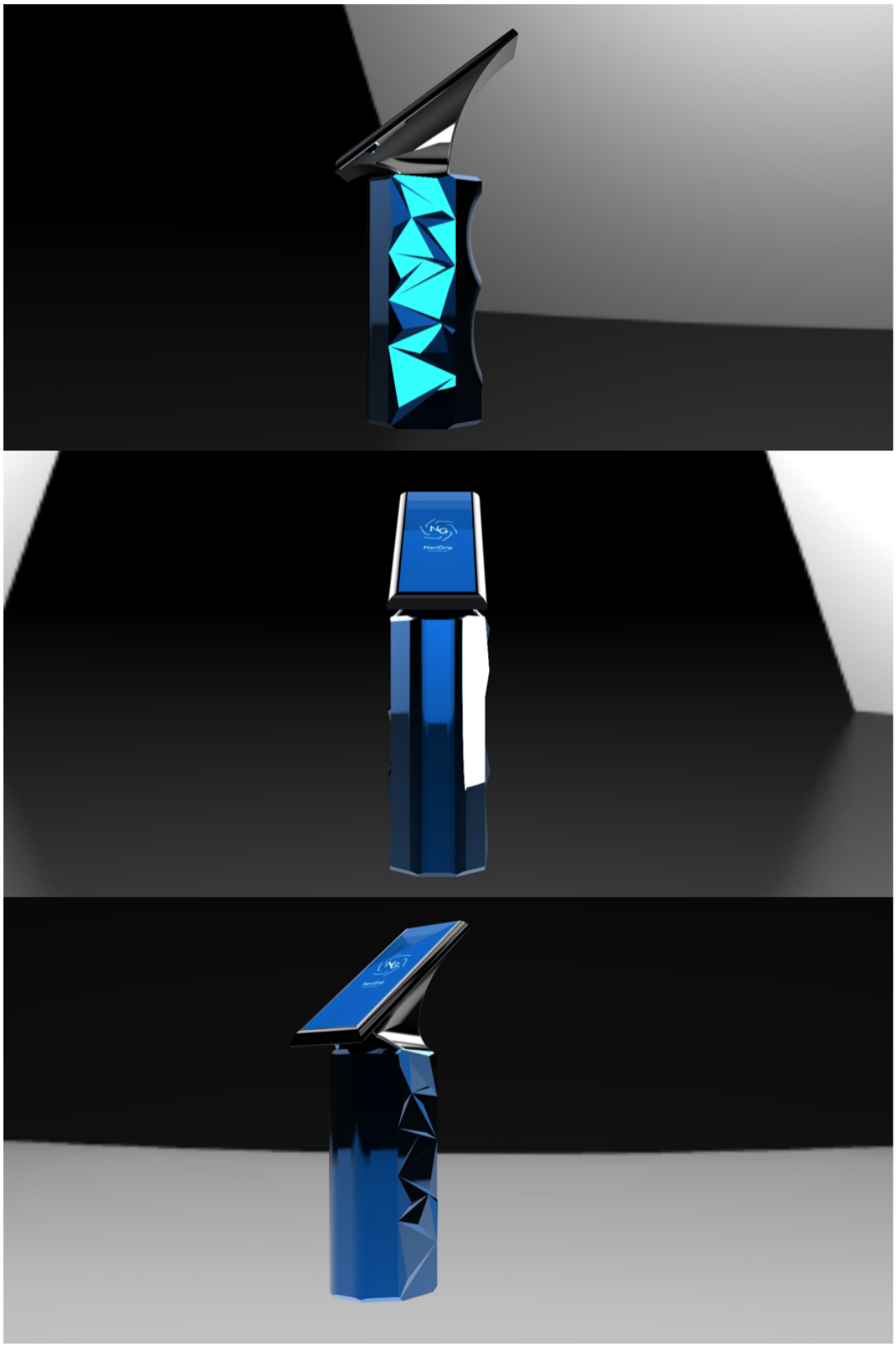


Figure 5.11: Three renders of the NaviGrip Pro in blue.

5.2 CONCLUSIONS

PROBLEMS AND POTENTIAL IMPROVEMENTS

The first major hurdle was learning the CAD (computer aided design) software. This was quickly overcome by simply making continuing iterations of the design. In doing this, ability no longer became the limiting factor but instead it was the limits of the printer.

The 3D printer at school is the Maker Bot Replicator. 3D print technology evolves rapidly, the latest tech being a printer that 'pulls' the sculpture out of a liquid layer by layer or water soluble filament to dissolve the supports. Once it was decided that all printing would be done at the Bath Uni engineering labs, design could quickly get more complex and we could actually have very few limits.

We experienced some software and hardware limitations. Theoretically, if brought to an industrial stage, more computing power would be welcomed as well as better programs for modeling and rendering.
The programs we used were:

- ANSYS SpaceClaim 2016 - for modeling
- Keyshot 5 - for rendering

Had we had more time to learn, we would have chosen,

- SolidWorks
- Adobe After Effects

The problem with rendering on a laptop is that it has far more cooling issues as well as a CPU that although is more than average for a laptop, is not up to the standard of complex animation rendering. One solution to this might be to outsource rendering to a network where a more powerful computer can be used in the meantime but the long term solution is to invest in a desktop computer that has the necessary horsepower to do such complex renders.

Another problem when designing is the durability of the material you are using. PLA is great for quickly prototyping ideas, but for building a device to last or that could actually be used a more robust model is needed. This is a possible future improvement.

All in all, what we got done was fine in the timescale and with the current resources but with the improvements outlined above, a much more streamlined workflow could be achieved and a better device enclosure could be developed.

5.3 HARDWARE DEVELOPMENT

5.3 COMPONENTS

All the components were sourced in *breakout board* form. This is where the actual integrated circuits we wish to use in out device are presoldered onto seperate PCBs with simple through hole headers soldered to them. This is useful because it allows easy prototyping using a breadboard, and also easy manufacture of the final device as we did not have the facilities to the surface mount soldering required by many of the raw components. If this device were to be produced commercially, a PCB could be easily produced that utilised the raw components instead. While these modules are significantly more expensive, they allow us to completely ignore several design issues that would occur - namely, the space required for surrounding circuitry - many of these components require auxilliary passive components for their operation, however, if we included these for all four sensors, we would not have enough space on our PCB, unless they were surface mount components, which we cannot solder. Secondly, it allows us to avoid any level-shifting circuitry - the most prominent example of this is the ESP8266 Wi-Fi module - this module runs on 3.3v logic instead of 5v like the Genuino 101, thus we need additional components to interface these two components (note that this issue would be circumvented in a commercial application - all the sensors run at 3.3v, as does the underlying Intel Curie module used in the Genuino 101, it is simply the design of the Genuino 101, that is backwards compatible with other 5v Genuino models, that forces us to do this).

These sensors each have different methods of communicating with the Genuino (protocols), and communicate over different pins on the Genuino (busses). On the I^2C bus, we have the barometer, accelerometer and magentometer, on the serial bus, we have the Wi-Fi module, and on the SPI bus, we have the SD card and the touch screen.

I^2C BUS

This bus is the easiest to implement, since each sensor can be connected to the same two pins on the Genuino. This is because each sensor is given a unique ID and every message sent by it is prefixed with this ID, thus allowing data packets from multiple sensors to be sent over the same bus. The only other component required is a pair of resistors pulling the data and clock lines to ground when not used.

SPI BUS

This bus requires four wires, data in, data out, clock, and chip select. This bus is capable of high speed, which makes it suitable for large data transfers such as SD card

reads and screen writes. In order to implement this, each device can be connected to the same wires (the Genuino 101 has hardware SPI support to enable even higher speeds), except for the chip select lanes, which must be unique to each sensor - since only one device can send data on the bus at any given time, the chip select lanes are used to control which device is sending and recieving data at any given time.

SERIAL BUS

This bus simply requires transmit and recieve lanes. Unfortunately, each device requires its own seperate serial lanes. In the first revision of our design, we had the Wi-Fi module using the hardware serial port on the Genuino, however we realised that this is also used for programming the Genuino with the computer, and so the Wi-Fi module was unusable while the device was connected to the computer. In order to solve this, the Wi-Fi module was connected to two other pins on the Genuino. We then emulated a serial port on these two pins in software, allowing us to use the Wi-Fi module at all times. While this method has a slower speed than hardware serial ports, the Wi-Fi module is only being used once every two seconds, so this is not speed critical.

5.3 PCB

The PCB (printed circuit board) was our method of mounting all the components, allowing them to be soldered to the board. The design for this was very simple, only serving to connect the busses leading to each sensor to the Genuino. The main advantage of the PCB was to give a solid platform to mount all of our components, and, crucially to be able to mount some of these components upside down, thus not letting the space between the PCB and the Genuino go to waste.

5.3 BATTERY

Our battery system uses a automatic charging circuit to charge a lithium ion battery, that is housed in the handle of the device. This charging circuit is manufactured on its own PCB so it is easy to mount inside the handle of the device by simply gluing it. Since the battery we had hoped to use was not available to us due to shipping restrictions from the US, we had to modify the design to include a larger battery. This also required us to manufacture our own battery connectors. For this we used conductive foil tape which was secured to the inside of the bottom battery cover on the device. This also allowed us to create contacts on the bottom of the device with foil tape through which it could charge. We developed a prototype dock station which would allow these contacts to be used.

5.4 SOFTWARE DEVELOPMENT

5.4 TOUCH SCREEN

The touch screen on our device is a *4-wire resistive* touch screen. The touch screen in this case acts like a two dimensional variable resistor, where the position of the users finger in the x and y directions increases the resistance between two x-axis and two y-axis wires respectively. This has an effective circuit diagram of:

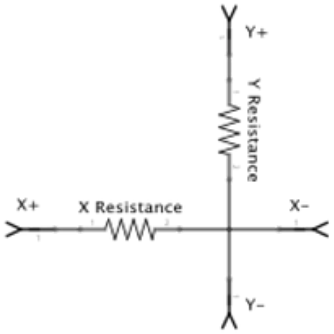


Figure 5.12: Effective circuit diagram.

Thus in order to measure the resistance, we setup up a potential divider, as follows. By taking positive-y to five volts and negative-y to ground, leaving, positive-x in a high impedance state and reading the voltage from negative-x. This allows us to calculate the resistance in the y-axis. A similar method is used for the x-axis, reading from negative-y, and letting positive-x be five volts, negative-x be ground and positive-y by high impedance.

In order to detect whether the user is touching the display or not, we examine the resistance values. If they are outside of a threshold, then the user is not touching the display, as the resistances are high when there is no pressure on the display. Additionally, to reduce noise in these data values, we use a digital low-pass filter, based on exponential smoothing. This filter is a versatile and has low computation and memory requirements, in contrast to filters such as the moving average, which requires much more memory. This filter is implemented according to the following equation:

$$y_i = y_{i-1} + \alpha(x_i - y_{i-1})$$

where y_i is the current output, y_{i-1} is the previous output, x_i is the current data value, and $\alpha \in (0, 1)$ is a parameter that is related to the cutoff frequency of the filter - higher values of α reduce the cutoff frequency, allowing less noise. This is known as

an exponential smoothing filter, as the contribution of previous output values to the current output value follows an exponential function.

5.4 USER INTERFACE

The device is equipped with a touch screen, that serves as the only interface with the user. This touch screen is used to display different images that can be interacted with by the user. These images are organised as *states*. The device has some state, and the required image for that state is displayed on the screen. When the user touches the screen, or at regular intervals, the device can change state, depending on the state it is in, and any user input that has occured. This system is known as a *finite state machine*. In this device, we have many states:

- The startup state, shows the device logo. This is simply a filler state, only providing an image that automatically transitions to the welcome state when the image is drawn.

- The welcome state shows an image welcoming the user to the device. This is also a filler state, transitioning to the first menu state immediately.

- The first menu state allows the user to select which building the wish to navigate to. Once the user has touched the drop-down box, three buttons are displayed allowing them to select the building. Once the user does this, the device transitions to the second menu state.

- The second menu state allows the user to select which floor they wish to navigate to, but is otherwise identical to the first menu.

- The third menu state allows the user to select which room they wish to navigate to. This is accomplished by the allowing the user to select two digits from a number pad which is displayed.

- The loading state is then shown, which gives the device time to prepare the navigation state, loading the list of shortest paths from the SD card, as well as initializing the Wi-Fi, accelerometer, magnetometer and barometer. When these have loaded, the system moves to the navigation state.

- The navigation state is described in the next section.

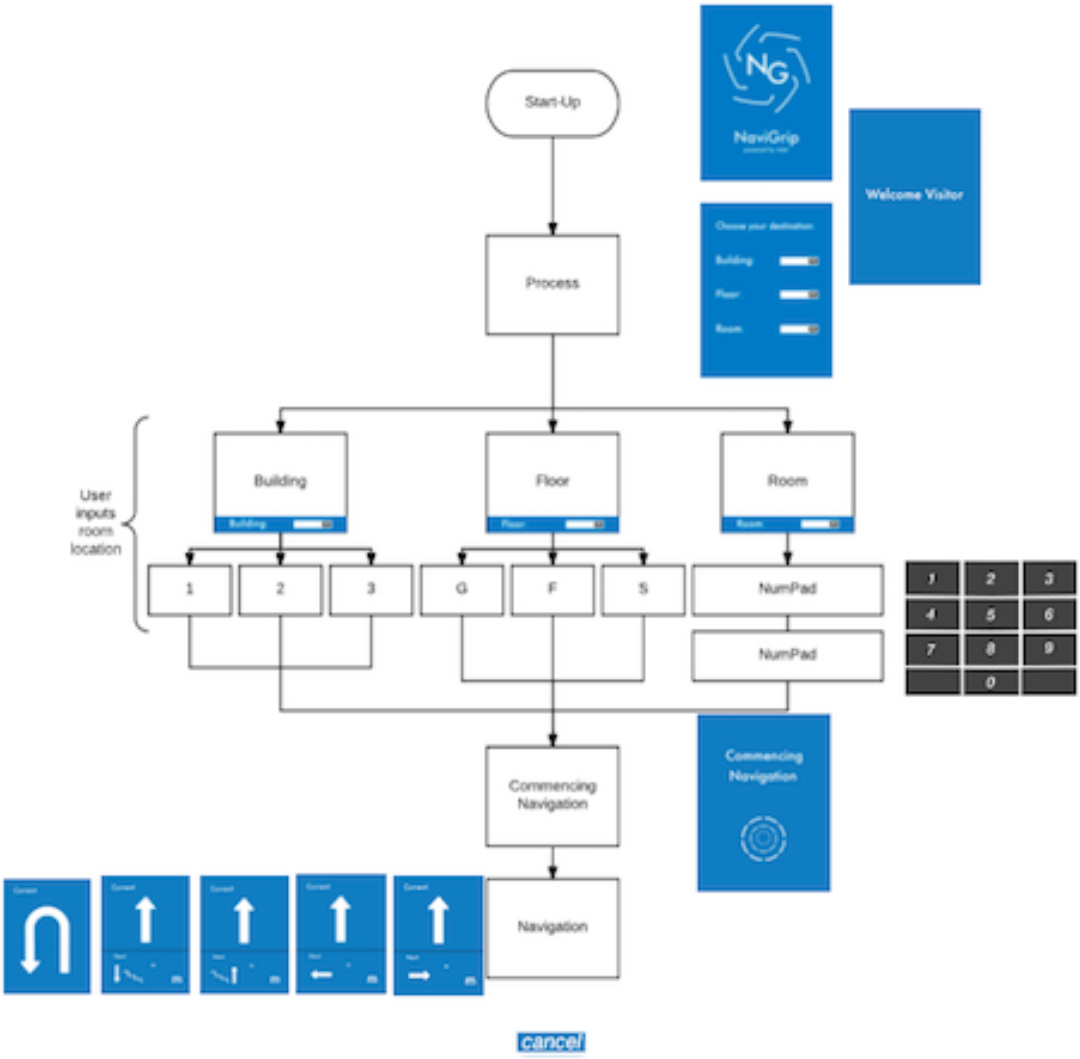


Figure 5.13: A diagram of all the states and their transitions.

5.4 NAVIGATION SYSTEM

When the device has gone through its menus and has started navigation, the following procedure is followed:

- Using the list of shortest paths, the path to be followed is selected.
- At each node, we display the direction and distance to the next node, as well the next instruction. We update this distance to the next node by projecting our position vector, relative to the current node, onto the vector from the current node to the next one. This is done using the equation:

$$d = |\mathbf{n}_c - \mathbf{n}_n| - \frac{(\mathbf{p} - \mathbf{n}_c) \cdot (\mathbf{n}_c - \mathbf{n}_n)}{|\mathbf{n}_c - \mathbf{n}_n|}$$

  where $\mathbf{n}_c$ is the current node, $\mathbf{n}_n$ the next node, and $\mathbf{p}$ the current position vector.
- We update this position and thus distance as quickly as the screen and Wi-Fi module will allow.
- At each update we also check the distance away from the current path, as opposed to how far along the path we are, by using the formula

$$w = \sqrt{|\mathbf{p} - \mathbf{n}_c|^2 - d^2}$$

- If this becomes to large, we check what the closest node to the user is, and if we are within a threshold of a node that is not the current or next node, then we assume the user has gone off course, and we restart the navigation procedure, from this node to the endpoint.
- When we reach our destination node, we simply reset the device to its welcome state.

5.4 PATHFINDING

Pathfinding is the process of picking the path between our starting point and the point we want to get to. We want to minimize the length of this path as much as possible. To do this we calculate the shortest path between all possible points, and then when we need to know what path to take, we can look up which path is shortest from where they are stored on the device's SD card. We precompute these paths on a seperate more powerful computer, as this process is computationally expensive. The algorithm we use to do this is the *Floyd-Warshall algorithm*. The idea behind the algorithm is this:

- Label the possible points 1 through to n.
- The shortest path between two points that does not go through any other points, is the corridor directly connecting them, if one exists.
- The shortest path between two points that may pass through the first $k+1$ points is either the shortest path that may pass through the first k points, or it is a path the connects the start point and the $k+1$th point followed by the path that connects the $k+1$th point to the end point.

In order to find the best path between each pair of points using these facts, the following method is used:

- The initial value for the current best path between each pair of points is the corridor directly connecting them, if one exists, or else it is left undefined.
- For each start point i, end point j and path length k.
- If the length of the current best path between i and kth point plus the length of the current best path between the kth point and j is less than the length of the current best path between i and j (or the current best path between i and j is undefined), then the new best path between i and j is the current best path between i and the kth point followed by the current best path between the kth and j.

By going through every value of k, we ensure that we find all the paths using any possible intermediate nodes, thus enumerating all possible paths.

5.4 BAROMETER POSITIONING

While Accelerometer-Magnetometer and Wi-Fi based positioning offers strictly two dimensional north-east positioning, we use a barometer to extend this to three dimensional, allowing us to detect which floor we are on. The barometer we have used has an accuracy of 0.3mbar, which is not enough to enable accurate absolute altitude positioning due to the high variability in indoor pressure by temperature, location and weather. Instead of this, we instead to measure when the user changes floors, either up or down. This is based on the observation that pressure changes only occur when changing floors. While we do need to know the direction of the pressure change, we do not attempt to predict actual altitude change from the change in pressure. To achieve this we pass the barometer data through two filters - a high band-pass filter and a low-pass filter. The first filter is used to detect when the pressure is changing, and the second is used to see whether it is changing upwards or downwards. The first filter has a high-pass component so that only the changes are visible, and also a low-pass

component, albeit with a very high frequency cutoff, to remove noise that would cause incorrect readings. The second filter will allow us to see the direction of the pressure change as it should produce average pressure values which we can compare against the previous pressure to see whether we have gone up or down a floor.

5.4 ACCELEROMETER-MAGNETOMETER POSITIONING

In order to measure our position using a accelerometer and magnetometer, we need to measure both the speed and our direction at any given moment. We can use these to give us a velocity and then add these velocities together over time to approximate the integral

$$\mathbf{p} = \int \mathbf{v} \, \mathrm{d}\, t$$

by Riemann integration:

$$\mathbf{p} \approx \sum_{t} \Delta t \cdot \mathbf{v}(t)$$

Where $\mathbf{p}$ is position, and Δt is the time between successive observations.

MEASURING DIRECTION

Measuring direction from magnetometer readings is easily accomplished in certain circumstances: by assuming that the device is held perfectly horizontal at all times, calculations are greatly simplified. However, in reality, this is not the case, and we need to correct the compass for any tilt in the device.

In order to correct for tilt, we must first measure it. This is done by measuring the accelerometer values when the device is stationary. While we cannot always assume that this is true it is easy to simulate by applying a low-pass filter to the data to smooth out small changes to the acceleration, i.e changes in velocity, leaving only the underlying gravitational acceleration. This smoothing can be achieved easily using an exponential filter, the same kind as used in the touch screen smoothing.

When these stationary value have been measured, the tilt can be calculated. This is computed as follows: We know that our measured accelerometer values are a rotation of the downward pointing gravitational acceleration vector.

$$\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi) \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

where $\mathbf{R}_a(\theta)$ is rotation matrix along the a-axis of an angle θ, ϕ is roll, θ is pitch and ψ is yaw, which does not affect the calculation of the tilt, since

$$\mathbf{R}_z(\theta) \begin{bmatrix} 0 \\ 0 \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ x \end{bmatrix}$$

Inverting this, we can obtain our roll and pitch angles:

$$\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = \mathbf{R}_y(-\theta)\mathbf{R}_x(-\phi) \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix}$$

Substituting in the formulas for the rotation matrices, we have

$$\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = \begin{bmatrix} \cos\theta A_x + \sin\theta\sin\phi A_y + \sin\theta\cos\phi A_z \\ \cos\phi A_y - \sin\phi A_z \\ \cos\theta\cos\phi A_z + \cos\theta\sin\phi A_y - \sin\theta A_x \end{bmatrix}$$

Thus we can see that

$$\phi = \tan^{-1}\frac{A_y}{A_z}$$

$$\theta = \tan^{-1}\frac{-A_x}{\cos\phi A_y + \sin\phi A_z}$$

We also know that the magnetometer readings should be some rotation of a vector pointing to the center of the earth - that is in some unknown direction ω. From this we can write:

$$\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi) \begin{bmatrix} M\cos\omega \\ 0 \\ M\sin\omega \end{bmatrix}$$

where M is the Earth's magnetic field strength. Again inverting the first two rotations, we obtain

$$\mathbf{R}_y(-\theta)\mathbf{R}_x(-\phi) \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \mathbf{R}_z(\psi) \begin{bmatrix} M\cos\omega \\ 0 \\ M\sin\omega \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta M_x + \sin\theta\sin\phi M_y + \sin\theta\cos\phi M_z \\ \cos\phi M_y - \sin\phi M_z \\ \cos\theta\cos\phi M_z + \cos\theta\sin\phi M_y - sin\theta M_x \end{bmatrix} = \begin{bmatrix} M\cos\psi\cos\omega \\ -M\sin\psi\cos\omega \\ M\sin\omega \end{bmatrix}$$

Thus we can compute the final yaw angle, that is the heading relative to north:

$$\psi = \tan^{-1}\frac{\sin\phi M_z - \cos\phi M_y}{\cos\theta M_x + \sin\theta\sin\phi M_y + \sin\theta\cos\phi M_z}$$

MEASURING SPEED

The noise in the accelerometer data is significant enough that the acceleration associated with small changes in velocity, such as those that occur while walking, are not easily distinguishable from background noise. As such, instead of calculating speed by directly integrating the acceleration values, we instead recognise when a single steps occur, and then calculate the speed by the product of step frequency and step length. The step lengths used are the average step length for adults - approximately 74cm.

The accelerometer used internally by the Genuino 101's Intel Curie module is a Bosch BMI160. This has the advantage of having a built in, hardware based step detector. This is more accurate and much faster than a software based solution, so we will be using this. The method used by the Curie module to find when the user steps works as follows: to see when the user steps, we must look for changes in acceleration in the z-axis. First we must correct for tilt in the same way as described above, calculating the true z-axis acceleration value as

$$a_z = \cos\theta\cos\phi A_z + \cos\theta\sin\phi A_y - \sin\theta A_x$$

Using a band-pass (i.e both a low-pass and high-pass component) exponential filter tuned to cut out both noise and the underlying gravitational acceleration, we can clean up the signal to only the changes due to walking. Then we simply look for when this value passes above a certain threshold. To eliminate further sources of error, such when moving the device around in movement that is not walking, we only look for steps in a specific range: studies have shown that step frequency varies by about 15%, so we need only look for steps that occur 15% either side of where we predict the next step should be, based on the current average step frequency. If we detect a step before this range, it is most likely not a step, and if we do not detect a step at all, then we have most likely stopped. Finally, we check if the current speed is close to the previous speed. If so, we filter it through a low-pass filter to remove noise. If it is not close to the previous speed, we leave it as it is. This is because sudden large changes in speed do occur frequently, while small changes are more likely to be noise than actual changes.

To combine the speed and direction data, we calculate a velocity vector as follows:

$$\mathbf{v} = \begin{bmatrix} c \sin \psi \\ \\ c \cos \psi \end{bmatrix}$$

5.4 WI-FI POSITIONING

To transform the mathematical model into a system that can be implemented in software, we first need to express all the power loss equations in decibel form to allow easy practical calculations, as all power values available from the Wi-Fi module are in dBm (decibel-milliwatts).

FREE SPACE PATH LOSS

$$L_{FSP} = 20 \log \frac{c}{4\pi f} - 20 \log d$$

DIELECTRIC LOSS

$$L_D = -10k \log e \sum_{i}^{n} \delta_i x_i$$

Where the sum is over n different materials that are travelled through by each wave - with δ_i representing the dielectric loss tangent for the ith material and x_i the distance travelled through that material.

FRESNEL LOSS

$$L_F = 20 \sum_{i=2}^{n} \log\left(n_{i-1}\cos\theta_i - \sqrt{n_i^2 - n_{i-1}^2 \sin^2\theta_i}\right) - \log\left(n_{i-1}\cos\theta_i + \sqrt{n_i^2 - n_{i-1}^2 \sin^2\theta_i}\right)$$

Here, the sum again represents the sum over the materials travelled through by the wave, but in this case we start the sum at $i = 2$ and not $i = 1$ as we only wish to add a loss at each material boundary. n_i represents the refractive index in the nth material.

MULTIPATH PROPAGATION

When a wave encounters the boundary of two materials, the wave is partially reflected and partially refracted. The loss of energy due to reflection is captured in the Fresnel loss equation, however, this reflected wave can go on to reach some other point in space, thus increasing the total power incident at that point and thus decreasing the effective path loss at that point.

Effectively simulating multipath propagation is a computationally difficult problem. Luckily, lots of work into this has been done in the field of computer graphics. In computer graphics this is know as the global illumination problem. Formalising this we arrive at an equation known as the *rendering equation* (Kajiya 1986):

$$L_o(\mathbf{x}, \omega_o, t) = \int_\Omega f_r(\mathbf{x}, \omega_o, \omega_i, t) L_i(\mathbf{x}, \omega_i, t)\,\mathrm{d}\,\omega_i$$

Where L_o represents the output radiance of any given position $\mathbf{x}$ towards some angle ω_o at time t, L_i represents the incoming light to $\mathbf{x}$ at some angle ω_i, and f_r is a function describing how much of the incoming radiation at ω_i is transmitted at the angle ω_o, it is known as the *bidirectional reflectance distribution function*. The integral is taken over the unit sphere, Ω, to account for the incoming light in all directions. The equation is modified for our usage by integrating over a whole sphere instead of a hemisphere, as we are not interested at the incident radiance on a surface, but rather at any given point. Similarly, we ignore a term accounting for emitted radiance in the original equation, as the only source of EM waves in our system is the Wi-Fi router we are simulating.

What we want to compute is the value for our position, $\mathbf{x}$, given the value of L_i, which we can work out from the path loss that we measure. However, existing algorithms only exist for calculating L_i given $\mathbf{x}$, so perform the inverse, we calculate L_i for every point in space and then see which point corresponds to our measured L_i

value. This is a very computationally expensive process. Luckily however, this map of L_i at every position is invariant for each building, so we can calculate it once on a powerful desktop computer and then keep it on the SD card on the device, ready to be used whenever necessary.

In order to compute this map we will be using a simplification of an algorithm known as photon mapping (Jenson 2000). First we quantise the space which we wish to simulate into blocks of a size comparable to the volume of the recieving device. Then we execute the simplified algorithm as follows:

- A large number of simulated photons are sent out in random directions.
- At each timestep, the position of each photon is advanced according to its velocity (calculated from the refractive indices of the material it is travelling through).
- At each timestep, each photon has the chance to be absorbed by the medium through which it is travelling or to be unchanged. These absorptions happen with a probability of
$$1 - e^{-\delta k \Delta x}$$
where Δx is the distance travelled by the photon during the last timestep (this equation is derived from the dielectric loss formula).
- At each timestep each photon is checked to see if it has changed mediums. If it does it is either reflected with a probability of
$$\left| \frac{n_1 \cos\theta_i - \sqrt{n_2^2 - n_1^2 \sin^2\theta_i}}{n_1 \cos\theta_i + \sqrt{n_2^2 - n_1^2 \sin^2\theta_i}} \right|$$
which corresponds to the reflected power, as predicted by the Fresnel equations. If it is not reflected, it is refracted according to Snell's law, and its velocity is changed according to its new medium.
- Finally, the number of photons in each block in our simulation is counted. This, combined with the transmission power and the number of photons simulated, allows the calculation of L_i values at each point.
- The simulation is run until there are no photons left in the simulation space.

In this algorithm, we do not account for the free space path loss as this is automatically simulated by the spreading out of the photons as the simulation is run. This algorithm is part of a family of algorithms known as the Monte Carlo algorithms, which solve integral equations by summing many random samples. The accuracy of these algorithms can be improved by using more samples, which in this case corresponds to simulating more photons.

TRIANGULATION

Given the path loss from one Wi-Fi transmitter, we may not be able to calculate a unique, sensible position. This is because the measured path loss is very inaccurate, so a path loss that *exactly* matches a certain position may be completely wrong, while one that is quite close to the measured path loss, but not exact, may be correct. To make our system more accurate, we can see how credible each proposed position is by evaluating it against our predicted location from the accelerometer data.

To do this, we first assume that the position as predicted by our path loss is normally distributed about some true position, as is the predicted accelerometer position. Then we use Bayes' theorem to find the probability that any given position is the correct one:

$$\mathrm{P}(\mu = a | x = b) = \frac{\mathrm{P}(x = b | \mu = a)\,\mathrm{P}(\mu = a)}{\mathrm{P}(x = b)}$$

Where a is the potential 'correct' path loss and b is the observed path loss. We know that $\mathrm{P}(x = b|\mu = a)$ is the probability that we observe b given that the actual path loss is a. We know then that $\mathrm{P}(x = b|\mu = a)$ is normally distributed about a with some standard deviation σ_1 that is the error in the Wi-Fi position. $\mathrm{P}(\mu = a)$ is the probability that we are at the position with path loss a, given our accelerometer position. Using $\mathbf{p}_x$ to indicate the position with path loss x, we can say that

$$\mathrm{P}(\mu = a) = \mathrm{P}(\mathbf{p}_p = \mathbf{p}_a)$$

where $\mathbf{p}_p$ is the position predicted by the accelerometer. This is normally distributed with some standard deviation σ_2 which is the error in the accelerometer position. Finally, $\mathrm{P}(x = b)$ is the probability that we observe b given that we are in the position predicted by the accelerometer, $\mathbf{p}_p$. We can then say that

$$\mathrm{P}(x = b) = \mathrm{P}(p = b)$$

where p is the path loss at the accelerometer location. We know that this must be normally distributed with a standard deviation $\sigma_3 = \sigma_1 + \sigma_2$ as the error in the accelerometer and the error in Wi-Fi data are independent.

Substituting the value of the probability density function for the normal distribution and simplifying, we get:

$$\begin{aligned}\mathrm{P}(\mu = a | x = b) &= \frac{\mathrm{P}(x = b | \mu = a)\,\mathrm{P}(\mathbf{p}_p = \mathbf{p}_a)}{\mathrm{P}(p = b)} \\ &= \frac{\sigma_1 + \sigma_2}{\sqrt{2\pi}\sigma_1\sigma_2} e^{-\frac{1}{2}\left(\frac{(b-a)^2}{\sigma_1^2} + \frac{|\mathbf{p}_a - \mathbf{p}_p|^2}{\sigma_2^2} + \frac{(b-p)^2}{(\sigma_1+\sigma_2)^2}\right)}\end{aligned}$$

In order to find our position, we then, theoretically, compute this value for every position in the building and select the most probable. However, in practise it suffices to compute this in all the positions two standard deviations or less away from the predicted accelerometer position, drastically increasing performance. This system may still not be very accurate, so we can also take into account multiple Wi-Fi transmitters, and compute the probabilities for each transmitter and then simply multiply the corresponding values (and optionally square root - this helps avoid rounding errors introduced by using very small numbers) and select the most probable position.

6

TESTING AND MANUFACTURE

The first tests we conducted were for the components which we could use without a PCB or a finished product. The acceleromter was tested first followed by the barometer and then the magnetometer.

The accelerometer was hooked up to a computer and the output was measured. Then it was calibrated and tests were ran to try and determine step count and stride length for individual people. These tests proved inconclusive as manual step detection (not using the built in step counting function) was unreliable. As such we used average step sizes for adult humans, obtained from several studies.

The barometer tests consisted of hooking up the barometer to a computer and carefully calculating the small or large changes in pressure that occured when it was brought up stairs, moved up and down while stationary and finally with windows or doors open so the NaviGrip can account for each variable and ensure that it knows when it has actually changed floors or if someone in the hallway has just opened a door.

The magnetometer tests were conducted by calculating the heading of the device and comparing it to the heading obtained from a compass and we had no problems, it worked perfectly, and was accurate to about 5 degrees, which was expected.

The Wi-Fi transceiver's testing required that we had three working routers. We set them up in different classrooms and other locations in and around a hallway and used them to accurately triangulate a poisition to around two metres of accuracy which is close enough to be used at Intel HQ.

6.1 PROBLEMS ENCOUNTERED

The Wi-Fi testing proved more difficult than initially expected and although there were problems getting permissions to access the school's Wi-Fi service without restriction, it was accomplished on time.

Whilst all of these could be tested individually, the readings they gave would mean nothing without the PCB to connect them to. This was difficult due to the fact that the PCB took longer than expected because of other complications.

6.2 SPECIALISED TESTING

6.2 PCB

The PCB is the central hub of the project and without out it the final product could not work. We had to use two different methods to make it and then solder all of our components to it and ensure one-hundred percent functionality, as discussed in the next section.

6.2 MATHEMATICAL MODELLING

The mathematical models were used primarily to determine the loss of radio signal strength due to walls and other obstructions. The signal strength change used to determine the exact location was found by calculating dielectric loss, free space path loss and variable attenuation in irregular materials giving us an accurate picture in the horizontal plane with which to pinpoint the NaviGrip. The dielectric loss coefficients and refractive indexes were measured using a laptop equipped with Wi-Fi, and not the device itself. This was easier to use and develop code for, and the measured values would be the same as if they had been measured on the device. As direct measurements of the electric and magnetic permittivities are not possible, we instead measured the refractive indexes by measuring the spread of Wi-Fi signals as they passed through the object we wanted to measure - the object would act like a dispersive lens (if it had a refractive index more than that of air) or a focussing lens (if its refractive index was less than that of air), and by measuring the signal strength at an angle to the material, we could see how spread out the signal was, and thus estimate the refractive index. In order to measure the dielectric loss tangent, we simply measured loss in signal strength through a wall of known width. To minimize other losses, we ensured that the transmitter and reciever were both as close to the wall as possible and that both devices were places normal to the wall to prevent refraction.

6.3 MANUFACTURE

The parts were printed, as per the Mark 4 design, in PLA at the Bath University residential (as discussed below). The device was then assembled:

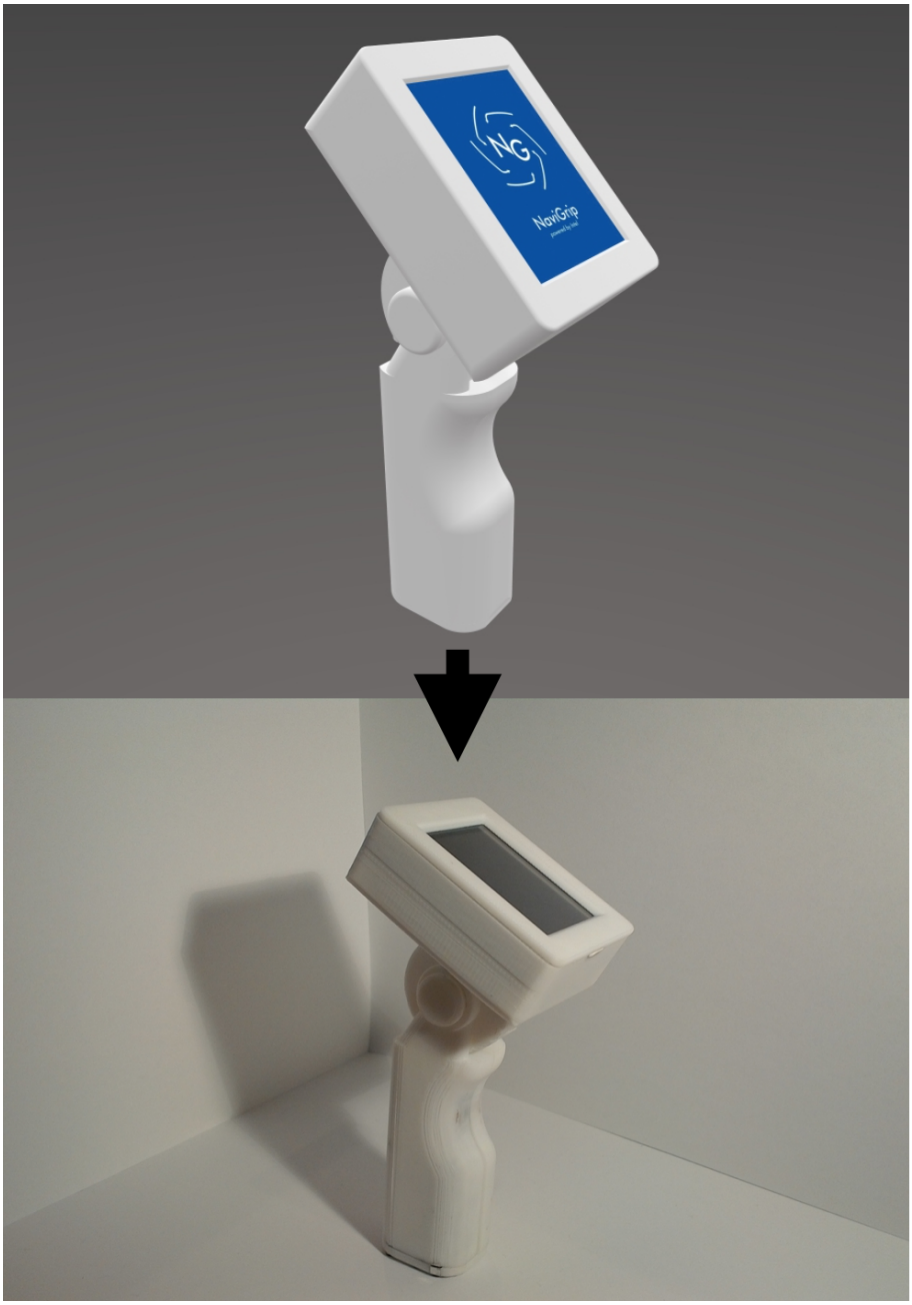


Figure 6.1: The full device, as designed and as printed.

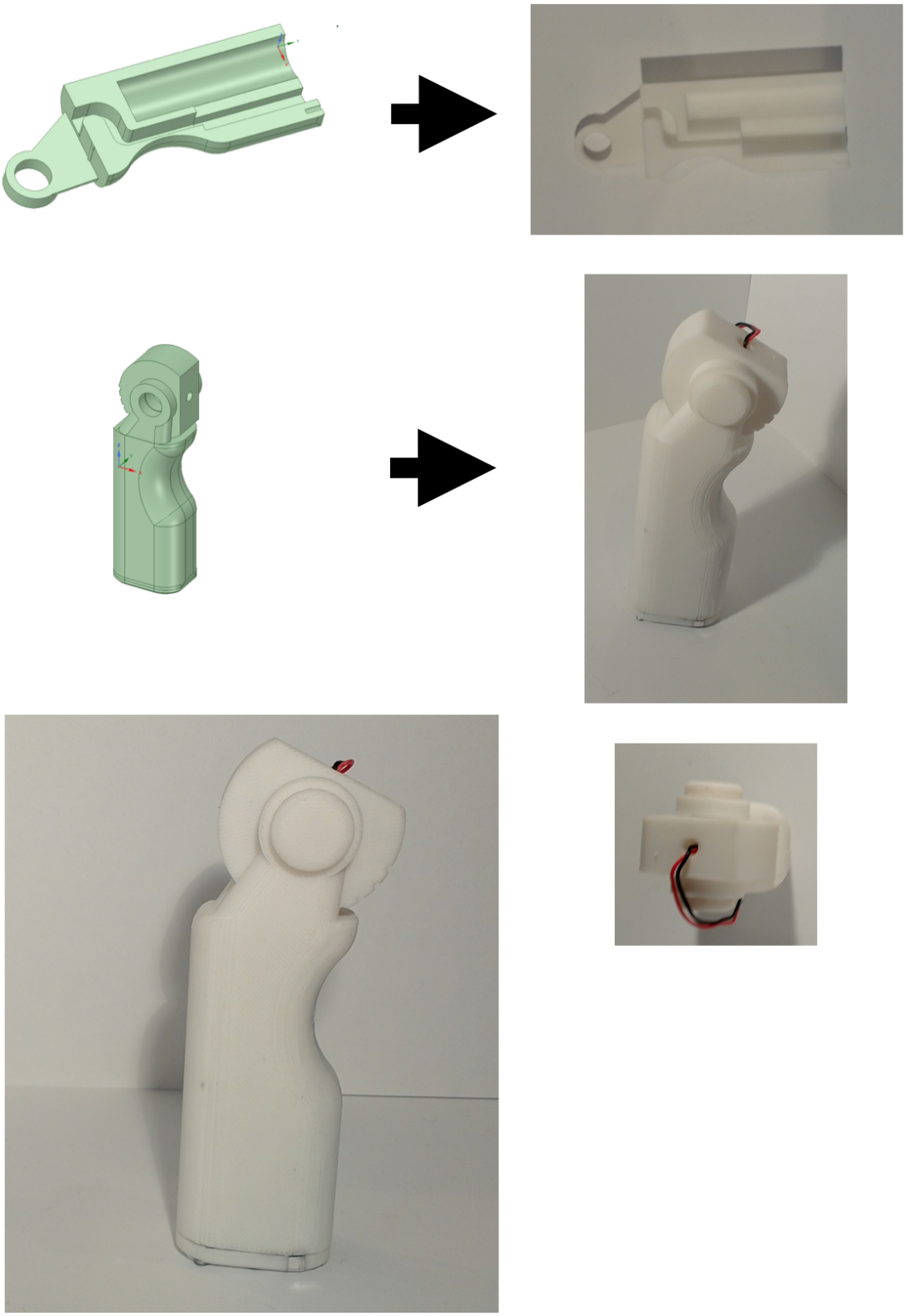


Figure 6.2: Each individual part as printed.

6.4 RESIDENTIAL

From January 31st until February 2nd, Bath University hosted many EES groups to allow them to either make or complete their projects.

This is a timetable for the three days at Bath which the team roughly adhered to. It shows the various requirements that had to be met every day and the processes needed to finsih a project like this.

Day 1

11:00	Load printing program onto 3D printer for printing ABS
	Finish programming
11:30	Start the ABS module printing
11:30	Prepare the PCB and begin the etching process
12:30	**LUNCH**
13:30	Start populating the PCB soldering on the components
15:00	Briefing Session 4E 3.38

Day 2

9:00	Finish any uncompleted work from day 1
9:30	Student briefing for one person
10:00	Printing Ninja Flex
10:30	Assess whether the ABS module has printed and check if the PCB works and correct mistakes
12:30	**LUNCH**
13:30	Finish off perfecting PCB
14:30	**BREAK**
15:00	Testing and start creating presentation/report
17:30	Supper

Day 3

9:00	Finishing off undone stuff
9:30	Student briefing for one person
10:00	Testing and completing presentation/report
12:30	**LUNCH**
13:30	Reprogramming any inaccuracies
14:30	Presentation Practise

Figure 6.3: The team schedule while attending the Bath residential.

The time spent at Bath University was immensly helpful, not only for completing the major elements of the project but also for learning new methods and processes and perfecting our ideas, really giving the team free reign with both imagination and with feasibility.

6.4 PROBLEMS ENCOUNTERED

The 3D-Printer designs took about a month to fully complete for the residential. This was because the software was quite difficult to use and intially to understand. Eventually, once the designs were finished, the printers were the problem. 3D-Printers are notoriously difficult to maintain and are prone to motor failures, printing head failures and unforseeable printing errors due to the thinness of a wall or a misprint. While making the final handle at Bath University, one wall was made thinner to allow for more wires and a small hole was printed in both grip-halves. The problem was fixed by altering the wall thickness for future versions and also by melting some of the excess print raft into place and filling the holes.

The plan for the first day at the residential program was to etch the PCB using a PCB milling machine at the engineering lab. However, the machine had broken down and was not working properly, which was only discovered just before we made our first attempt. Once the machine was fixed, the double sided PCB was etched with copper connections and was taken to the electronics lab to solder. We then faced problems with soldering the PCB as many of the connections did not work during the first few attempts. Correcting this involved a lot of soldering and desoldering over the course of the last two days of the residential, which resulted in the erosion of the copper connections. The harm this caused to the functionality of the PCB could not be detected before the PCB was connected to a computer and had a programme uploaded, because when tested with a multimeter, the PCB appeared to be working, but did not fully calibrate to any computer that contained the code which would run it. Due to this, we had to reattempt at making a PCB from scratch in the Design and Technology lab in school.

As we were making the PCB from scratch this time, without a PCB milling machine, both sides of the PCB had to be aligned an etched separately, This was tricky as misalignment was highly probable. Moreover, one side could easily get overexposed while removing the covering off the other. We encountered both problems, as our PCB was misaligned twice; once with the incorrect orientation and the next with a centimetre shift to the left, and finally it was also overexposed once. To prevent misalignment, we used tape to attach the tracing paper to the second side of the PCB, taking precise measurements. Overexposure was prevented by working quickly and preparing as much of it as we could in advance. The PCB was tested with a multimetre and appeared to be working, and we even got the resistive touch screen to work with the PCB.

When testing the PCB with the Arduino, a design flaw was noticed. The serial bus was being used for both the wifi module and the connection with the computer, which didn't fit in and work with the Arduino components. The board was then modified by cutting the traces between the wifi and the serial bus on the Arduino. Unfortunately, the Arduino broke. In order to use the PCB with a new Arduino, header stacks were needed, but the ones available in England were too short. So, instead, the board was put on a breadboard for testing, until the new Arduino was modified to accomodate the header stacks.

The code is difficult to comprehend at first as we were writing from scratch and creating a program that has never been used or realeased before. This meant long hours and time consuming back checking to ensure all of it worked before our field testing. The most difficult part of the code was accounting for the response time of the touch screen and needed quite a lot of work before it worked properly.

7 EVALUATION

7.1 OUTCOME

Overall this project has been a success. We have met the initial requirements set by the Intel Engineers of producing a navigational device which is:

- Easy to use - the device includes a touch screen, the interface of which will be familiar to smartphone and tablet users.
- Low cost - while all the components were not bulk ordered, and they were purchased in their much more expensive breakout board form, the device was still relatively low cost.
- Compact - the device fits easily in the users hand, and is not very heavy so can be carried for long periods of time without tiring.
- Robust - by printing in solid PLA, the device is sturdy. This combined with the rechargeable battery means the device can be used many times.

Our device will have a range of 1.5m which is more than accurate to be used in navigating round an office building, where there are often long stretches of corridors. Also the use of Wi-Fi in the positioning of the device allows it to be transferable. This is a key point which we feel was perhaps not covered in the specification but was useful to include when building the product.

7.2 TEAMWORK THROUGHOUT THE PROJECT

We believe our unique range of skills have allowed us to work efficiently and effectively of the team. Not only do we have strength in coding and mathematics but it was also our ability of foresight and constant time management that allowed this project to flourish. A major benefit of ours lies in our instant task allocation:

Tuomas Researching coding methods possibly applicable

Simran Research mathematical modelling techniques

Isabella Begin creating time plans, logistical plans and costing sheets

Jack Ensure communication is effective and efficient and keep detailed notes of our progress

Alexander Begin creating concept designs and identify their strengths and weaknesses

Freddie Take concept designs and begin modelling for the prototype

This clear distinction of our roles allowed us to dive straight into the project and saved us time. This was crucial as the deadline for materials needed at Bath University was our first major deadline we had to meet and we did so successfully.

A weakness of the team in general was our busy life schedules, meaning sometimes we got diverted away from the project and our own set deadlines got put on hold as we dealt with other things. However, we were generally good at forewarning the rest of the team if it didn't look like we would meet a deadline and we ensured to meet it as soon as possible.

Most of our communication thorough out this project has been conducted at weekly meetings with the Intel Engineers and over Email and online social networking. This has allowed us to have quick and instant responses from all team members and assisted us in staying on track.

We never had any conflict between team members and this is due to our tolerant attitudes, which allowed us to understand each other's views and thoughts.

7.3 TIME MANAGEMENT AND PLANNING

Overall, as a team we were quite efficient at our time management and often managed to stay on schedule. Generally, the only times we ran over was when we had problems outside of our control, such as the school's 3D printer breaking, meaning creating the prototype was delayed. This was an unforeseeable problem that couldn't be accounted for. However, we have been good at estimating the specific time needed for completing individual parts of the project and scheduling the appropriate amount of time.

Our cost planning was also an important factor of our project, as we didn't have a specified budget but we understood that if the device were to be replicated in a business it would have to be as low cost as possible. Therefore, we created a costing sheet to keep the project's spending on track. Unfortunately, as we were making only a single device this didn't allow for bulk purchase discounts. This meant that our final cost of design was higher than it would be had we been making 10 or 100.

At each weekly meeting we ensured that we measured our progress against the Gantt charts we created. This made sure that we knew exactly where we were supposed to be each week and stopped us from falling too far behind schedule. Often we found that we were slightly ahead of schedule. Where that was the case we would take the time to write up what we had done so that the latter stages of report writing would be made easier.

Overall, we believe that this project has been a success. We have met our deadlines; worked well as a team and met the set criteria in an innovative way. The NaviGrip is an autonomous self-guided system that we believe fits the brief that Intel set us.

8 BIBLIOGRAPHY

- Ozyagcilar, Talat. "Implementing A Tilt-Compensated Ecompass Using Accelerometer And Magnetometer Sensors". N.p., 2015. Web. 1 June 2017.
- Lindhardt. "Chapter 11: Dielectric Properties Of Materials". N.p., 2017. Web. 20 Jan. 2017.
- "Illustrated Glossary Of Organic Chemistry - Beer's Law (Beer-Lambert Law)". Web.chem.ucla.edu. N.p., 2017. Web. 2 June 2017.
- Darabi, Houshang, and Pat Banerji. "Survey Of Wireless Indoor Positioning Techniques And Systems - IEEE Xplore Document". Ieeexplore.ieee.org. N.p., 2016. Web. 26 Nov. 2016.
- Gaudlitz, Eva. "Techniques For Client-Based Indoor Positioning – GPS, Wi-Fi, Blue". Infsoft.com. N.p., 2016. Web. 27 Nov. 2016.
- "A Practical Guide To Global Illumination Using Photon Maps". graphics.stanford.edu. N.p., 2000. Web. 15 Dec. 2016.
- T. Kajiya, James. "The Rendering Equation". http://www.cse.chalmers.se/. N.p., 2017. Web. 16 Dec. 2016.

9

APPENDIX A - DERIVATION OF FRESNEL'S EQUATIONS

Fresnel's equation works on the principle that light is an electromagnetic wave, and it is therefore subject to experiencing the dieletric properties of various materials. It can be derived using a combination of Maxwell's Equations and Snell's Law.
Electric Component:

$$\mathbf{E}_i + \mathbf{E}_r = \mathbf{E}_t$$

Magnetic Component:

$$\mathbf{B}_i \cos\theta_i - \mathbf{B}_r \cos\theta_r = \mathbf{B}_t \cos\theta_t$$

where:

- θ_i , θ_r and θ_t are the angles of incidence, reflection and transmission
- $\mathbf{E}$ is the component of the electric field
- $\mathbf{B}$ is the component of the magnetic field

$$\frac{\mathbf{E}}{\mathbf{B}} = \frac{\omega}{k}$$

therefore,

$$\mathbf{B} = \frac{k}{\omega}\mathbf{E}$$

as ω is the same for both media by the law of conservation of energy,

$$k_1(\mathbf{E}_i - \mathbf{E}_r)\cos\theta_i = k_2\mathbf{E}_t \cos\theta_t$$

Using the laws of refraction,

$$n = \frac{c}{v} = \frac{k}{\omega}c$$

where n is the refractive index.
This relation is derived from the assumption that the wave vector and the angular frequency are not independent, and that the above dispersion relation must be followed to come up with a vaild solution to the equation
Therefore, replacing ω and c,

$$n_1(\mathbf{E}_i - \mathbf{E}_r)\cos\theta_i = n_2\mathbf{E}_t\cos\theta_t$$

Substituting $\mathbf{B}_t$,

$$\mathbf{E}_i(n_1\cos\theta_i - n_2\cos\theta_t) = \mathbf{E}_r(n_1\cos\theta_i + n_2\cos\theta_t)$$

rearranging the equation:

$$\frac{\mathbf{E}_i}{\mathbf{E}_r} = \frac{(n_1\cos\theta_i - n_2\cos\theta_t)}{(n_1\cos\theta_i + n_2\cos\theta_t)}$$

therefore for intensity; using the inverse square law:

$$\frac{I_r}{I_i} = \left[\frac{(n_1\cos\theta_i - n_2\cos\theta_t)}{(n_1\cos\theta_i + n_2\cos\theta_t)}\right]^2$$

Let

$$n_1 = \sqrt{\epsilon_2\mu_2}$$

and

$$n_2 = \sqrt{\epsilon_1\mu_1}$$

Which are derived from Maxwell's equations. Substituting these yields

$$\frac{I_r}{I_i} = R = \left[\frac{\left(\sqrt{\epsilon_2\mu_2}\cos\theta_i - \sqrt{\epsilon_1\mu_1}\cos\theta_t\right)}{\left(\sqrt{\epsilon_2\mu_2}\cos\theta_i + \sqrt{\epsilon_1\mu_1}\cos\theta_t\right)}\right]^2$$

where μ is Magnetic Permittivity and ϵ is Electrical Permittivity

10

APPENDIX B - DEVICE CODE

EES.INO

```
1
2 #include "GUI.h"
3 #include "Scheduler.h"
4
5 Compass compass = Compass();
6 Barometer barometer = Barometer();
7 Wifi wifi = Wifi();
8 GUI gui = GUI();
9 Position position = Position(&wifi, &compass, &barometer);
10
11
12 void setup() {
13   begin_misc();
14   compass.begin();
15   barometer.begin();
16   gui.begin();
17   wifi.begin();
18   position.begin();
19   Serial.begin(9600);
20
21   //use the scheduler. this allows multiple update loops to be run
        effectively simultaneously
22   //we put yields()s in throughout the code to give the scheduler an
```

```
    opportunity to run a
23  //different task while we are waiting for things to complete, such
    as IO reads or long drawImage calls.
24  //this keeps the device operating even while long draw calls or sd
    card reads are occuring.
25  Scheduler.startLoop([]() { wifi.update(); });
26  Scheduler.startLoop([]() { barometer.update(); });
27  Scheduler.startLoop([]() { compass.update(); });
28  Scheduler.startLoop([]() { position.update(); });
29  Scheduler.startLoop([]() { gui.loop(); });
30 }
31
32 void loop() {
33  yield();
34 }
```

BAROMETER.H

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>

#define SEALEVELPRESSURE 1017
#define BAROMETER_ALPHA_H 0.1
#define BAROMETER_ALPHA_L 0.2
#define BAROMETER_THRESHOLD 13

class Barometer {
  //the interface to the pressure sensor
  Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
  sensors_event_t event;

  float last_band_pass = 0;
  float last_low_pass = SEALEVELPRESSURE;

  int dir = 0;
  bool reset = true;

  public:

  void begin();
  void update();
  int delta();
};
```

BAROMETER.CPP

```
#include "Barometer.h"

void Barometer::begin() {
  this->bmp.begin();
}

void Barometer::update() {
  this->bmp.getEvent(&this->event);

  //pass the pressure through a low-pass filter to remove noise
  float low_pass = this->last_low_pass + BAROMETER_ALPHA_L *
      (this->event.pressure - this->last_low_pass);

  //then through a high-pass filter to detect changes
  float band_pass = BAROMETER_ALPHA_H * (this->last_band_pass +
      low_pass - this->last_low_pass);

  this->last_low_pass = low_pass;
  this->last_band_pass = band_pass;

  //we want the magnitude not the direction.
  float abs_band_pass = abs(band_pass);
  //if we are changing pressure sufficiently
  if(abs_band_pass > BAROMETER_THRESHOLD && this->reset) {
    //compute the sign of the change
    //this the delta value to add to the floor
    this->dir = band_pass/abs_band_pass;
    this->reset = false;
  } else {
    this->dir = 0;
    this->reset = true;
  }
}
int Barometer::delta() {
  int d = this->dir;
  this->dir = 0;
  return d;
}
```

COMPASS.H

```
#include "math.h"
#include "Wire.h"
#include "Adafruit_Sensor.h"
#include "Adafruit_LSM303_U.h"

#define COMPASS_ACC_ALPHA 0.1
#define COMPASS_MAG_ALPHA 0.2

class Compass {
    //interface to the magnetometer and the accelerometer.
    Adafruit_LSM303_Mag_Unified mag =
       Adafruit_LSM303_Mag_Unified(12345);
    Adafruit_LSM303_Accel_Unified accel =
       Adafruit_LSM303_Accel_Unified(54321);
    sensors_event_t event;

    //previous values
    float acc_x = 0;
    float acc_y = 0;
    float acc_z = 0;
    float mag_x = 0;
    float mag_y = 0;
    float mag_z = 0;

    public:

    void begin();
    void update();
    float heading();
};
```

COMPASS.CPP

```
1  #include "Compass.h"
2
3  void Compass::begin() {
4    this->mag.enableAutoRange(true);
5    this->mag.begin();
6    this->accel.begin();
7  }
8
9  void Compass::update() {
10   //run the latest acc/mag values through low-pass filters.
11
12   this->accel.getEvent(&this->event);
13
14   //use exponential smoothing filter
15   this->acc_x = this->acc_x + COMPASS_ACC_ALPHA *
         (this->event.acceleration.x - this->acc_x);
16   this->acc_y = this->acc_y + COMPASS_ACC_ALPHA *
         (this->event.acceleration.y - this->acc_y);
17   this->acc_z = this->acc_z + COMPASS_ACC_ALPHA *
         (this->event.acceleration.z - this->acc_z);
18
19   this->mag.getEvent(&this->event);
20
21   this->mag_x = this->mag_x + COMPASS_MAG_ALPHA *
         (this->event.magnetic.x - this->mag_x);
22   this->mag_y = this->mag_y + COMPASS_MAG_ALPHA *
         (this->event.magnetic.y - this->mag_y);
23   this->mag_z = this->mag_z + COMPASS_MAG_ALPHA *
         (this->event.magnetic.z - this->mag_z);
24 }
25
26 float Compass::heading() {
27   //we know that if the device is held upright, our acceleration
       vector should be downwards.
28   //we use this to work out roll and pitch.
29   float roll = atan(this->acc_y/this->acc_z);
30   float pitch = atan(-this->acc_x/
31                      (cos(roll)*this->acc_y + sin(roll)*this->acc_z));
```

```
32
33   //we then derotate the magnetometer readings using our pitch and
        roll angles, and work out yaw.
34   float numer = sin(roll)*this->mag_z -
35                 cos(roll)*this->mag_y;
36   float denom = cos(pitch)*this->mag_x +
37                 sin(roll)*sin(pitch)*this->mag_y +
38                 sin(pitch)*cos(roll)*this->mag_z;
39   return atan2(numer, denom);
40 }
```

WIFI.H

```
#include "string.h"
#include "String.h"
#include "SoftwareSerial.h"
#include "Arduino.h"

#define WIFI_TX 5
#define WIFI_RX 6
#define WIFI_MAX_AP 5

class Wifi {
  SoftwareSerial serial = SoftwareSerial(WIFI_TX, WIFI_RX);

  char ap_macs[WIFI_MAX_AP][18];
  int ap_rssis[WIFI_MAX_AP];

  public:

  void begin();
  void update();
  int get_rssi(char* mac);
};
```

WIFI.CPP

```
#include "Wifi.h"

void Wifi::begin() {
  this->serial.begin(9600);
}

void Wifi::update() {
  //AT+CWLAP is the command to list access points. It returns a list of values in this format:
  //+CWLAP:(index,"SSID",rssi,"MAC",channel)
  //we want the mac and the rssi values
  this->serial.print("AT+CWLAP");
  String response = "";
  char c;
  //give the esp8266 2000ms to give all of its values
  long int endtime = millis() + 2000;
  while (endtime > millis()) {
    yield();
    //read everything that comes through.
    while (this->serial.available()) {
      c = this->serial.read();
      response += c;
    }
  }
  //we can expect that each line returned is <= 64 characters.
  int i = 0;
  char response_lines[WIFI_MAX_AP][65];
  memset(response_lines, 0, WIFI_MAX_AP * 65);
  //these are initially zero. We need to memset them because we don't know
  //how they'll be initialized and we're using the zero for detection later on.
  //strtok allows us to split the string into tokens, we're using it split into lines
  char* tptr = strtok((char*)response.c_str(), "\n");
  while (tptr != NULL && i < WIFI_MAX_AP) {
    strcpy(response_lines[i], tptr);
    i++;
```

```
35      //get the next token.
36      tptr = strtok(NULL, "\n");
37    }
38    int j;
39    //for each response line
40    for(i = 0; i < WIFI_MAX_AP; i++) {
41      if (response_lines[i][0] != 0) {
42        j = 0;
43        //we want to split it by the ","s so we use strtok again
44        tptr = strtok(response_lines[i], ",");
45        //we only want to find up to j == 3 so quit after the first 3
              tokens.
46        while(tptr != NULL && j < 4) {
47          if (j == 2) {
48            //rssi is in the second column
49            this->ap_rssis[i] = atoi(tptr);
50          } else if (j == 3) {
51            //and mac is in the third.
52            strcpy(this->ap_macs[i], tptr);
53          }
54          j++;
55          //get the next token
56          tptr = strtok(NULL, ",");
57        }
58      } else {
59        break;
60      }
61    }
62  }
63
64  int Wifi::get_rssi(char* mac) {
65    //search for the hotspot with this mac.
66    //we could use a hash table but its overkill for so few hotspots
67    for (int i = 0; i < WIFI_MAX_AP; i++) {
68      if (strcmp(this->ap_macs[i], mac) == 0) {
69        return this->ap_rssis[i];
70      }
71    }
72    return -1;
73  }
```

SCREEN.H

```
#include "SPI.h"
#include "SD.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

//touch screen pins
#define YP A0
#define YM A1
#define XP A2
#define XM A3

//tft control pins, hardware SPI is used.
#define TFT_DC 9
#define TFT_CS 10

//tft sd card parameters, for bitmaps
#define BUFFPIXEL 20

//parameters for touch screen smoothing
#define THRESHOLD 840
#define TFT_WIDTH 240.0
#define TFT_HEIGHT 320.0
#define C_X (TFT_WIDTH / (1024.0 - XST))
#define C_Y (TFT_HEIGHT / (1024.0 - YST))
#define TOUCH_ALPHA 0.22
#define XST 150
#define YST 150

//a recorded point on the touch screen.
class Point {
  public:
  float x;
  float y;
  bool t; //are we touching the screen

  Point(float x, float y, bool t) {
    this->x = x;
    this->y = y;
```

```
39      this->t = t;
40    }
41
42    Point() {
43      this->x = 0;
44      this->y = 0;
45      this->t = false;
46    }
47  };
48
49  //our touch screen. this class only shows display capabilities. a
        sepereate instance of TouchScreen
50  //is required to access the resistive touch sensor.
51  class Screen {
52    Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
53
54    File bmpFile;
55    int      bmpWidth, bmpHeight;        // W+H in pixels
56    uint8_t  bmpDepth;                   // Bit depth (currently must
          be 24)
57    uint32_t bmpImageOffset;             // Start of image data in file
58    uint32_t rowSize;                    // Not always = bmpWidth; may
          have padding
59    uint8_t  sdbuffer[3*BUFFPIXEL];      // pixel buffer (R+G+B per
          pixel)
60    uint8_t  buffidx = sizeof(sdbuffer); // Current position in
          sdbuffer
61    boolean  flip    = true;             // BMP is stored bottom-to-top
62    int      w, h, row, col;
63    uint8_t  r, g, b;
64    uint32_t pos;
65
66    uint16_t read16(); //read 16 bits from the bmp file
67    uint32_t read32(); //read 32 bits from the bmp file
68
69    public:
70
71    void begin();
72    void drawRect(uint16_t, int, int, int, int, bool);
73    void drawText(char*, uint16_t, int, int, int);
```

```
74   void drawImage(char*, uint8_t, uint8_t);
75 };
76
77 class TouchScreen {
78   float rawReadY();
79   float rawReadX();
80
81   Point last = Point(1023, 1023, false);
82
83   public:
84
85   void readTouch(Point* pos);
86   void waitTouch(Point* pos);
87 };
```

SCREEN.CPP

```
#include "Screen.h"

//read 16 and 32 bytes respectively from the SD card
//luckily both BMP and Arduino are little-endian.
uint16_t Screen::read16() {
  //each successive byte is shifted over by 8 bits
  uint16_t result = 0;
  result += this->bmpFile.read(); // LSB
  result += this->bmpFile.read() << 8; // MSB
  return result;
}

uint32_t Screen::read32() {
  uint32_t result = 0;
  result += this->bmpFile.read(); // LSB
  result += this->bmpFile.read() << 8;
  result += this->bmpFile.read() << 16;
  result += this->bmpFile.read() << 24; // MSB
  return result;
}

void Screen::begin() {
  this->tft.begin();
  this->tft.fillScreen(ILI9341_BLACK);
  this->tft.setRotation(0);
}

//a combined drawRect method which can also fill
//currently does not support both filling and drawing, or differing
    border and fill colors
void Screen::drawRect(uint16_t color, int x, int y, int w, int h,
    bool filled) {
  if (filled) {
    this->tft.fillRect(x, y, w, h, color);
  } else {
    this->tft.drawRect(x, y, w, h, color);
  }
}
```

```
37
38 void Screen::drawText(char* text, uint16_t color, int font_size, int
       x, int y) {
39   this->tft.setTextColor(color);
40   this->tft.setTextSize(font_size);
41   this->tft.setCursor(x, y);
42   this->tft.print(text);
43 }
44
45 //reads an image off of the SD and draws it at the same time.
46 //this assumes the file is a valid bmp and does not report errors
47 void Screen::drawImage(char* filename, uint8_t x, uint8_t y) {
48   this->pos = 0;
49   this->bmpFile = SD.open(filename);
50   Serial.println("opened bmp");
51
52   read16(); //ignore BMP magic number
53   read32(); //ignore creator bytes
54   this->bmpImageOffset = read32();
55   this->bmpImageOffset += read32(); //add header size to offset
56   Serial.println(this->bmpImageOffset, DEC);
57   this->bmpWidth = read32();
58   this->bmpHeight = read32();
59   Serial.println(this->bmpWidth, DEC);
60   Serial.println(this->bmpHeight, DEC);
61   read16(); //ignore number of planes, must be one.
62   this->bmpDepth = read16(); //must be 24.
63   read32(); //ignore compression, must be zero - uncompressed.
64   this->rowSize = (this->bmpWidth * 3 + 3) & ~3; //pad to four byte
       boundary
65
66   //handle the case of images stored top-down
67   if (this->bmpHeight < 0) {
68     this->bmpHeight *= -1;
69     this->rowSize *= -1;
70     this->flip = false;
71   }
72
73   //crop to tft size
74   this->w = this->bmpWidth;
```

```
75  this->h = this->bmpHeight;
76
77  if ((x + this->w - 1) >= tft.width()) {
78    this->w = tft.width() - x;
79  }
80
81  if ((y + this->h - 1) >= tft.height()) {
82    this->h = tft.height() - y;
83  }
84
85  this->tft.setAddrWindow(x, y, x + w - 1, y + h - 1);
86
87  //set initial position
88  //if we store upside down, we need to start from the other end
89  if(this->flip) {
90    this->pos = this->bmpImageOffset - (this->bmpHeight - 1) *
        this->rowSize;
91  } else {
92    this->pos = this->bmpImageOffset + this->rowSize;
93  }
94
95  //for each row
96  for (this->row = 0; this->row < this->h; this->row++) {
97    yield();
98    //seek to start of row
99    this->pos += this->rowSize;
100   if(this->bmpFile.position() != this->pos) { //do we need to seek?
101     this->bmpFile.seek(this->pos);
102     this->buffidx = sizeof(this->sdbuffer); //if so force buffer
        reload
103   }
104
105   //for each pixel
106   for(this->col = 0; this->col < this->w; this->col++) {
107     //do we need more data?
108     if(this->buffidx >= sizeof(this->sdbuffer)) {
109       this->bmpFile.read(this->sdbuffer, sizeof(this->sdbuffer));
          //reload buffer
110       this->buffidx = 0;
111     }
```

```
112
113         //copy over one pixel to the screen
114         this->b = this->sdbuffer[this->buffidx++];
115         this->g = this->sdbuffer[this->buffidx++];
116         this->r = this->sdbuffer[this->buffidx++];
117         this->tft.pushColor(this->tft.color565(r, g, b));
118       }
119     }
120
121     this->bmpFile.close();
122 }
123
124 //these functions read the raw position data from the tft.
125 //these work as the four pins act as a potential divider:
126 //for reading the y-value, we power y- and y+,
127 //leave x- floating and read from x+, where x+ acts as
128 //the wiper pin on a potentiometer between y- and y+
129 //the same goes for reading the x-value except with x and y swapped.
130
131 float TouchScreen::rawReadY() {
132   pinMode(YP, OUTPUT);
133   digitalWrite(YP, HIGH);
134   pinMode(YM, OUTPUT);
135   digitalWrite(YM, LOW);
136   pinMode(XM, INPUT);
137   pinMode(XP, INPUT);
138   digitalWrite(XM, LOW);
139   float x = analogRead(XP);
140   return x;
141 }
142
143 float TouchScreen::rawReadX() {
144   pinMode(XP, OUTPUT);
145   digitalWrite(XP, HIGH);
146   pinMode(XM, OUTPUT);
147   digitalWrite(XM, LOW);
148   pinMode(YM, INPUT);
149   pinMode(YP, INPUT);
150   digitalWrite(YM, LOW);
151   float y = analogRead(YP);
```

```
152   return y;
153 }
154
155 //this function reads a point from the tft, smooths out the noise,
156 //and determines whether or not the screen is being pressed.
157 void TouchScreen::readTouch(Point* pos) {
158   float x = rawReadX();
159   float y = rawReadY();
160   if (this->last.x >= THRESHOLD || this->last.y >= THRESHOLD
161       || x >= THRESHOLD || y >= THRESHOLD) {
162     //if we weren't touching the screen last time or this time, jump
          straight to the point
163     pos->y = y;
164     pos->x = x;
165   } else {
166     //otherwise apply a exponential smoothing low pass filter
167     pos->y = this->last.y + TOUCH_ALPHA * (y - this->last.y);
168     pos->x = this->last.x + TOUCH_ALPHA * (x - this->last.x);
169   }
170
171   pos->t = x < THRESHOLD && y < THRESHOLD;
172
173   this->last.y = pos->y;
174   this->last.x = pos->x;
175 }
176
177 void TouchScreen::waitTouch(Point* pos) {
178   readTouch(pos);
179   while (!pos->t) {
180     readTouch(pos);
181   }
182 }
```

POSITIONING.H

```
#include "CurieIMU.h"
#include "SD.h"
#include "Wifi.h"
#include "Barometer.h"
#include "Compass.h"
#include "Misc.h"

#define STEP_SIZE 0.74
#define SIGMA_1 2
#define SIGMA_2 2
#define SIGMA_CONST (SIGMA_1 + SIGMA_2) / (2.5066283 * SIGMA_1 *
    SIGMA_2)
#define SQUARE(x) ((x)*(x))
#define NUM_HOTSPOTS 3
#define GRID_SIZE 4096

class Position {
    Vector acc_position;
    Vector position_v;

    Wifi* wifi;
    Compass* compass;
    Barometer* barometer;

    File path_loss_file;
    File mac_file;

    int last;

    float read_path_loss(int x, int y, int floor, int hotspot);
    void read_mac(int hotspot, char* mac);
    float get_path_loss(int hotspot);
    double get_probability(int, int, int, int);

    public:

    Position(Wifi* wifi, Compass* compass, Barometer* barometer) {
        this->wifi = wifi;
```

```
38          this->compass = compass;
39          this->barometer = barometer;
40      }
41
42      void begin();
43      void step_interrupt();
44      void update();
45      Vector position();
46  };
```

POSITIONING.CPP

```
#include "Positioning.h"


//The Curie accelerometer can only have a function pointer as an argument
//when registering an interrupt handler, so we must create a plain function.
//the difficulty is that our actual handler is a method on Position,
//so we must save our copy of Position in a global variable. This means that
//only one copy of Position can be working at any given time.
Position* global_position;
void step_detected() {
    if (CurieIMU.stepsDetected()) {
      global_position->step_interrupt();
    }
}

void Position::begin() {
  CurieIMU.begin();
  CurieIMU.setStepDetectionMode(CURIE_IMU_STEP_MODE_SENSITIVE);
  //the BMI160 has three modes, NORMAl, SENSITIVE, and ROBUST. We need
  //to be sure that we catch every step, so we use SENSITIVE.
  CurieIMU.setStepCountEnabled(true);
  global_position = this;
  //set a function to be called every time a step is detected.
  CurieIMU.attachInterrupt(&step_detected);
  CurieIMU.interrupts(CURIE_IMU_STEP);
  this->last = 0;


  //we require files that show the path loss at each position for each hotspot,
  //and the mac addresses of each hotspot, so that we can use the wifi module to measure them.
  this->path_loss_file = SD.open("path_loss.dat");
  this->mac_file = SD.open("macs.dat");
```

```
33 }
34
35 //get predicted path loss from some hotspot at a given point.
36 float Position::read_path_loss(int x, int y, int floor, int hotspot)
       {
37   //We assume that x and y aren't going to be more than GRID_SIZE,
       and then we structure
38   //the file such that the there are 9 grids of size GRID_SIZE, for
       each floor and for each hotspot
39   //the path loss is stored as a 32 bit floating point number in
       these grids.
40   int index = x + GRID_SIZE*y + GRID_SIZE*GRID_SIZE*(NUM_HOTSPOTS *
       floor + hotspot);
41   this->path_loss_file.seek(index);
42   char buff[4];
43   this->path_loss_file.read(buff, 4);
44   //pretend to the compiler that we read a float and not four bytes
45   //by taking a pointer, casting it and dereferenceing. This is
       called type punning.
46   return *(float*)&buff;
47 }
48
49 void Position::read_mac(int hotspot, char* mac) {
50   //each mac address is seventeen characters long, so they are just
       packed together
51   //spaced by seventeen bytes
52   this->mac_file.seek(hotspot * 17);
53   this->mac_file.read(mac, 17);
54   //c strings require a null terminating byte, so we add one at the
       18th position.
55   mac[17] = 0;
56 }
57
58 //get the actual measured path loss
59 float Position::get_path_loss(int hotspot) {
60   char buff[18];
61   this->read_mac(hotspot, (char*)buff);
62   return (float)this->wifi->get_rssi((char*)buff);
63 }
64
```

```
65 //use our model to get the probability of a given point.
66 double Position::get_probability(int x, int y, int floor, int
       hotspot) {
67   //NUM_HOTSPOTS*floor + hotspot is the absolute number of the
       hotspot
68   //the original hotspot value is the hotspot per floor
69   float actual = this->get_path_loss(NUM_HOTSPOTS*floor + hotspot);
70   float predicted = this->read_path_loss(x, y, floor, hotspot);
71   float acc_predicted =
       this->read_path_loss((int)this->acc_position.x,
72                                    (int)this->acc_position.y,
73                                    (int)this->acc_position.floor,
74                                   NUM_HOTSPOTS*floor + hotspot);
75
76   float ba = SQUARE(actual - predicted) / SQUARE(SIGMA_1);
77   float papp = (Vector(x, y, floor) - this->acc_position).mag2() /
       SQUARE(SIGMA_2);
78   float bp = SQUARE(actual - acc_predicted) / SQUARE(SIGMA_1 +
       SIGMA_2);
79
80   return SIGMA_CONST * exp(-0.5 * (ba + papp + bp));
81 }
82
83 void Position::step_interrupt() {
84   //see how many steps have occured since we last checked.
85   //should be one, but we should check anyway.
86   int delta = CurieIMU.getStepCount() - this->last;
87   this->last += delta;
88
89   float heading = this->compass->heading();
90
91   //add the calculated velocity vector to our current position.
92   this->acc_position.x += delta * STEP_SIZE * sin(heading);
93   this->acc_position.y += delta * STEP_SIZE * cos(heading);
94 }
95
96 void Position::update() {
97   //update our vertical position
98   this->position_v.floor += this->barometer->delta();
99
```

```
100   double best = 0;
101   int bestx = 0;
102   int besty = 0;
103   double total;
104   int hs;
105
106   //go through a square of values from the acc_position, two
107   //standard deviations in either direction. Techincally,
108   //we should only compute a circle of radius 2sigma but
109   //that is more computationally expensive.
110   for (int x = (int)this->acc_position.x - 2*SIGMA_2;
111        x < (int)this->acc_position.x + 2*SIGMA_2; x++) {
112     yield();
113     if (x > 0 && x < GRID_SIZE) {
114       for (int y = (int)this->acc_position.y - 2*SIGMA_2;
115           x < (int)this->acc_position.y + 2*SIGMA_2; x++) {
116         if (y > 0 && y < GRID_SIZE) {
117           yield();
118           total = 1;
119           //for each hotspot multiply the probabilities together
120           for (int i = 0; i < NUM_HOTSPOTS; i++) {
121             total *= this->get_probability(x, y,
                 this->position_v.floor, i);
122           }
123
124           //if its the most probable so far, record its x, y and
                probability.
125           if (total > best) {
126             best = total;
127             bestx = x;
128             besty = y;
129           }
130         }
131       }
132     }
133   }
134
135   this->position_v.x = (float)bestx;
136   this->position_v.y = (float)besty;
137 }
```

```
138
139 Vector Position::position() {
140   return this->position_v;
141 }
```

GUI.H

```
#include "Screen.h"
#include "Positioning.h"

#define START_POINT 0
#define DISTANCE_THRESHOLD 10
#define NODE_THRESHOLD 10
#define MAX_NODE 2000

class GUI;

//this is a type describing which state the device's GUI is currently in
enum GUIState {
  STARTUP,
  WELCOME,
  MENU,
  LOADING,
  NAVIGATION
};

//an abstract class describing what a GUI page needs to be able to handle.
//each state is associated with one page.
class GUIPage {
  public:

  //when we enter this state from another state
  virtual void enter(GUIState from, GUI* gui) = 0;

  //draw stuff to the screen
  virtual GUIState render(Screen screen) = 0;

  //called when the screen is touched
  virtual GUIState handle_touch(Point point) = 0;

  //what is the state associated with this page?
  virtual GUIState page_state() = 0;
};
```

```
37
38
39 //A static page that simply displays an image while performing an
       action
40 //and then transfers to another state.
41 class StaticLoadingPage : public GUIPage {
42   //the image to display
43   virtual char* filename() = 0;
44
45   //the state to progress to afterwards
46   virtual GUIState next_state() = 0;
47
48   //the action to perform
49   virtual void action() = 0;
50
51   virtual GUIState page_state() = 0;
52
53   void enter(GUIState from, GUI* gui);
54   GUIState render(Screen screen);
55   GUIState handle_touch(Point point);
56 };
57
58 //the initial startup state.
59 class StartupPage : public StaticLoadingPage {
60   char* filename() {
61     return "startup.bmp";
62   }
63
64   GUIState next_state() {
65     return GUIState::WELCOME;
66   }
67
68   GUIState page_state() {
69     return GUIState::STARTUP;
70   }
71
72   void action();
73 };
74
75 //the welcome state.
```

```
76 class WelcomePage : public StaticLoadingPage {
77   char* filename() {
78     return "welcome.bmp";
79   }
80
81   GUIState next_state() {
82     return GUIState::MENU;
83   }
84
85   GUIState page_state() {
86     return GUIState::WELCOME;
87   }
88
89   void action();
90 };
91
92 //this is the loading page that is shown after the user has
93 //selected their destination.
94 class LoadingPage : public StaticLoadingPage {
95   char* filename() {
96     return "loading.bmp";
97   }
98
99   GUIState next_state() {
100    return GUIState::NAVIGATION;
101  }
102
103  GUIState page_state() {
104    return GUIState::LOADING;
105  }
106
107  void action();
108 };
109
110 //the menu state. this handles the first second and third menu
       states as one page
111 //once the user has selected all the values, we exit immediately to
       the loading state.
112 class MenuPage : public GUIPage {
113   //the values we have collected so far
```

```
114   char floor_v = 0;
115   int building_v = -1;
116   int room_1_v = -1;
117   int room_2_v = -1;
118
119   //do we need to draw again when we render?
120   bool redraw = true;
121
122   //are we displaying a keypad or dropdown?
123   bool building_up = false;
124   bool floor_up = false;
125   bool room_1_up = false;
126   bool room_2_up = false;
127
128   public:
129
130   void enter(GUIState from, GUI* gui);
131   GUIState render(Screen screen);
132   GUIState handle_touch(Point point);
133
134   //get a code for use in the navigation state
135   int get_location_code();
136   //reset the page so it can be used again after we are done with
        navigation.
137   void reset();
138
139   GUIState page_state() {
140     return GUIState::MENU;
141   }
142 };
143
144 //this is the main navigation state. It redirects back to the menu
      when the user is
145 //at their destination
146 class NavigationPage : public GUIPage {
147   int destination;
148   int next;
149   int current = START_POINT;
150
151   //the nodes we need to get to and in what order
```

```
152   unsigned short path[100];
153   //the instructions to the user at each node
154   unsigned char path_i[100];
155   //the total number of nodes
156   int path_len;
157   //where we are along the path right now.
158   int path_index = 0;
159
160   Position* position;
161
162   //read in the paths from file.
163   void read_path(int start, int end);
164   Vector node_position(int node);
165   int nearest_node();
166
167   float distance_next();
168   float distance_away(float dn);
169
170   //file handles for the node positions, and paths.
171   File node_file;
172   File path_file;
173
174   public:
175
176   void enter(GUIState from, GUI* gui);
177   GUIState render(Screen screen);
178   GUIState handle_touch(Point point);
179
180   GUIState page_state() {
181     return GUIState::NAVIGATION;
182   }
183 };
184
185 //the is the handler class that contains the main rendering loop.
186 //this loop is run by the scheduler to simulate multithreading.
187 class GUI {
188   Screen screen = Screen();
189   TouchScreen touch_screen = TouchScreen();
190   Point last;
191   Point touch;
```

```
192   GUIState current;
193
194   StartupPage* sp;
195   WelcomePage* wp;
196   LoadingPage* lp;
197   MenuPage* mp;
198
199   public:
200
201   void begin();
202   void loop();
203
204   GUIPage* get_page_with_state(GUIState state);
205 };
```

GUI.CPP

```
#include "GUI.h"

void GUI::begin() {
  this->screen.begin();
  this->current = GUIState::STARTUP;
  this->last = Point();
  this->sp = new StartupPage();
  this->wp = new WelcomePage();
  this->lp = new LoadingPage();
  this->mp = new MenuPage();
}

void GUI::loop() {

  //render the current page and see if we need to switch
  GUIState new_state;
  new_state =
      this->get_page_with_state(this->current)->render(this->screen);

  yield();

  //if we do, enter the new state from the current state
  if (new_state != this->current) {
    this->get_page_with_state(new_state)->enter(this->current, this);
    this->current = new_state;
  }

  //get the latest touch from the touch screen and see if we have a
      different touch event.
  this->touch_screen.readTouch(&this->touch);
  if (this->touch.t != this->last.t) {
    //check if we need to switch state after handling the touch
    new_state = this->get_page_with_state(this->current)
            ->handle_touch(this->touch);

    //if we need to, switch state
    if (new_state != this->current) {
      this->get_page_with_state(new_state)->enter(this->current,
```

```
          this);
37      this->current = new_state;
38    }
39  }
40 }
41
42 GUIPage* GUI::get_page_with_state(GUIState state) {
43   switch (state) {
44     case GUIState::STARTUP: return this->sp;
45     case GUIState::WELCOME: return this->wp;
46     case GUIState::LOADING: return this->lp;
47     case GUIState::MENU: return this->mp;
48     default: return this->sp;
49   }
50 }
51
52 void StaticLoadingPage::enter(GUIState from, GUI* gui) {
53   return;
54 }
55
56 GUIState StaticLoadingPage::render(Screen screen) {
57   screen.drawImage(this->filename(), 0, 0);
58   this->action();
59   return this->next_state();
60 }
61
62 GUIState StaticLoadingPage::handle_touch(Point point) {
63   return this->page_state();
64 }
65
66 void StartupPage::action() {
67   //do nothing while displaying the logo page
68 }
69
70 void WelcomePage::action() {
71   //do nothing while displaying the welcome page
72 }
73
74 void LoadingPage::action() {
75   //do nothing while loading the navigation page
```

```
76 }
77
78 void MenuPage::enter(GUIState from, GUI* gui) {
79   if (from == GUIState::NAVIGATION) {
80     //if we come here from navigation, we need to reset all our
         values, as we have just got to our destination.
81     this->reset();
82   }
83 }
84
85
86 //return a code identifying the destination we have selected
87 int MenuPage::get_location_code() {
88   //this simply encodes the options (building, floor, room) as a
       four digit number.
89   int ret = this->room_2_v + 10*this->room_1_v;
90   int floor;
91   if (this->floor_v == 'G') {
92     floor = 0;
93   } else if (this->floor_v == 'F') {
94     floor = 1;
95   } else if (this->floor_v == 'S') {
96     floor = 2;
97   }
98
99   return ret + 100 * (3 * this->building_v + floor);
100 }
101
102 void MenuPage::reset() {
103   this->redraw = true;
104   this->building_v = -1;
105   this->floor_v = 0;
106   this->room_2_v = -1;
107   this->room_1_v = -1;
108   this->building_up = false;
109   this->floor_up = false;
110   this->room_1_up = false;
111   this->room_2_up = false;
112 }
113
```

```
GUIState MenuPage::render(Screen screen) {
  if (this->redraw) {
    this->redraw = false;
    //if we need to redraw, draw the background image
    screen.drawImage("menu.png", 0, 0);

    //draw in the values we've already picked
    char* s_buffer = " ";
    if (this->floor_v > 0) {
      s_buffer[0] = floor_v;
      screen.drawText(s_buffer, ILI9341_BLACK, 146, 164, 10);
    }

    if (this->building_v >= 0) {
      s_buffer[0] = this->building_v + '1';
      screen.drawText(s_buffer, ILI9341_BLACK, 146, 102, 10);
    }

    if (this->room_1_v >= 0) {
      s_buffer[0] = this->room_1_v + '0';
      screen.drawText(s_buffer, ILI9341_BLACK, 146, 226, 10);
    }

    if (this->room_2_v >= 0) {
      s_buffer[0] = this->room_2_v + '0';
      screen.drawText(s_buffer, ILI9341_BLACK, 156, 226, 10);
    }


    //draw the dropdowns or numpad if required.
    if (this->building_up) {
      screen.drawImage("building.png", 0, 160);
    }

    if (this->floor_up) {
      screen.drawImage("floor.png", 0, 160);
    }

    if (this->room_2_up || this->room_1_up) {
      screen.drawImage("numpad.png", 0, 160);
```

```
154     }
155   }
156
157   if(this->building_v >= 0 &&
158        this->room_1_v >= 0 &&
159        this->room_2_v >= 0 &&
160        this->floor_v > 0) {
161       return GUIState::LOADING;
162       //if we have filled in all the fields, then we are done here
163   }
164
165   return this->page_state();
166 }
167
168 GUIState MenuPage::handle_touch(Point point) {
169   if (point.t) {
170     //if we need to handle a touch to a popup
171     if (this->building_up || this->floor_up || this->room_1_up ||
          this->room_2_up) {
172       //if we are handling a three button popup
173       if (this->building_up || this->floor_up) {
174
175         //if we are in the first button
176         if(point.y > 160 && point.y < 210) {
177           if (this->building_up) {
178             this->building_v = 0;
179             this->building_up = false;
180             this->redraw = true;
181           } else {
182             this->floor_v = 'G';
183             this->floor_up = false;
184             this->redraw = true;
185           }
186
187         //if we are in the second button
188         } else if (point.y > 213 && point.y < 263) {
189           if (this->building_up) {
190             this->building_v = 1;
191             this->building_up = false;
192             this->redraw = true;
```

```
193           } else {
194             this->floor_v = 'F';
195             this->floor_up = false;
196             this->redraw = true;
197           }
198
199         //if we are in the third button
200         } else if (point.y > 266 && point.y < 316) {
201           if (this->building_up) {
202             this->building_v = 2;
203             this->building_up = false;
204             this->redraw = true;
205           } else {
206             this->floor_v = 'S';
207             this->floor_up = false;
208             this->redraw = true;
209           }
210         }
211
212       //if we are handling a numpad
213       } else if (this->room_2_up || this->room_1_up) {
214
215         //check each row of the numpad
216         for (int row = 0; row < 3; row++) {
217           //if we are in the right row, check its columns
218           if (point.y > 160 + 40*row && point.y < 200 + 40*row) {
219             //check each column, using 1-3 instead of 0-2 because
                  the numbers on the numpad start at 1 and not 0.
220             for (int column = 1; column <= 3; column++) {
221               if (point.y > 60*column - 60 && point.y < 60*column) {
222                 //if we touch one and we are finding the first digit
                      of the room number
223                 if (this->room_1_up) {
224                   //take down the first numpad and put up the second
225                   this->room_1_v = 3*row + column;
226                   this->room_1_up = false;
227                   this->room_2_up = true;
228                 } else {
229                   //if this is the second digit, just take down the
                        numpad afterwards.
```

```
230                   this->room_2_v = 3*row + column;
231                   this->room_2_up = false;
232                 }
233                 this->redraw = true;
234                 return this->page_state();
235               }
236             }
237           }
238         }
239         //if we pressed 0.
240         if (point.y > 280 && point.x > 60 && point.x < 120) {
241           if (this->room_1_up) {
242             this->room_1_v = 0;
243             this->room_1_up = false;
244             this->room_2_up = true;
245           } else {
246             this->room_2_v = 0;
247             this->room_2_up = false;
248           }
249           this->redraw = true;
250         }
251       }
252     } else {
253       //if we are opening a popup
254       if (point.x > 146 && point.x < 210) {
255
256         //if we are opening the building selection
257         if (point.y > 102 && point.y < 114) {
258           this->building_up = true;
259           this->redraw = true;
260
261         //if we are opening the floor selection
262         } else if (point.y > 164 && point.y < 176) {
263           this->floor_up = true;
264           this->redraw = true;
265
266         //if we are opening the room selection
267         } else if (point.y > 226 && point.y < 238) {
268           this->room_1_up = true;
269           this->redraw = true;
```

```
270         }
271       }
272     }
273   }
274
275   return this->page_state();
276 }
277
278 void NavigationPage::read_path(int start, int end) {
279   //The path file is structured as a 900 by 900 grid of paths
280   //i.e each of the possible start nodes to each of the destination
        nodes
281   //each path is 301 bytes - 1 length byte, up to 100 pairs of bytes
        specifying
282   //nodes, and 100 bytes specifying the directions (left, right, up,
        down).
283   this->path_file.seek(301 * (start * 900 + end));
284   this->path_len = this->path_file.read();
285   this->path_file.read(this->path, 200);
286   this->path_file.read(this->path_i, 100);
287 }
288
289 Vector NavigationPage::node_position(int node) {
290   //the node file is structured as a list of x, y and floor
        coordinates for the nodes
291   //each section is 5 bytes long: two for the x, two for the y and
        one for the floor.
292   this->node_file.seek(4*node);
293   char buff[2];
294   this->node_file.read(buff, 2);
295   //use type punning to read the buffer into a integer that we cast
        to a float.
296   float x = (float)*(short*)buff;
297   this->node_file.read(buff, 2);
298   float y = (float)*(short*)buff;
299   return Vector(x, y, this->node_file.read());
300 }
301
302 int NavigationPage::nearest_node() {
303   int best = 0;
```

```
304   //initially, the closest node we can find is really far away.
305   //ideally this value would be infinity, but it is big enough
          practically.
306   float bd = 10000000000;
307   float d;
308
309   Vector pos = this->position->position();
310   Vector npos;
311
312   //search all the nodes for a closer one
313   //this is _really_ inefficient but its the only option we have
          without doing something complicated
314   //like a quadtree or a binary search tree.
315   //as the position is always changing we can't sort and do a binary
          search.
316   for(int i = 0; i < MAX_NODE; i++) {
317     npos = this->node_position(i);
318     d = (pos - npos).mag2();
319     if (d < bd) {
320       bd = d;
321       best = i;
322     }
323   }
324
325   return best;
326 }
327
328 float NavigationPage::distance_next() {
329   Vector npos = this->node_position(this->next);
330   Vector cpos = this->node_position(this->current);
331
332   //get the distance to the next node as shown by the formula in our
          final design.
333
334   Vector nc = npos - cpos;
335   float ncmag = sqrt(nc.mag2());
336
337   return ncmag - ((this->position->position() - cpos) & nc) / ncmag;
338 }
339
```

```
340 float NavigationPage::distance_away(float dn) {
341   Vector cpos = this->node_position(this->current);
342   float cpmag2 = (this->position->position() - cpos).mag2();
343
344   //use pythagoras' theorem to compute the distance away from the
          path.
345
346   return sqrt(cpmag2 - dn*dn);
347 }
348
349 void NavigationPage::enter(GUIState from, GUI* gui) {
350
351   //get our destination by reading from the state of the menu page.
352   this->destination =
353     ((MenuPage*)gui->get_page_with_state(GUIState::MENU))
354       ->get_location_code();
355   this->path_file = SD.open("paths.dat");
356   this->node_file = SD.open("nodes.dat");
357
358   this->read_path(this->current, this->destination);
359   //set our initial node to find.
360   this->next = this->path[this->path_index];
361 }
362
363 GUIState NavigationPage::render(Screen screen) {
364   float dn = this->distance_next();
365   float da = this->distance_away(dn);
366
367
368   //load the appropriate image given the instruction to the user:
369   //the images are stored as "nav1.bmp", "nav2.bmp" etc with each
          number
370   //corresponding to a direction - left right up down.
371   char* nimg = "nav_.bmp";
372   nimg[3] = '0' + this->path_i[this->path_index];
373   screen.drawImage(nimg, 0, 0);
374
375   //draw the distance to the node underneath the arrow
376   char buff[2];
377   itoa((int)dn, buff, 10);
```

```
378   screen.drawText(buff, ILI9341_WHITE, 136, 256, 20);
379
380   Vector npos = this->node_position(this->next);
381   Vector pos = this->position->position();
382
383
384   //if we've got to the node
385   if (dn < NODE_THRESHOLD && npos.floor == pos.floor) {
386     this->current = this->next;
387     this->path_index++;
388     //if we've finished, go bach to the menu;
389     if (this->path_index >= this->path_len) {
390       return GUIState::MENU;
391     }
392     //otherwise select the next node
393     this->next = this->path[this->path_index];
394   }
395
396   //if we've gone to far astray
397   if (da > DISTANCE_THRESHOLD) {
398     //get our nearest node
399     this->current = this->nearest_node();
400     this->path_index = 0;
401     //and restart navigation from there.
402     this->read_path(this->current, this->destination);
403   }
404
405   return this->page_state();
406 }
407
408 GUIState NavigationPage::handle_touch(Point point) {
409   //the user cannot interact with the navigation page.
410   return this->page_state();
411 }
```

MISC.H

```
#include "SD.h"

#define SD_CS 2

void begin_misc();

//our vector class. This is used in multiple places,
//so we use floats for x and y as this is compatible with both
   floats and ints, as
//both are used in various contexts.
class Vector {
    public:

    float x = 0;
    float y = 0;
    int floor = 0;

    Vector() {
        this->x = 0;
        this->y = 0;
        this->floor = 0;
    }

    Vector(float x, float y) {
        this->x = x;
        this->y = y;
        this->floor = 0;
    }

    Vector(float x, float y, int floor) {
        this->x = x;
        this->y = y;
        this->floor = floor;
    }

    //an efficient specialization of + for in place addition.
    Vector& operator+=(const Vector& other) {
        this->x += other.x;
```

```
38          this->y += other.y;
39          this->floor += other.floor;
40          return *this;
41      }
42
43      Vector operator+(Vector other) {
44          return Vector(this->x + other.x, this->y + other.y,
                this->floor + other.floor);
45      }
46
47      Vector operator-(Vector other) {
48          return Vector(this->x - other.x, this->y - other.y,
                this->floor - other.floor);
49      }
50
51      //square of this vectors magnitude, ignoring the floor as this
            is accounted for seperately.
52      double mag2() {
53          return this->x * this->x + this->y * this->y;
54      }
55
56      //vector dot product
57      double operator&(Vector other) {
58          return this->x * other.x + this->y * other.y;
59      }
60  };
```

MISC.CPP

```
1  #include "Misc.h"
2
3  void begin_misc() {
4    SD.begin(SD_CS);
5  }
```

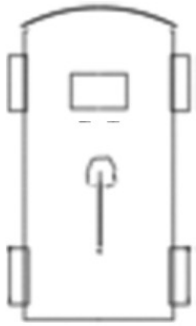
11

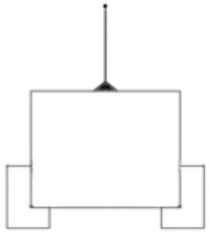
APPENDIX C - INITIAL DESIGN IMAGES

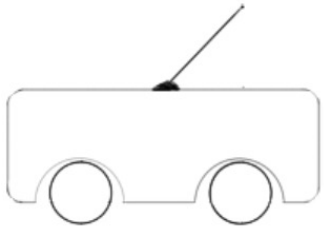
<u>EES Intel Indoor Navigation Project:</u>

Device Ideas: Ground based Robot (autonomous/semi-autonomous) Design and preliminary Evaluation

Feasibility:
- Takes up a moderate amount of space, but control points could store a robot as well
- Will cost a considerable amount
- Not very safe as it may run into things, trip over people in the corridors and/or block peoples paths
- More advanced manufacturing required
- External support is needed as it is an autonomous device (if with control points)
- Pre-Programming needs to be good as the robot will otherwise miss turns and hit walls, causing damage





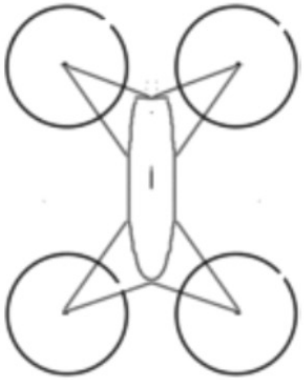


Description:
- An autonomous robot on wheels that will guide the visitor (much like the drone) to their pre-chosen destination
- Destination entered at a control point at any main location e.g. reception or canteen/cafeteria
- Ground based robot is able to carry more tech (than the drone) hence could be controlled from the robot itself (control points then unnecessary)

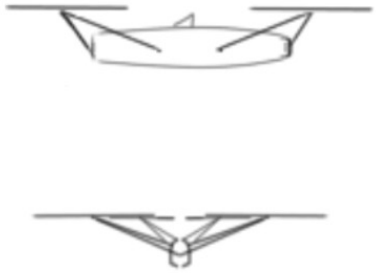
<u>EES Intel Indoor Navigation Project:</u>

Device Idea: Drone (autonomous)

Design and preliminary Evaluation



Feasibility:
- Takes up a lot of space (storage difficulties)
- Will cost a considerable amount
- Not very safe as it will be hitting walls and people, i.e. very dangerous indoors
- More advanced manufacturing required
- External support is needed as it is an autonomous guide and can not carry much tech.
- Pre-Programming needs to be very good as the drone will otherwise miss turns and hit walls causing damage

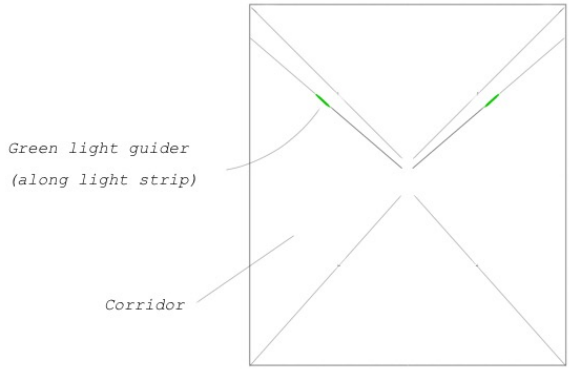


Description:
- An autonomous drone that will guide the visitor to their pre-chosen destination
- Destination entered at a control point at any main location e.g. reception or canteen/cafeteria
- Control points will need to have control of the drone as the drone can only carry a limited amount of technology (routes must be pre-programmed)

EES Intel Indoor Navigation Project:

Device Ideas: Light Strip Guidance (autonomous)

Design and preliminary Evaluation



Feasibility:
- Medium Cost
- No storage needed (however care must be taken so as to make sure the light strip does not obstruct exits or doorways, hence it will be placed close to the ceiling)
- Safe to use
- Easy to manufacture (just quantity)
- Would not use Intel board (all routes would need to be pre-programmed)
- Lighting requires specific programming
- Simple and easy to use
- Transferability is poor as amount of light strips depends on the b and its size and can vary largely.

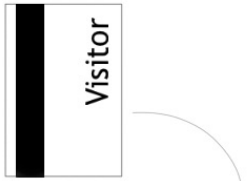
Description:
- Light strips are put along the walls indoors and at the start of the navigation a green light will appear at the nearest (appropriate for your navigation) light strips position to you and start travelling along the strip at a walking pace. You will follow the green light and avoid red lights (i.e. other, unwanted directions). The green light will lead you to your chosen destination

EES Intel Indoor Navigation Project:

Device Ideas: Visitor Card Scanning System ((semi-)handheld) Design and preliminary Evaluation

Description:

- The visitor receives a card from reception, then makes their way to a scan point (placed at locations throughout the building) where they can scan there visitors card, which will open an info tab on the scan point where the visitor may see their location, interesting things in their immediate vicinity and program a destination. Programming a destination will cause all the other scan points to show the visitor the direction they need to go to reach their destination.



Visitors Card

Visitors Card Scan Point
(touchscreen interface or non-touch)

Feasibility:

- Medium cost (scan points will cost most)
- No storage (apart from minimal for cards)
- Safe to use
- Easy to manufacture
- Will require an extensive network and good programming
- Easy to use (though possibly tedious)
- Transferability is restricted as the scan points are relatively permanent, however they are more transferable than e.g. the light strips

EES Intel Indoor Navigation Project:

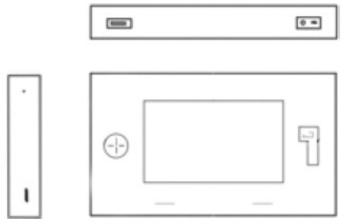
Device Ideas: Touchscreen/Non-Touch Device (handheld)

Design and preliminary Evaluation

Touchscreen

Description (touchscreen):
- Handheld device with a touchscreen for easy management and good interactivity
- Map of current building will be projected on to the screen and the destination can be chosen with one tap

Feasibility (both):
- Low cost
- Easily stored (however screens will need protection)
- Safe to use
- Simply manufactured
- Intel board and other tech will fit inside, hence only minimal external extras will be needed
- Digitalised floor plans will be needed to enable interactivity
- Easy to use



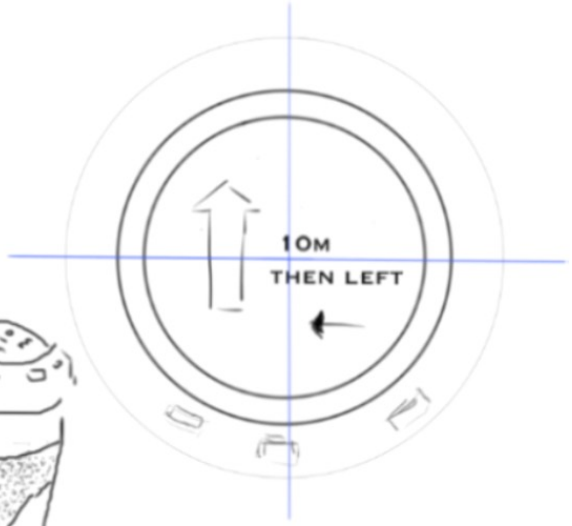
Non-Touch

Description (non-touch):
- Handheld device with a screen and physical buttons for medium management and medium interactivity
- Map of current building will be projected on to screen and the destination can be chosen with direction buttons

EES Intel Indoor Navigation Project:

Device Idea: "Navi-Grip" (handheld)

Design and preliminary Evaluation



Feasibility:

- Low cost
- Easily stored
- Safe to use
- Will require more advanced manufacturing; grip shape, overall device shape
- Board will not fit inside device (external support needed)
- Programming needed to give live feedback for direction and distance function

Description:

- 3 button control
- Small, round interface where simple menus and directions are projected
- Will need more external extras as the Intel board will not fit inside the device

EES Intel Indoor Navigation Project:
Device Ideas: Navi-Grip v.2 (handheld)
Design and secondary Evaluation
Touchscreen,
e.g. Aadafruit 2.8" TFT Cap
Charge port,
e.g. USB
Board: Genuino 101
Compass,
e.g. Aadafruit trials
Battery (rechargeable),
e.g. Aadafruit 4400mAh 3.7V Lipo
Hinged grip with set angles (clicks) for maximum comfort and adaptability
Body material,
e.g. PLA
Ergonomic rubber grip,
e.g. TPU
Battery charger,
e.g. Aadafruit USB charger
Front
Topside