Final Year Project Report

Robot mapping in low-visibility conditions using low-cost radar modules

June 14, 2022

Excerpt from the "ME40321 Engineering project instructions 2020/21" handout, page 12

APPENDIX E. Layout for title page of final year project report

UNIVERSITY OF BATH

Department of Mechanical Engineering
FACULTY OF ENGINEERING AND DESIGN

FINAL YEAR MEng PROJECT REPORT

Robot mapping in low-visibility conditions using low-cost radar modules (Alan Hunter) (ID 489)

Hugo Weston

2022/05/04

Word count: 11675

"I certify that I have read and understood the entry in the Student Handbook for the Department of Mechanical Engineering on Cheating and Plagiarism and that all material in this assignment is my own work, except where I have indicated with appropriate references."

Author's signature: Hugo Weston

Supervisor: *Dr Alan Hunter*

Assessor: *Dr Charles Courtney*

Candidate:	Hugo Weston
Supervisor:	Dr Alan Hunter
Assessor:	Dr Charles Courtney
Word Count:	11675

Summary

New advances in radar technology has resulted in commercially available small scale and low power coherent radar sensors. This new radar technology may have application in navigation. Current solutions for small-scale navigation involve the use of cameras and/or lidar, which can fail to provide accurate data in low visibility conditions such as smoke or fog. Radar sensors are generally unaffected by these adverse conditions. Therefore, project aims to determine the feasibility of using this new technology for navigation applications. This was done by applying techniques typically used for sonar navigation. Experiments took the form of mounting the sensors in a side-looking radar configuration on a custom-built piece of testing apparatus. The quality of data from the sensors was evaluated, as well as determining how different factors affect quality of this data.

It was found that the current generation of sensors are generally lacking the capabilities required for complex navigation. However, a new generation of sensors shows considerable improvement. A custom method, built on the foundations of a similar method in synthetic aperture sonar, is presented that uses an array of 3 sensors to estimate distance travelled in 1 direction with reasonable accuracy. This method would benefit from more testing over longer distances and different environments. This method is not applicable when changes in orientation are considered. It is therefore suggested that this method could be integrated into current navigation algorithms to assist other sensing methods in low-visibility conditions.

Synthetic aperture radar, a processing technique only possible with coherent radar, is experimented with in this project. Qualitative analysis shows that this technique is able to determine the approximate size and shape of obstacles. However, highly accurate knowledge of position is required for this method, and thus may prohibit use in autonomous navigation systems.

Acknowledgements

This project was supported by Dr Alan Hunter and Dr Ben Thomas, who assisted by providing their expertise.

Dr A J Hunter	Senior Lecturer	Dept. of Mechanical Engineering	University of Bath
Dr B W Thomas	Research Associate	Dept. of Mechanical Engineering	University of Bath

Contents

List of Figures

List of Tables

1 Introduction

Recent advances in radar sensing technology have enabled production of coherent, low-cost, small radar sensors. This project utilises coherent sensors manufactured by a company named Acconeer. The coherence of these sensors facilitate complex data processing techniques, which may give these sensors utility in navigation applications.

Most small-scale navigation algorithms utilise cameras and/or lidar as sensing methods. Both of these sensing methods are disrupted by low visibility conditions such as smoke, fog, or other particulates in the air. Radar is unaffected by these environmental conditions, and so could be utilised to augment current navigation methods to improve performance in adverse conditions.

This report aims to asses the feasibility of using small-scale coherent radar sensors for navigation and mapping. Techniques used in sonar technology are applied to these sensors to estimate distance travelled by an array of sensors. This project only examines distance-travelled estimation in 1 direction, with no changes in orientation. It is proposed that the distance estimation method could be expanded to 3 dimensions with relative ease, but changes in orientation remain a challenge to this method. Feasibility of the method is assessed by examining the errors in distance estimates, as well as how factors such as different operating environments could negatively impact navigation. It is found that while the current technology is somewhat inadequate for most navigation applications, the next generation of these sensors are capable of providing adequate localisation estimates.

This report also explores the use of synthetic aperture radar to produce high quality images of a surrounding environment. Qualitative analysis of images produced by the sensors shows that the general size and shape of obstacles are able to be determined.

Section 2 reviews current literature for the topics of robot navigation and radar, as well as techniques from synthetic aperture sonar. General background information required in this report is introduced in section 2. Section 3 details the different experiments in this project. Experimental data is presented in Section 4, which is reviewed in Section 5, where the relevance to potential applications is investigated. Section 6 summarises the main findings of this project. Section 7 details possible next steps for developing navigation algorithms for these sensors.

2 Background & Literature Review

2.1 Robot Navigation

The problem of robot navigation is generally split up into three distinct tasks [1]:

- Localisation. The task of determining where a robot is located relative to surroundings.
- Mapping. Processing known information into a representation of the surroundings.
- Path planning. Deciding how to move through the surroundings.

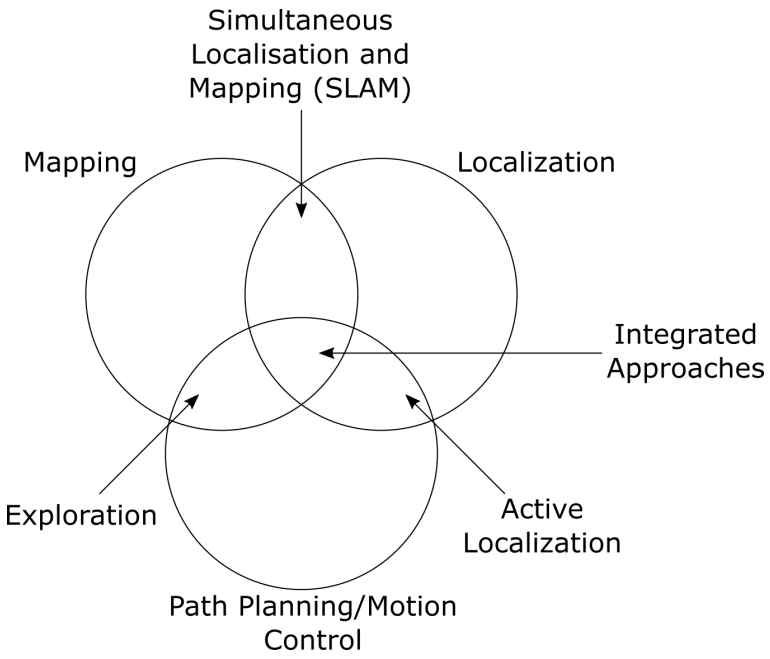

Figure 1: A Venn diagram showing different methods for completing the 3 tasks required for robot navigation [1] [2]

Full autonomous navigation requires completing all tasks simultaneously. Various navigation methods are presented in Figure 1 [2]. Simultaneous localisation and mapping has received significant attention from industry as it is considered an essential prerequisite for autonomous navigation. For both mapping and localization, a robot is required to sense the surrounding environment. Different sensing methods can be used. Common sensing methods are:

- Optical/Infra-red Cameras
- Sonar/Ultrasonics
- Lidar

These sensing methods can be used to solve the robot navigation tasks. However, sensing methods such as cameras and lidar are ineffective in low-visibility conditions, such as smoke or fog. In these conditions, other methods of navigation such as GPS (or other time of flight systems) can be used.

2.2 Current Navigation Methods

An area of navigation that has seen considerable development is simultaneous localisation and mapping (SLAM). SLAM techniques usually consist of recognising features of an environment and mapping these onto a grid. Common SLAM methods involve using particle filters to estimate robot motion. These estimates are constantly re-evaluated using data from the sensors. [1]

All navigation techniques require some form of input data. Cameras are favoured for their low cost, and lidar is favoured for high scanning rates [3]. However, both these sensing methods perform poorly in low visibility environments such as in smoke or fog, due to particulates in the air. Because of this, new algorithms have been developed that in order to combat this. Algorithms exist that automatically assess the quality of sensor data and decide whether to use it or not. Although this method improves the accuracy of navigation in robots, the low visibility conditions still significantly affect the quality of navigation [4].

A more promising method to improve navigation is to use sensors that are less affected by particulates in the air. Experiments show that integrating sonar sensors into robots can improve navigation, but results

are still prone to error. A 2015 article suggests that these errors are caused by sensor hardware limitations, and further suggests that more optimal results may be achieved using ultra-wideband radar sensors [3].

Radar has previously been integrated into feature-based SLAM algorithms. This has been successful in providing accurate results, but is computationally intensive [5] [6]. The new availability of coherent radar sensors enables new data processing methods, potentially improving navigation.

2.3 Radar Sensors

Radar (Radio Detection and Ranging) sensors use an antenna to send out pulses of electromagnetic radiation. Reflections of the transmitted signal can be measured and used to detect objects. Radar signals consist of radio waves, which generally have frequencies between 3KHz and 3000GHz [7]. Radar antennae send out signals over an angle, meaning that not all reflections to the sensor will come from objects directly in front of it. Figure 2 illustrates how strength of the signal may vary as a function of the angle from the sensor, where 0° is the direction the sensor is facing. The distance from (0,0) represents the strength of the signal. It can be seen that most of the signal strength is contained within the "Main Lobe", concentrated around 0°, but signal is also transmitted at larger angles in the sidelobes. Most signal strength is concentrated in the main lobe, and thus the sidelobes can generally be neglected [8].

The main lobe has a width that can be measured as an angle. This is either measured between the start and end points of the main lobe, θ_{null}, or between the points where the signal is half the maximum value, referred to as θ_{3dB} or the half-power beamwidth (HPBW) [8]. Both of these angles quantify how wide or narrow the main lobe is - the beamwidth. The value of the beamwidth is determined by the construction of the radar antenna. However, beamwidth can be modified by focussing the beam through a lens.

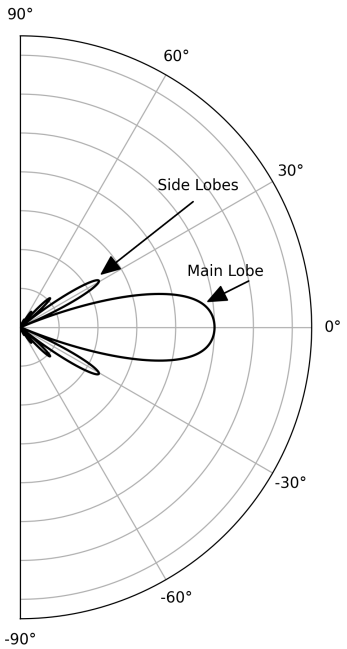

Figure 2: Generic radiation pattern of a radar antenna

2.4 Available Radar Sensors

Acconeer is a company that has developed new patented radar technology in the form of the A111 and A121 sensors. These sensors are able to produce coherent data - meaning that the phase difference between the

transmitted and received wave is known. This feature is not new to radar technology; coherent radar systems are not uncommon, but coherent systems are often large. However, Acconeer sensors have a small size (5.5mm x 5.2mm) and low power consumption (approximately 20mW) [10]. The combination of coherence, small size, and power efficiency mean that Acconeer sensors can be utilised in a wide variety of new applications that may not have been possible previously. One of these potential applications is in robot navigation. The coherence of Acconeer sensors make it possible to gain enough information from an environment to successfully navigate it.

Radar systems are generally large and utilised on satellites or planes for mobile operation. This presents a challenge as much of the literature available for radar is written from this large-scale perspective [8] [11] [9]. There is a clear gap in literature regarding radar on a small-scale. Therefore, several methods in this project rely on literature from synthetic aperture sonar.

2.5 Cross-Correlation

Cross-correlation is a commonly-used method used to compare the similarity of two datasets. Cross-correlation is integral to most aspects of this project. It should not be confused with the correlation coefficient, which is a measure of the strength of association between two datasets. Cross-correlation involves moving one set of data over the other and summing the products of the overlapping points. When a signal is correlated with itself, the correlation will have a maximum value [12]. This project utilises normalised correlations to measure the similarity of different sets of data. A cross-correlation value of 1 means the two correlated sets of data are identical, whereas a correlation value close to 0 means that the datasets are not similar at all. When correlation is mentioned in this report, it will always refer to cross-correlation unless stated otherwise.

2.6 Distance Estimation

Sonar can be used to generate high resolution images using Synthetic Aperture Sonar (SAS). SAS systems are designed to run at a constant velocity. However, disturbances such as waves and currents can result in a non-constant velocity. Non-constant velocity results in the SAS system being in a different position than expected. This can degrade the quality of SAS images. A method has therefore been developed with the goal of measuring these disturbances in position using sonar sensors. This information can then be used in post processing to increase the quality of SAS images [13]. Interestingly, a US patent exists for this method, although it restricted for use with sonar on an underwater vehicle [14].

There has been little work on applying this method to radar, because uncertain positioning is not an issue experienced with the typical use of large scale radar on planes (where GPS data is available) or satellites (where the position can be calculated to a high degree of accuracy). This project aims to use this method to estimate distance travelled by an array of sensors.

A brief explanation of the method is given here for context. Suppose a sensor gathers a set of data at a position in space. If the dataset is correlated with itself, it will produce a maximum value of 1. This data will be designated as the datum. If the sensor continues to gather data while moving away from this datum, it is expected that the signals will become less similar to the datum; the corresponding correlation values will decrease. The decrease in correlation follows a predictable pattern with respect to distance [13]. This relationship can be exploited to calculate distance travelled using a correlation value between two datasets. This method is summarised in Figure 3 - the “distance moved” may be calculated from datasets 1 and 2 [13].

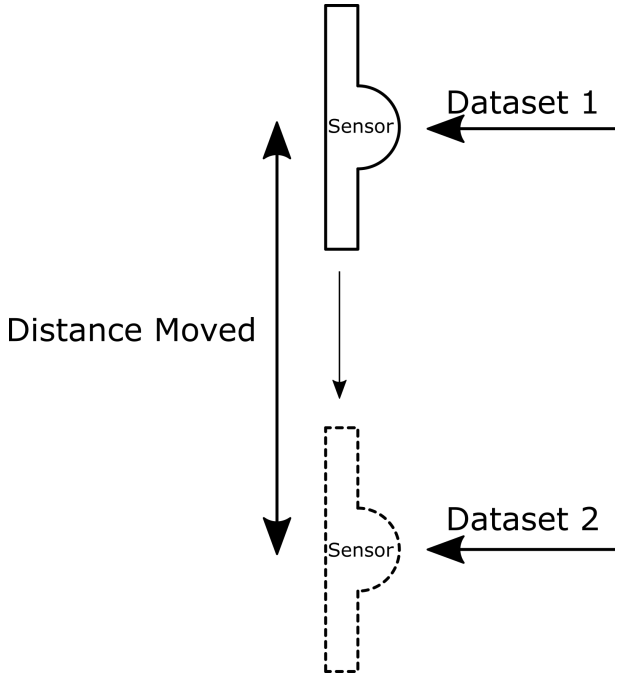

Figure 3: Basic explaination of distance estimation method. "Distance moved" may be calculated from datasets 1 & 2.

2.7 Synthetic Aperture Radar (SAR) Imaging

High accuracy maps can be generated by a technique called Synthetic Aperture Radar (SAR). This technique uses a small antenna to collect data at multiple points in space. This data can then be processed in a way that combines the data as if it were collected by a single, larger antenna. This gives a higher resolution image. This technique is typically applied to data gathered from planes or satellites. However, there are several hobbyist examples of synthetic aperture radar being applied on a smaller scale [15] [16]. The hobbyist nature of these examples reflects the lack of currently available small and coherent radar systems. The SAR processing technique used in this project is the "Delay and Sum" beamforming technique [17]. For each data point, the signal is projected outwards in a circular manner:

2.7.1 Minimum Length Requirement

To ensure an optimal SAR image, a minimum distance needs to be travelled by the sensor, shown in Equation 1, where x is the minimum distance required, r is the range from the sensor to the object, and θ is the half-power beamwidth. This is to ensure that all possible data has been added to the image. This requirement is derived from the geometry in Figure 4.

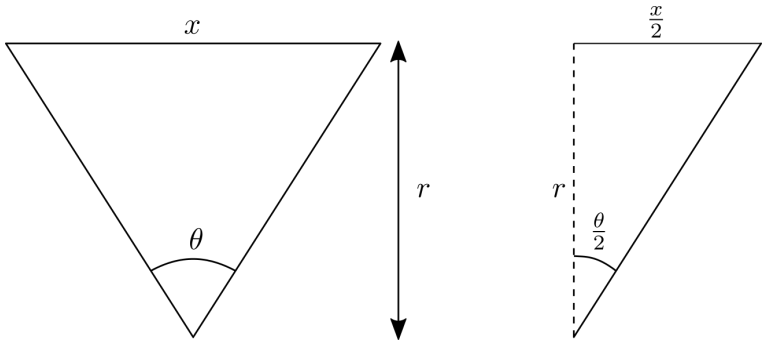

Figure 4: Minimum distance travelled needed for synthetic aperture radar

$$x = 2r \cdot \tan\left(\frac{\theta}{2}\right) \tag{1}$$

For a point to be properly mapped, all possible radar data must have been added to the point. This means that the length of properly focussed image will be equal to the distance travelled minus the minimum length requirement.

2.7.2 Maximum Sampling Spacing

The maximum spacing between samples taken should be spaced at no more than L/2, but ideally below L/4, where L is the effective physical aperture length. This is neither actual antenna length nor the synthesized antenna length, rather an antenna length that would produce a beamwidth equal to the current beamwidth. This allows for the use of lenses to focus the beam [18]. L is given by Equation 2, where λ is the wavelength of the signal and θ_{3dB} is the half-power beamwidth [18].

$$L \approx 0.88 \cdot \frac{\lambda}{\theta_{3dB}} \tag{2}$$

3 Experimental and Computational Methods

Before experimentation commenced, a health and safety evaluation was conducted. Electrical hazards and lone-working were identified as the key risks. All equipment passed Portable Appliance Testing (PAT) before usage.

3.1 Simplification to 1-Dimension

The goal of this project is to use a data processing method to determine how far a sensor (or an array of sensors) has moved in-between collecting two sets of data. In this project, movement will only be considered in 1 direction. This simplifies the experimental apparatus and data processing considerably. A distance estimation algorithm will be developed for this 1-dimensional case. In future, this algorithm could be expanded into 3 dimensions to localise a robot in a 3D environment.

The co-ordinate system for these experiments is shown in Figure 5. Displacements will be made to a sensor (or an array of sensors) in the along-track direction. The sensors will point in a direction perpendicular to the along-track direction. The system will be at a certain height above the ground. Height and across-track direction will be kept constant throughout experiments, while along-track displacement may be varied.

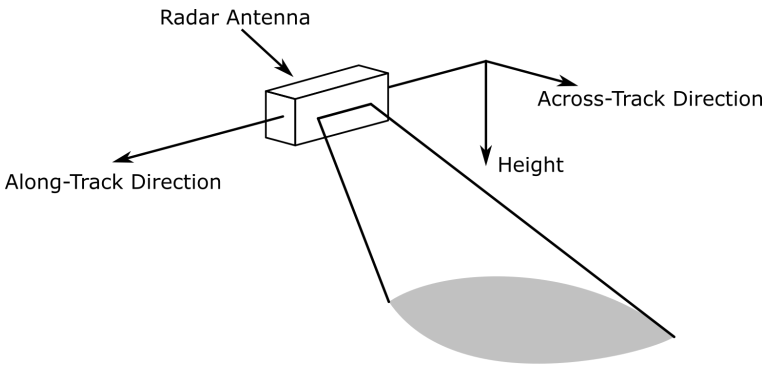

Figure 5: Coordinate system used in all experiments

This coordinate system is often used in SAR data gathering. It is referred to as "side looking radar" because the radar sensor is positioned perpendicular to the direction of travel [11]. Obviously a system such as this is not able to detect objects in the direction of travel; a robot fitted with this system could not tell if an obstruction was immediately ahead of it. Thus, any application of the algorithms developed in this project would likely need to be supplemented by existing navigation systems.

3.2 Sensors

All sensors used in this project are manufactured by Acconeer. Two types of sensor are used. These are referred to by product codes: A111 and A121. Both sensors are integrated circuits containing radar transmit and receive antennas. The A111 sensors are available for purchase from Acconeer. The A121 sensors are an updated version of the A111 sensors. A121 sensors are still in development by Acconeer, and thus are yet to be available to the public. Both sensor types are integrated into a development printed circuit board (PCB) module (XR112), which allows use of a raspberry pi to interface with the sensors. These modules have several useful in-built features, such as hardware averaging that averages multiple sets of data within the bare metal of the module, speeding up processing times [10]. Acconeer lenses and lens holders are also used in this project. These enable users to modify the beamwidth [19]. Acconeer equipment is shown in Figure 6. Four A111 sensors and three A121 sensors were used in this project.

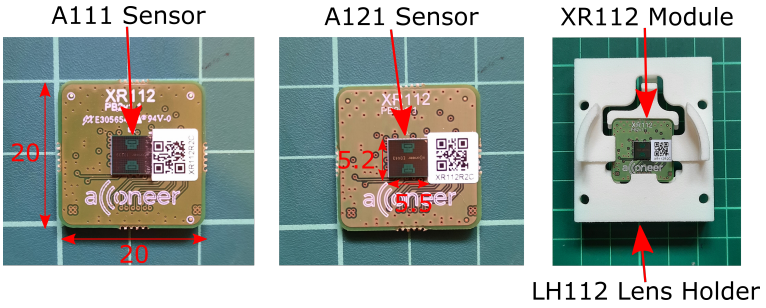

Figure 6: Images of the two different sensor types, A111 (left) & A121 (middle) integrated into the XR112 PCB module, which can be used in an Acconeer lens holder (right)

As mentioned in section 2.4, the most significant features of these sensors are that they are small and coherent. The coherence of the modules is what makes this project possible. Coherence allows the sensors to measure the phase difference between the transmitted and received waves. The phase data of a reflected wave is unique to the object that reflected it. This feature is required for both the localisation and mapping solutions discussed in this report. When using the coherent operating mode, the sensors have a maximum detection range of 7.0m, which limits usage of the sensors to small-scale applications.

Coherence has further applications as it can be used to detect Doppler-shift which can be used to estimate the speed of moving objects. This also has applications in robot navigation, but is beyond the scope of this report. Acconeer A111 sensors are available for purchase on Mouser Electronics, as shown in Table 1.

Table 1: Cost of Acconeer hardware used in this project from Mouser [20]

Part Number	Part Description	Part Cost	Quantity	Total Cost
LH112	Lens Kit	£30.65	4	£122.60
XR112	Radar Sensor	£38.26	4	£153.04
XC112	Raspberry Pi Connector Card	£156.36	1	£156.36
				£432.00

3.2.1 Lens

As mentioned in section 2.3, the signal transmitted from the antenna has a particular beamwidth. This beamwidth can be modified using Acconeer lenses. These are provided with the sensors. However, manufacturing custom lenses for these sensors is also possible. The possible beamwidths are shown in Table 2.

Table 2: Achievable beamwidths using different lens configurations [21]

Half Power Beam Width (Horizontal)	Lens	Position
80°	None	-
20°	Hyperbolic Lens	D1
12°	Hyperbolic Lens	D2

3.2.2 Sensor Frequencies

Table 3: Sampling frequencies of the sensors

Sensor Type	Sampling Frequency (GHz)	Distance per Sample (mm)
A111	619	0.45
A121	120	2.5

The sensors used in this project transmit a raised cosine oscillating at 60 GHz [10]. The sampling frequencies of both sensors are given in Table 3. The Nyquist frequency is the highest frequency of signal that can be measured using a given sampling frequency. If signals with frequencies above the Nyquist frequency are measured, aliasing starts to become an issue, which results in unreliable data [12].

Both sensor types fulfil this requirement. The A121 sensors only just fulfil this requirement, whereas the A111 sensors sample at a significantly higher frequency than required. This could potentially make the A111 sensors susceptible to high frequency noise.

Assuming that the sensors transmit a cosine at exactly 60 GHz, sampling at any frequency higher than 120 GHz will not provide any more information as per the Nyquist frequency. However, it should be noted that the cosine frequency may vary between 57-64 GHz depending on temperature, therefore a maximum sampling frequency of 128 GHz is optimal [10].

3.2.3 Sensor Signals

The sensors used in this project are capable of transmitting 5 different lengths of preconfigured signal. Signals vary in length from shortest (Signal 1) to longest (Signal 5). All signals are available to the A111

sensors, whereas only signals 2,3,4 are available to the A121 sensors. Signal length is directly related to the resolution of any gathered data, and will therefore impact any distance estimation. The range resolution of a radar is defined in Equation 3, where ρ is the minimum distance for which the radar can detect two separate objects, c is the speed of light, and t_{pulse} is the duration of the pulse [11]. Two objects closer than this resolution will appear as a single reflector.

$$\rho = \frac{c \cdot t_{pulse}}{2} \tag{3}$$

Equation 3 shows that reducing the pulse length allows a radar to detect more objects (or more fine detail). This is advantageous when using the sensors for navigation. However, a shorter pulse also means a lower energy of transmitted signal, which can give a poor signal-to-noise ratio (SNR). Therefore, the different available signals should be experimented with to understand which give the best performance for this application. Acconeer recommends that signals 4 & 5 are only used for presence detection as these have especially long signal lengths. Therefore, signals 4 and 5 do not appear in any experimental results.

3.3 Test-Rig

The experiments in this project require gathering data from the sensors at different positions in 1 direction. A test-rig was built to facilitate this. The design consists of a wheeled carriage on a v-slotted aluminium extrusion. The carriage is positioned by a NEMA-17 stepper motor (RS components: 180-5279) and a toothed belt. This is a common design in 3D printers and small laser cutters where accurate positioning of a payload is required. A full CAD model was developed to aid design and manufacture, shown in Figure 7. Design requirements are identified in Appendix 9.1. All custom parts used were 3D printed from PLA material.

Although the aluminium extrusion is 1000mm long, only 700mm of movement is possible due to the legs and carriage taking up space on the aluminium extrusion. The maximum range of movement could easily be extended by using a longer aluminium extrusion and belt. The carriage has two mounting points top and bottom to attach sensors and other equipment. This mounting point allows different attachments to be designed and used without having to change the carriage.

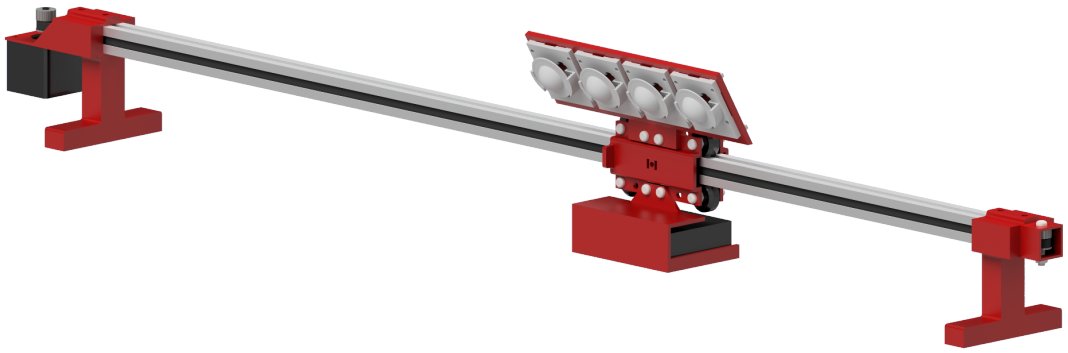

Figure 7: A render of the CAD model

The electronics are assembled on a solderless breadboard (Figure 8). The test-rig is driven by an Arduino Uno and a DRV8825 stepper motor driver. The power supply is a 12V DC adapter rated up to 3A. A user

can interface with the test-rig via a USB serial connection to the Arduino. This is done using Python. A wiring diagram is provided in Appendix 9.2.

The distance travelled per step of the motor was calculated as 0.21mm/step using the angle of a step, 1.8 °, and the diameter of the pulley, 6.92mm, measured using a vernier calliper. This may not be an accurate value due to errors in measuring the diameter of the pulley. The stepper motor has a step angle inaccuracy of ±5%, meaning that at any position an error of ±0.09° is possible. Assuming that no steps are skipped, this error will not increase over time. Assuming that no steps are skipped is a fair assumption, as the motor has a high holding torque (0.158Nm) [22] and will only be used to move light loads at slow speeds.

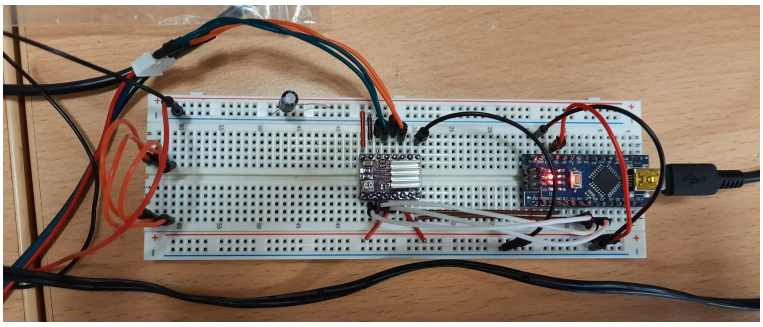

Figure 8: The Arduino circuit on a breadboard

3.4 Software

Acconeer provide a comprehensive Python library to interface with the sensors through a raspberry pi. This library has extensive documentation [23]. Custom software has been written to interface with this library to carry out experiments. This was also written in Python to make integration easier.

3.5 Sensor Test

To test that the sensors work as expected, a basic signal test is conducted. This consists of pointing the sensor at a reflective mirror. Metal is one of the strongest reflectors of radio waves [24] and so a sheet of metal will be used to act as a mirror. The experimental setup is shown in Figure 9. The mirror (3mm thick aluminium) is angled parallel to the sensor. The distance from sensor to mirror was measured to be 506mm. Assuming that the sheet metal acts as a mirror, the received signal should be a duplicate of the signal sent out. The data gathered from this experiment will test:

1. The sensors return a peak when a reflective object is present in the scene. This is a basic requirement for distance estimation and mapping.

2. The sensors detect an object at the expected range.

3. There are no strange artefacts in the reflected signal.

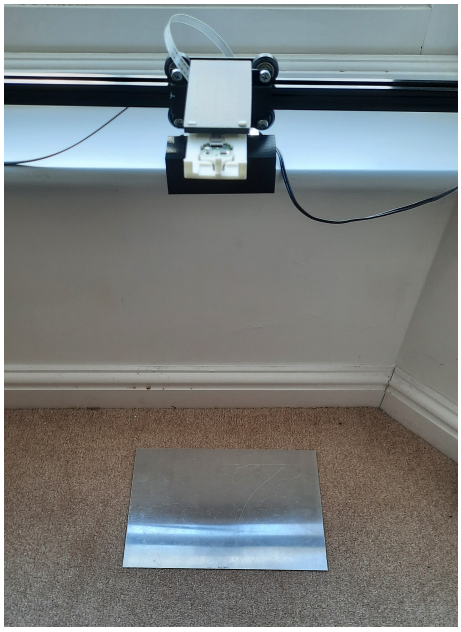

Figure 9: The experimental setup to test the sensors

3.6 Correlation Variation With Distance

A correlation against along-track distance plot is required before distance estimations can be developed. Data is gathered from a single sensor looking at a scene at different points in the along-track direction. A set of data taken at a nominal along-track distance can be chosen as a datum. Each other set of data can then be correlated with the datum. These normalised correlation values are then plotted against along-track displacement from the datum. This data can be used to model correlation as a function of distance, which can be used for distance estimation. The experimental setup for this experiment is shown in Figure 10

Figure 10: The experimental setup to gather data at different points in the along-track direction

For accurate distance estimation, it is required that the variation of correlation against along-track distance is well defined and follows a predicable pattern. A well-defined correlation variation should produce low values of correlation when datasets are dissimilar and high values of correlation when datasets are similar.

The way in which correlation varies with respect to along track distance may be dependent on several factors. These factors are tested to find the setup that produces the optimal correlation peak.

- Number of Samples. Instead of taking just one sample of data at a position, multiple samples can be taken and then averaged. Averaging more samples should increase the signal to noise ratio (SNR), which should make the correlation peak more well defined.
- Filtering of signals. The received signal may contain high frequency noise, especially data from the A111 sensors due to a high sampling frequency. These high frequencies can be filtered out to remove noise from the data.
- Length of transmitted signal. Profiles with shorter signal lengths theoretically have higher resolution. A higher resolution means that more information can be captured and subsequently correlated. This is beneficial. However, shorter signals also have lower power, which therefore decreases the signal to noise ratio. This can be somewhat compensated for by increasing the number of samples taken.

3.6.1 Scene

The quality of data gathered (and subsequent distance estimation) will be dependent on the scene that the sensors illuminate. The phase data of a reflected wave is unique to that reflector, acting as a fingerprint [9]. This characteristic is what makes distance estimation and SAR imaging possible. Therefore, this reflector should have a complex surface to give a unique phase pattern. This complex surface should have features greater than the wavelength of the signal (5.0mm) [10]. The quality of the phase data is dependent on the strength of the reflected signal. Steel wool was determined to be ideal for this application as it has both a complex surface and will produce a strong reflection.

3.6.2 Radar Footprint Geometry

The angle between the sensor and the surface of reflectors will impact the received data. Geometrically, the reflections from the surface are received at a delay proportional to the distance from the sensor to the reflector. Features will be better defined in the data if the reflection is spread over a longer range. High correlation between sets of data is dependant on identifiable features in the data. Therefore, the sensors should be positioned at a low downward-looking angle to give the best quality of data for correlation

An arbitrary downward looking angle of 30° was chosen. This gives a suitable balance between gathering lots of relevant data, but not having to populate the scene with large amounts of reflective surface. The vertical and horizontal geometry associated with this is shown in Figures 11 & 12 respectively.

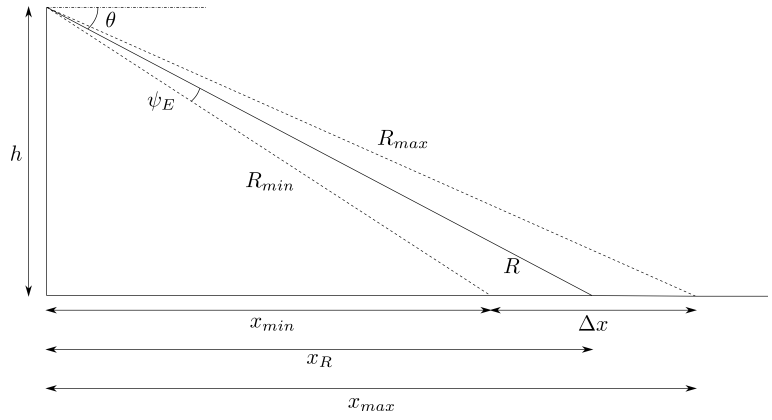

Figure 11: Vertical radar footprint geometry

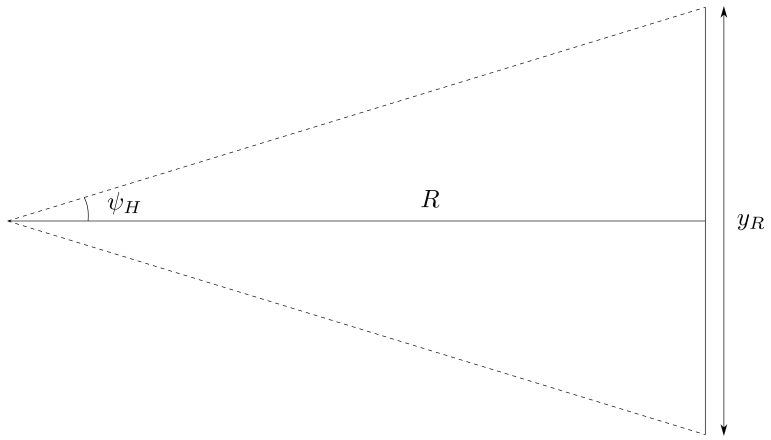

Figure 12: Horizontal radar footprint geometry

In this experiment, the value of h, θ, ψ are all known. All other variables can be expressed as functions of these parameters. Equations for all variables are given in Appendix 9.3. Equations 4, 5, and 6 describe the dimensions on the illuminated footprint. These equations can be used to confirm that the footprint of the radar will be entirely covered with steel wool.

$$x_{min} = \frac{h}{\tan(\theta + \psi_E)} \tag{4}$$

$$x_{max} = \frac{h}{\tan(\theta - \psi_E)} \tag{5}$$

$$y_R = 2 \cdot \frac{h}{\sin(\theta)} \cdot \tan(\psi_H) \tag{6}$$

3.7 Inter-Sensor Correlation

Different sensors collecting data in the same position and orientation should produce the same data. This is required for potential distance estimation. However, slight manufacturing differences and slight differences in position and orientation of the sensors may cause this to not be the case. Differences in position and orientation can be mitigated by using a square sensor mount that holds the sensors securely at all four contact points of the lens holder. However, an experiment should be done to determine whether the sensors do indeed produce similar data when positioned at the same location. This will be done in two different ways.

3.7.1 Direct Comparison

The extent to which different sensors do not correlate with each other can tested directly. Each sensor is positioned in a set position with a set scene. Theoretically, the data gathered by each sensor should be identical. Correlating sets of data collected by different sensors will give an indication of how well each sensor correlates with each other sensor.

3.7.2 Inter-Sensor Correlation Against Distance

As discussed in Section 3.6, variation of correlation with respect to distance will impact the quality of distance estimation. For this reason, the correlation variation between unlike sensors should be tested. This can be done by running experiment 3.6, but correlating sensors to a datum from a different sensor.

3.8 Distance Estimation

A distance estimation algorithm is developed and tested using simulated data. Initial testing on simulated data means issues with the algorithm can be identified before using potentially imperfect real-world sensor data. Once a suitable algorithm is developed, sensor data is input to this algorithm. The output displacement estimates are compared to the known displacements to determine the accuracy of the system.

3.9 Synthetic Aperture Radar (SAR)

The data gathering technique for SAR processing is identical to that for correlation variation against distance, with the additional requirements of minimum distance travelled (Section 2.7.1) and maximum separation of data points (Section 2.7.2).

4 Results and Analysis

4.1 Testing The Test-Rig

The test-rig uses open loop control. Therefore, for accurate positioning, the steps per millimetre needs to be known. This was estimated to be 0.21mm/step by the geometry of the pulley. The millimetre per steps was also worked out via experimentation. This was done by moving the carriage a large number of steps, measuring the distance travelled, and working out the mm/steps from that data. This was done 4 times to give the data in Table 4. The carriage started in a different position for each run.

Table 4: Recorded number of steps and distance travelled for the testrig

Number of Steps	Distance Travelled (mm)	mm/Step Estimate (mm)
3500	701	0.2003
2500	500	0.2000
3600	719	0.1997
3400	680	0.2000

The results in Table 4 suggest that the test-rig has 0.2mm/steps. This result is consistent across all runs to an accuracy of 3 decimal places. This new value is more accurate than the value calculated in Section 3.3 as it calculates the mm/steps over a large number of steps, which distributes the absolute error from measuring distance over thousands of steps, reducing the percentage uncertainty in the measurement. Therefore, this value is used in future calculations. The stepper motor may have a $\pm 0.09^\circ$ error at any time [22], which results in a potential ± 0.01mm linear error.

4.2 Sensor Test

The amplitude of the received signals from the experiment described in Section 3.5 are shown in Figure 13. Note that a low pass filter has been applied to the A111 data as the raw signal has significant noise, which would make the plot unclear. Both sets of data contain a high peak at a distance of approximately 500mm, corresponding to the known distance of 506mm. However, both sensors also contain a smaller peak in amplitude at approximately 1000mm. It is hypothesised that this signal is a result of a *double bounce*, where the signal returning from the mirror is reflected off the sensor, and so travels between the sensor and mirror more than once. If this is the case, multiple peaks would be expected at integer multiples of the separation between sensor and mirror (500mm, 1000mm, 1500mm, 2000mm, etc). To test this hypothesis, data with a longer range is plotted in Figure 14. A logarithmic y axis makes it easier to identify the small peaks in amplitude. In this plot, peaks are clearly seen at 500mm, 1000mm, 1500mm, and 2000mm. This corroborates the hypothesis. Therefore, the peaks at these values are not caused by the sensors.

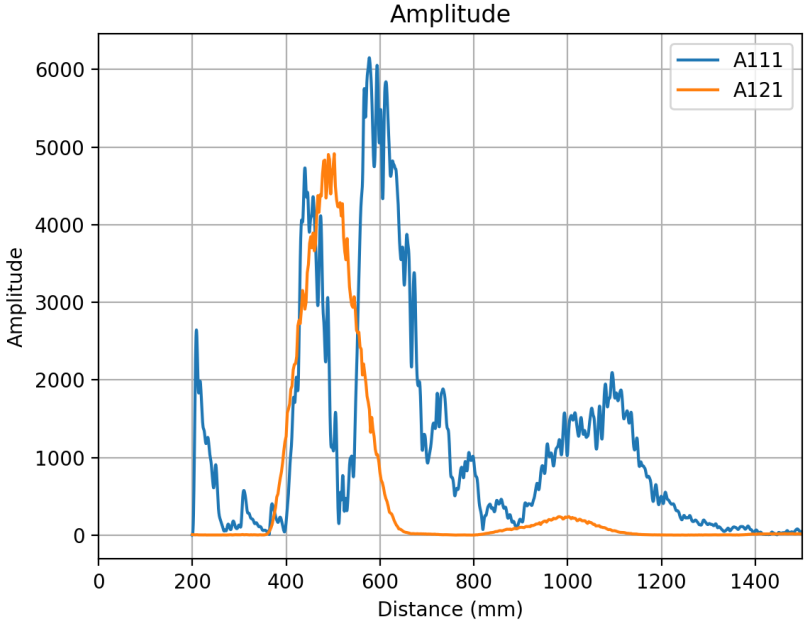

Figure 13: Signals received from both sensor types when facing a reflective surface

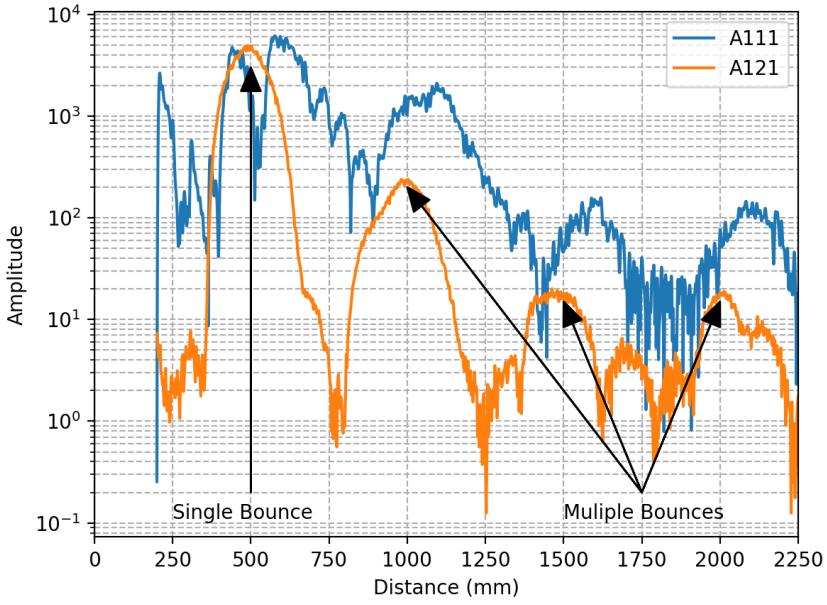

Figure 14: Logarithmic plot of signals received from both sensors when facing a reflective surface

Figure 13 shows that the A111 sensors detect two peaks in amplitude where the reflector is. These peaks are well-defined as they have a significant trough in-between them. It is therefore unlikely that this artefact is caused by noise. After speaking with representatives from Acconeer, it was suggested that these artefacts are caused by the inbuilt averaging step within the bare metal of the module. It was further suggested that this issue can be mitigated by reducing the inbuilt averaging. This was tested by collecting multiple sets of data with differing amounts of inbuilt averaging. In each case, additional samples were taken manually

to ensure that each dataset is an average of the same number of samples. Figure 16 shows it is clear that reducing inbuilt averaging does indeed reduce these artefacts of the A111 sensors. This was an important discovery as these artefacts significantly degrade the quality of any gathered data, which would negatively affect applications in navigation. For completeness, the same experiment was run with the A121 sensors. Figure 15 shows that differing inbuilt averaging makes no significant difference for the A121 sensors.

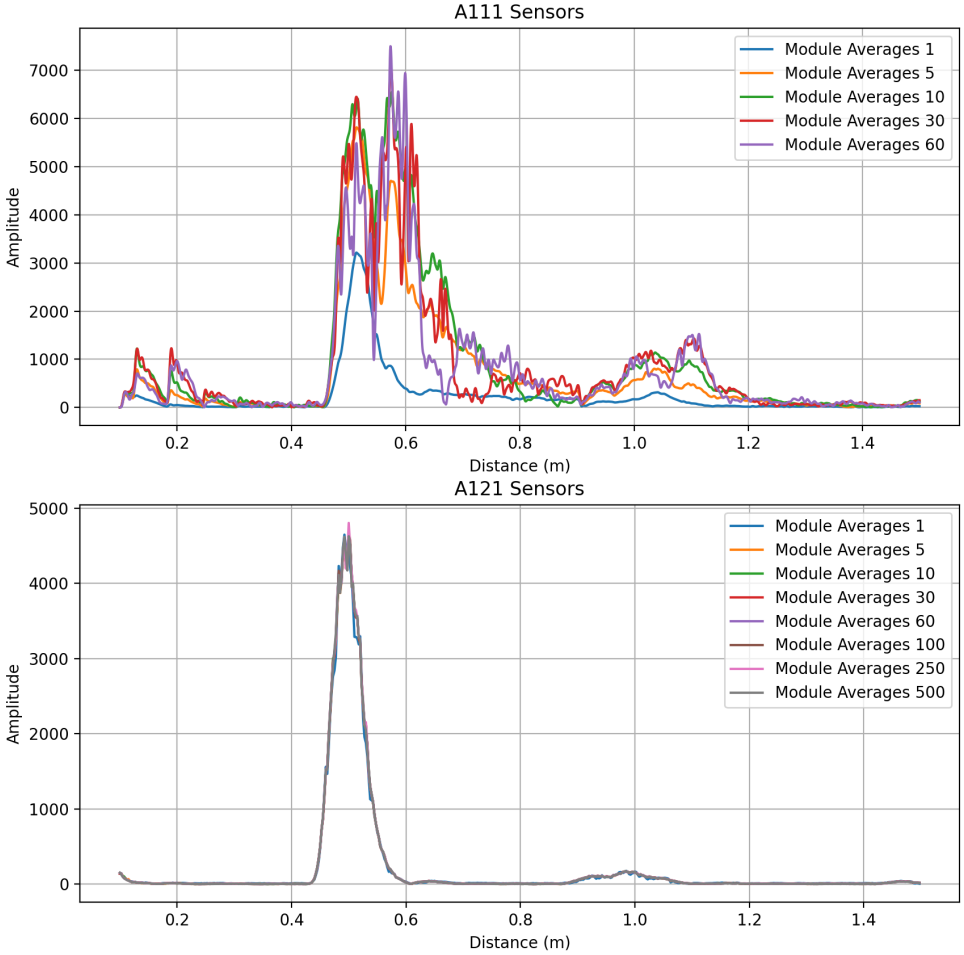

Figure 15: A comparison of inbuilt averaging settings

A frequency analysis of the signals is shown in Figure 16. This was done using the numpy fft method. Figure 16 shows that the A111 sensors have significant spikes in amplitude of high-frequency noise at 154 GHz and 247 GHz. This is consistent with the raw A111 data, which has significant high-frequency noise. As previously mentioned a sampling frequency over 128 GHz is not required, so this provides a sensible cut-off frequency for a low-pass filter.

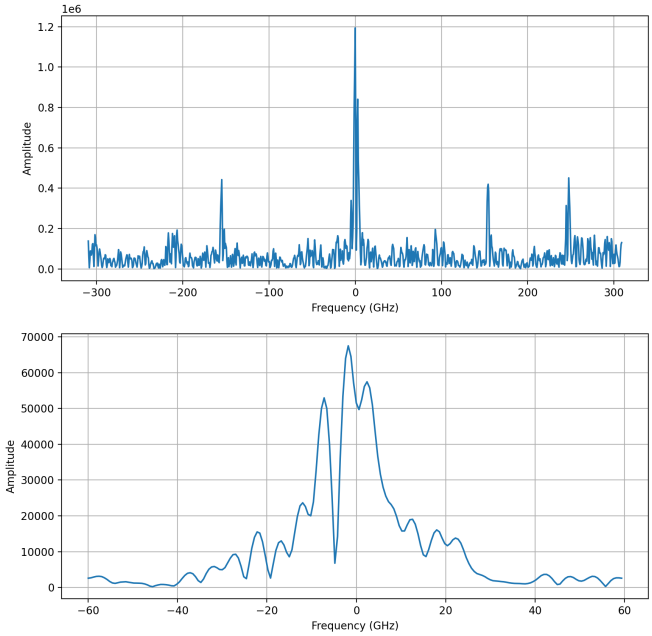

Figure 16: The spectrum of frequencies in the received signals

4.3 Expected Correlation Against Distance Plot

The theoretical variation of correlation with respect to distance can be calculated using well known methods [13]. This was done using MATLAB code provided by Dr Ben Thomas. The correlation profile is dependent on the signal frequency, bandwith, wavelength, and half power beamwidth. The only parameter that can be changed on these sensors is the beamwidth (by adding or removing lenses). Therefore, a series of theoretical correlation profiles with different beamwidths are calculated and shown in Figure 17. Beamwidths that can be achieved with the sensors (and lenses) are plotted as solid lines.

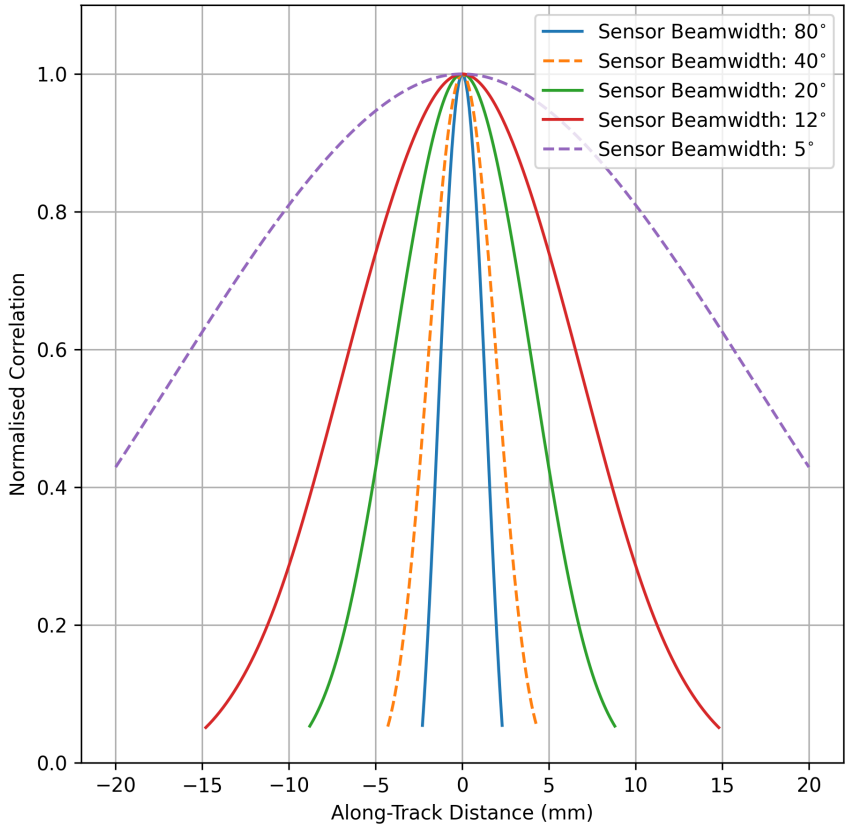

Figure 17: Theoretical correlation profiles for possible sensor beamwidths

Figure 17 shows that narrower beams give wider correlation profiles. Wider correlation profiles are desirable for the distance estimation method. Therefore, it is desirable for half power beamwidth to be as small as possible. The narrowest beamwidth achievable is 12° using a hyperbolic lens in position D2 (Table 2). Therefore, this setup will be used to gather data for the real correlation profile.

4.4 Real Correlation Against Distance Plot

Figure 18 shows the real correlation profiles produced by the sensors. A data sample in the middle of the dataset has been designated as the datum. All other samples have been cross-correlated with this datum. In Figure 18, the datum is at a displacement of 0. By definition, a correlation value of 1 is obtained at this point, as the data sample at 0 displacement is the same as the datum. The A111 correlation peak is much shorter than the theoretical peak; actual correlations are much lower than expected. Furthermore, neither correlation profile is ever as low as the theoretical correlation profile. This is likely because correlation is upward-biased when noise is present in the signal [25].

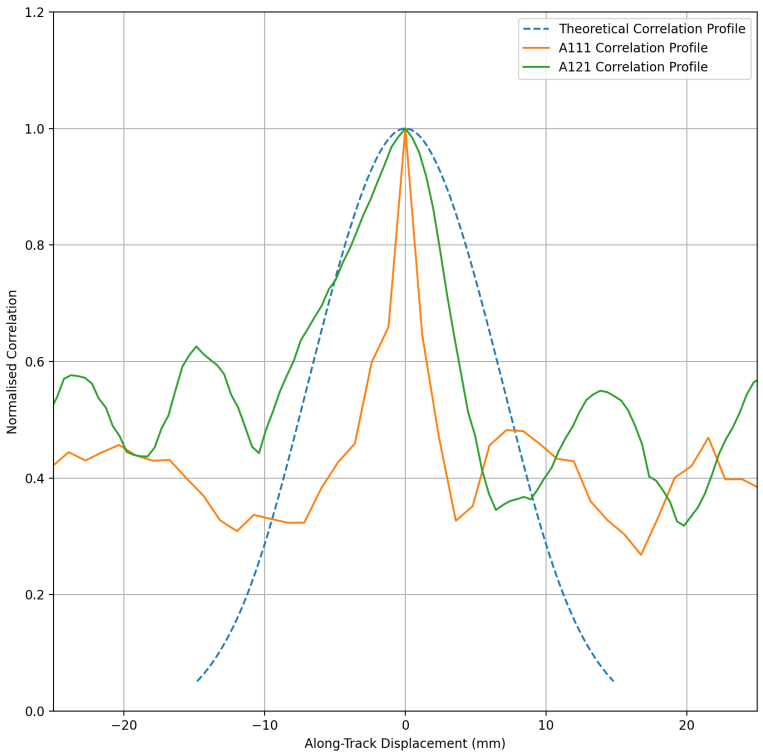

Figure 18: Correlation profiles of the sensors

A comparison of the number of samples averaged is shown in Figure 19. It can be seen that more averaging improves the A111 correlation profile - making it more similar to the theoretical profile. This is likely because increasing the number of samples increases the SNR. However, averaging has little impact on the A121 sensors, suggesting that the SNR of these sensors is already high.

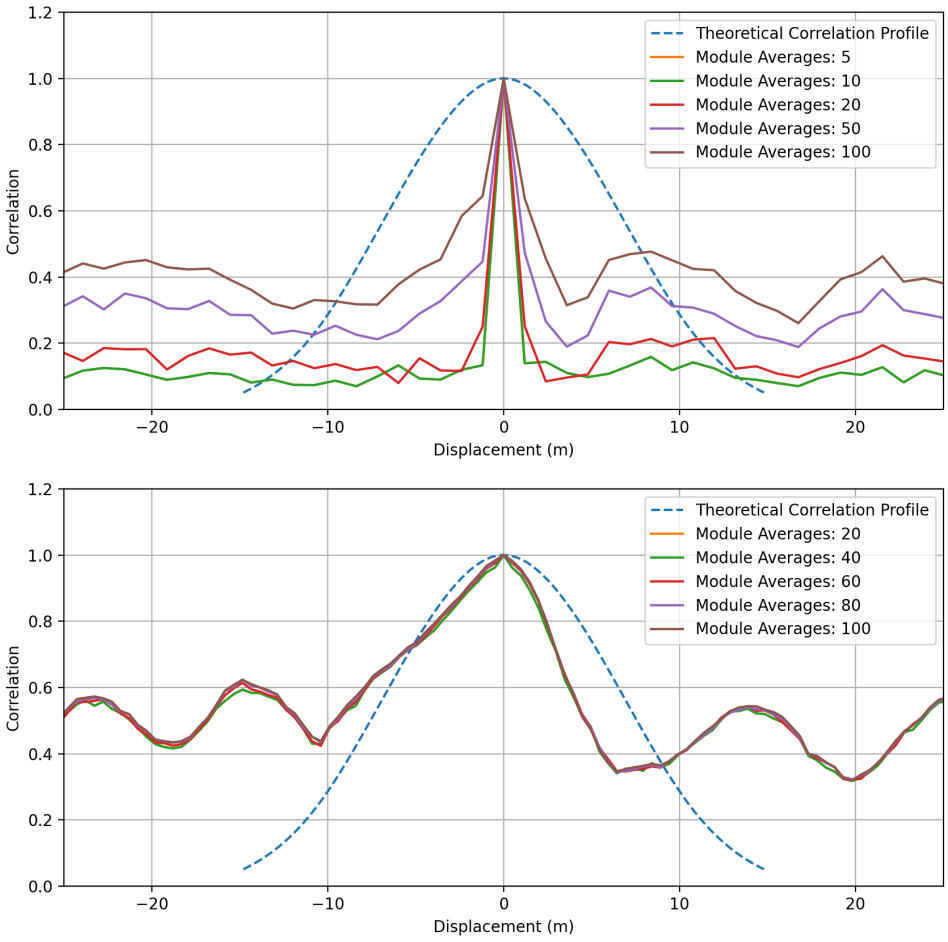

Figure 19: Correlation profiles comparison by number of samples. Signal 2 was used. No filters were applied.

A comparison of the transmitted signal type is shown in Figure 20. The most noticeable effect of using longer signals is that the base correlation value is raised. This is as expected; a longer signal means the data is lower resolution, and thus has lower detail, which makes each signal look more similar. For the A111 sensors, the correlation near the peak is significantly increased by increasing the signal length. However, the floor of the peak is also raised. Therefore, it could be argued that the quality of correlation between different signal lengths is negligible for this reason. The peak of the A121 sensors is similar for both signals. However, similar to the A111 sensors, a longer signal means a higher base correlation value, which results in a less well defined correlation peak. For the A121 sensors, a longer signal gives a worse profile. This is consistent with the hypothesis that longer signals produce a worse correlation peak.

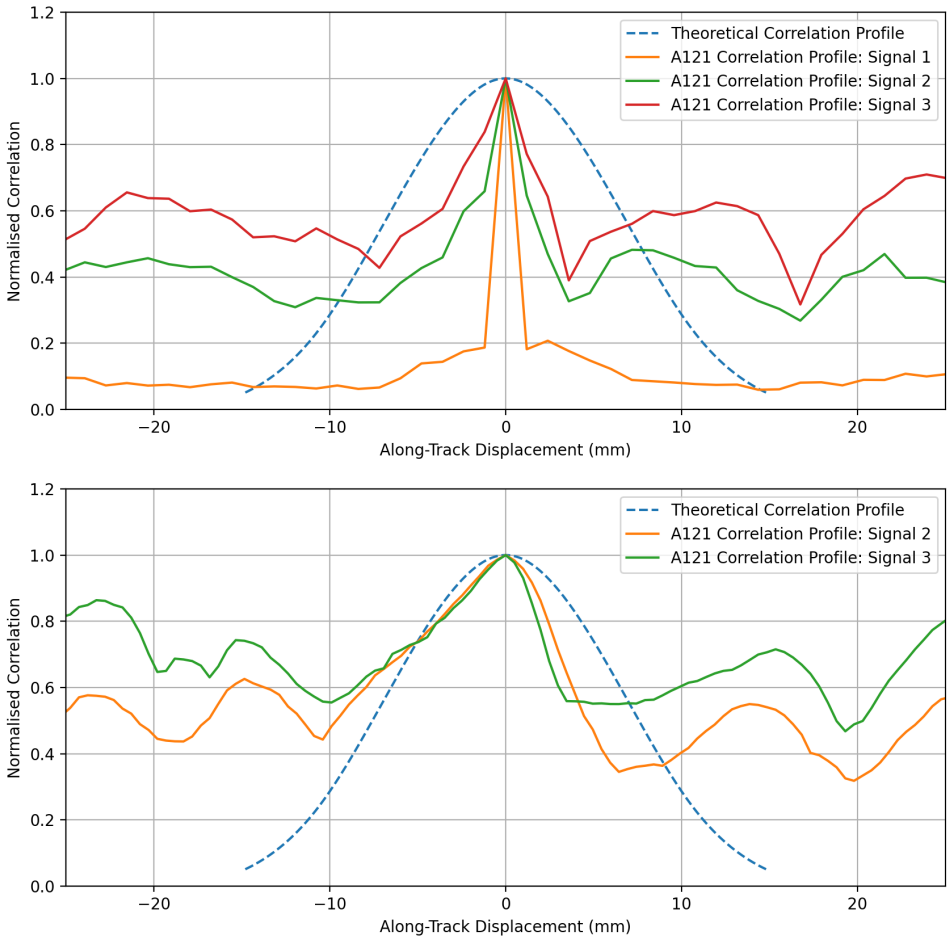

Figure 20: Correlation profiles comparison by transmitted signal. No filters have been applied

Figure 21 shows the impact of different low-pass filter frequencies. A Butterworth low-pass filter was utilised to remove high frequencies [26]. It can be seen that applying a filter to the A111 data raises the value of correlation. Similar to Figure 19, it can be argued that quality of correlation between different filtering values is negligible as the value of correlation is raised everywhere - including the base correlation value.

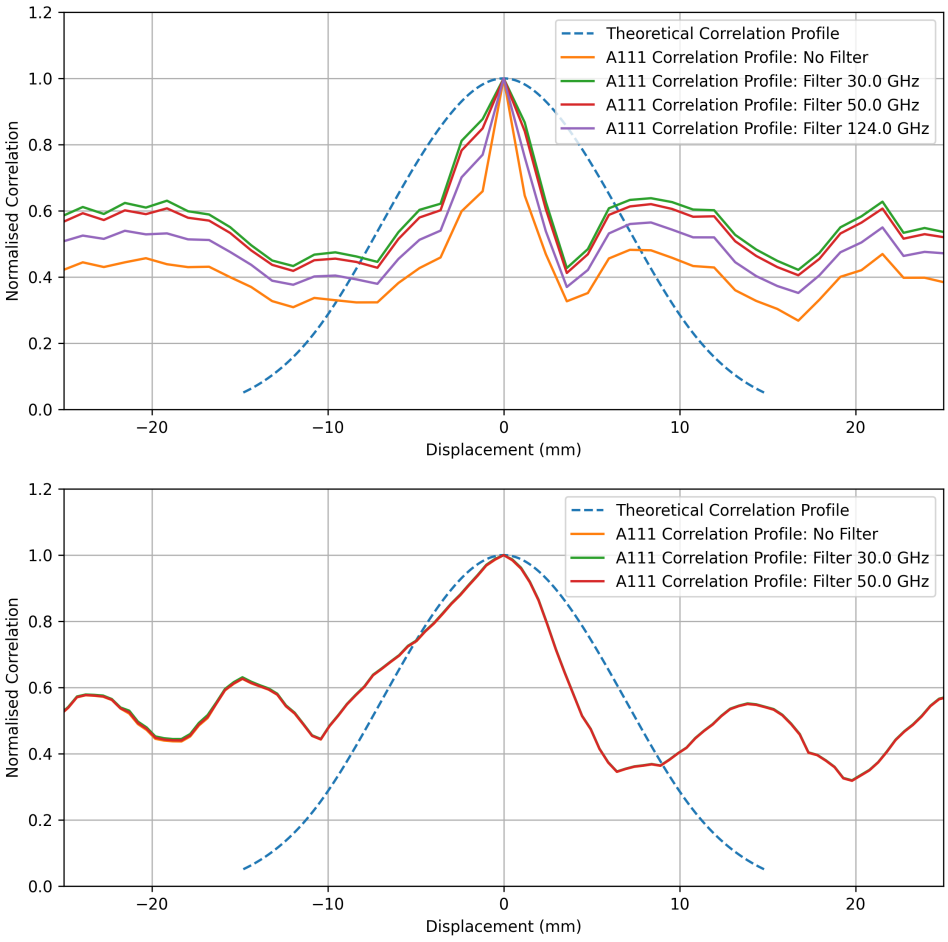

Figure 21: Correlation profiles comparison by filtering

4.5 Problem With The Correlation peak

Originally it was hoped that a distance estimation algorithm used in SAS could be used with these sensors. This algorithm works by placing (at least) 3 sensors in an array at separations such that each sensor correlates highly with a set of data gathered at a previous point in time. This arrangement in shown in Figure 22. The data from sensor 2 at a previous sample is chosen as the datum. The current data from all sensors is correlated with this datum, producing the correlation values represented as black crosses. These 3 correlation values can then be used to estimate the shape of the correlation profile (the red curve), and thus estimate distance travelled by considering how far the peak of the profile has shifted [13]. In order for this method to work, sensors must produce high correlation values with the datum.

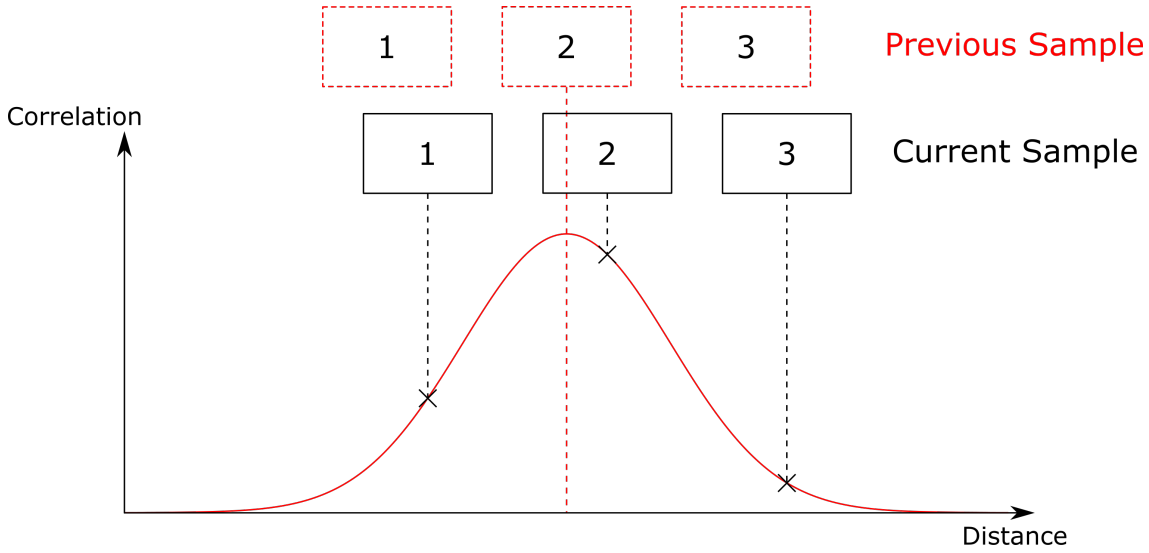

Figure 22: A distance estimation method for a wide correlation peak representing a best-case scenario

However, this method is not possible with the real sensors as the correlation profile is too narrow, as shown in Figure 23. This was discovered when calculating the theoretical correlation peak. The maximum possible width of the theoretical correlation peak is approximately 30mm. However, the width of the modules in the lens holders is approximately 48mm. Therefore the absolute minimum separation that could be achieved between modules is 48mm. This is wider than the correlation peak and so it is not even possible to simultaneously have 2 sensors correlate highly with the same data point.

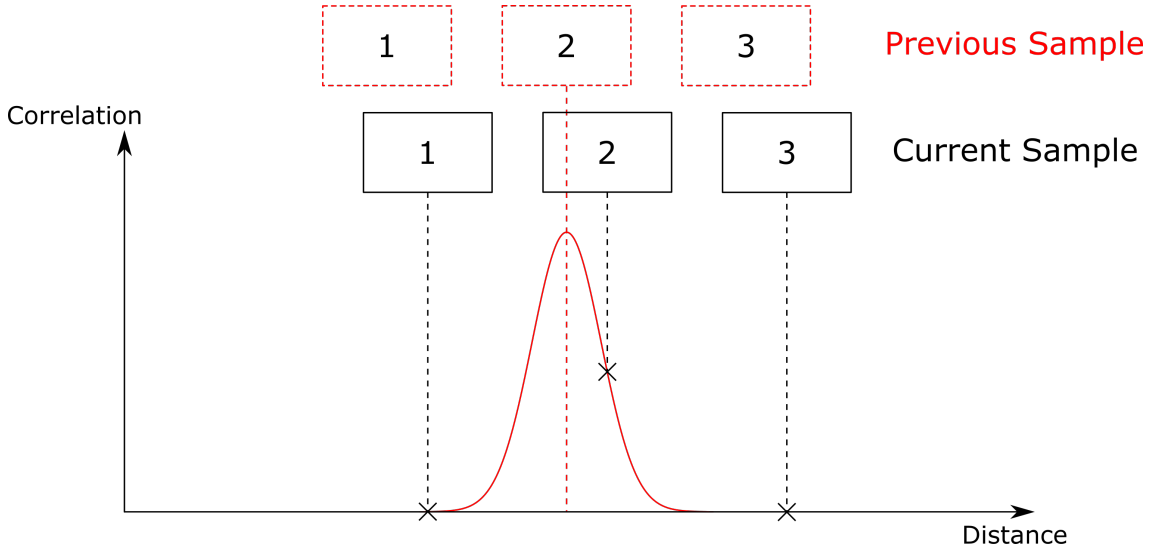

Figure 23: A distance estimation method for a narrow correlation peak representitve of the measured data

This issue can be overcome by either

1. Increasing the width of the correlation profile. It has been shown that narrowing the beamwidth increases the width of the correlation peak. The beamwidth can be narrowed with lenses. The beamwidth in the experimental setup has already been reduced from 80° to 12°. It is unlikely that the beam can be narrowed significantly more than this. Narrowing the beam further would require designing and manufacturing a new lens.

2. Moving the sensors closer together. The sensors themselves are only 5mm wide. The modules are 20mm wide and the lens holders are 48mm wide. It is therefore possible to redesign a PCB whereby multiple sensors are housed on a single PCB. This would allow a minimum of 5mm spacings. The lenses would also need to be redesigned. This would require significant time, but is feasible. Unfortunately, this was not practical to achieve within the time frame of this project.

4.6 Inter-Sensor Correlation

4.6.1 A111

Table 5 shows how the A111 sensors correlate with each other. Each sensor was looking at the exact same scene. This was done by moving the array to align each sensor with the scene individually. For clarity, each cell is coloured proportionally to the correlation value - a value of 1.0 is green and a value of 0.5 is red. A low pass filter of 128 GHz was applied to all data. The leading diagonal is populated by maximum values of 1 as data is correlated with itself at these points. From Table 5 it can be deduced that the sensors do not correlate well with each other. There are two likely causes of this:

1. Misalignment of the sensors could cause them to correlate poorly. This is because misaligned sensors will return slightly different data. Differences between data will result in lower correlations.

2. Slight manufacturing differences in the sensors will result in different data being returned. For example, the sensors may have different beamwidths, causing them to return data from different parts of the scene.

Table 5: Correlation between A111 sensors

-	Sensor 1	Sensor 2	Sensor 3	Sensor 4
Sensor 1	1.000	0.7016	0.643	0.7083
Sensor 2	-	1.000	0.7859	0.8669
Sensor 3	-	-	1.000	0.736
Sensor 4	-	-	-	1.000

4.6.2 A121

The results of the same experiment for the A121 sensors are shown in Table 6. It can be seen that A121 sensors correlate much more strongly with each other than A111 sensors. Because the same array mount was used in both experiments, it can be deduced that improvements have been made to the hardware of the sensors to ensure they correlate highly with each other. The remaining difference in correlation between sensors is still likely due to a combination of small manufacturing differences and alignment errors.

Table 6: Correlation between A121 sensors

-	Sensor 1	Sensor 2	Sensor 3
Sensor 1	1.000	0.9408	0.8697
Sensor 2	-	1.000	0.8859
Sensor 3	-	-	1.000

4.7 Along-Track Inter-Sensor Correlation

The previous experiment has quantified how well different sensors correlate with each other. How correlation between sensors varies with respect to along-track distance is also important for distance estimation. Figure 24 shows how correlation varies between sensors. This plot was created by selecting a set of data from the right-most sensor as a datum. All other data points from all other sensors are then plotted. The vertical dashed lines represent displacement where the datum overlaps with the position of a sensor; a peak in correlation is therefore expected at each line. For the A111 sensors, two correlation peaks appear to be missing. These correspond to sensor 4-3 and 4-1 correlations, which were the inter-sensor correlations with low values in Table 5.

As expected, the A121 sensors correlate much better with each other. Significant correlation peaks appear where they should, although the peaks appear to reduce with respect to distance from the datum.

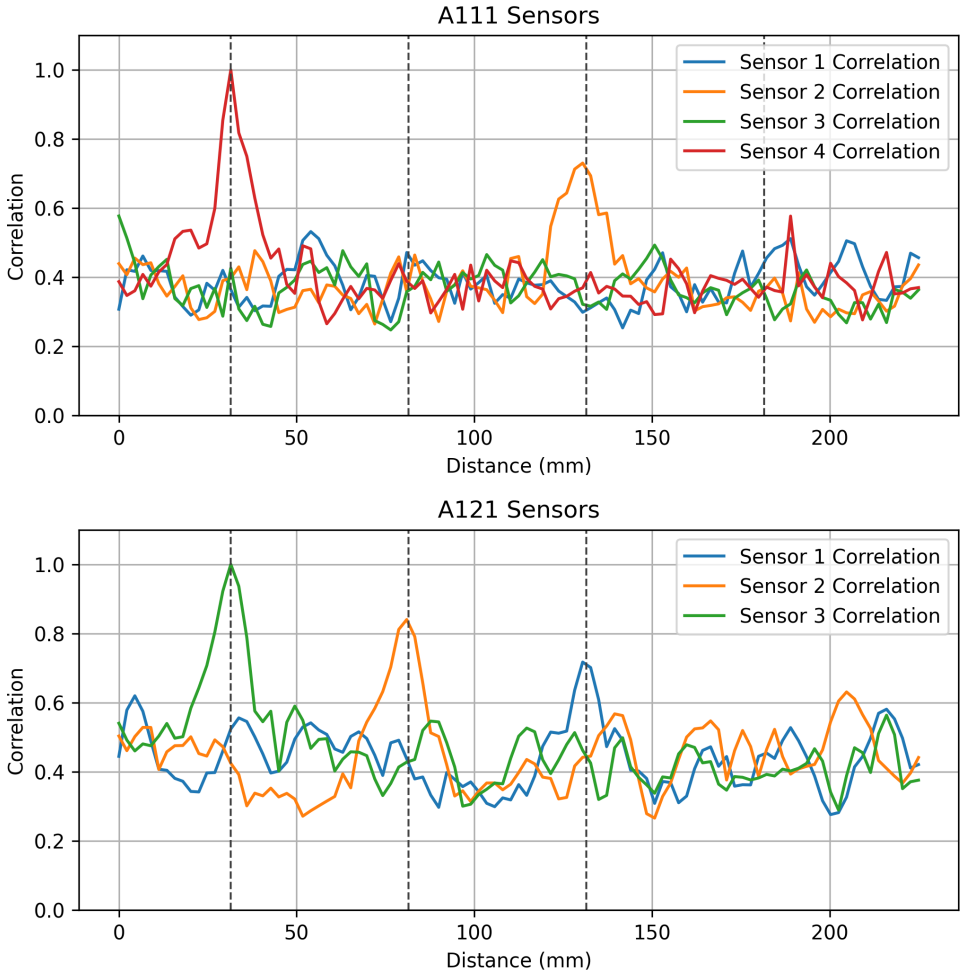

Figure 24: Variation of correlation between sensors against distance

4.8 Applying Distance Estimation to Simulation

In this section, a distance estimation algorithm is developed. To ensure that this algorithm can be used on real-world data, three different domains are considered: sample number, time, and distance. Assuming that

the sampling frequency of the sensor is constant, then time and samples are simply scaled versions of one another. Distance is a function of time, although the shape of this function is not known. Simulations will take place in the time domain, whereas real-world data will take place in the samples domain.

4.8.1 Modelling The Correlation Profile

The theoretical correlation against distance profile can be calculated using code provided by Dr Ben Thomas. This gives an accurate profile. However, it is slow (as it uses an iterative approach), and it is not possible to find the inverse of the correlation profile using this method. Therefore, a curve will be fitted to the output data of this code to model the profile. This is a common approach for distance estimation methods. The inverse of this curve can then be used for simulation purposes. Three functions are applicable to the correlation profile.

1. Quadratic Approximation - Equation 7.

$$\mu\left(x\right) = ax^2 + bx + c \tag{7}$$

   Modelling the correlation profile as a quadratic presents a challenge as the gradient of a quadratic function must increase/decrease at a constant rate. It therefore is not able to model the features of the theoretical profile as it has multiple points of inflection. Therefore, in order to produce a somewhat accurate fit, the quadratic approximation is restricted to only fit the curve for points where the correlation exceeds 0.3.

   The inverse of this function can be found from the quadratic formula (Equation 8).

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{8}$$

2. Gaussian Approximation - Equation 9.

$$\mu\left(x\right) = A \cdot e^{-\frac{1}{2} \cdot \frac{(x-\zeta)^2}{\sigma^2}} \tag{9}$$

   In Equation 9, ζ determines the x offset, σ determines the width of the curve, A determines the height. The inverse of a Gaussian is given by Equation 10.

$$x = \zeta \pm \sqrt{2\sigma^2 \ln\left(\frac{A}{y}\right)} \tag{10}$$

3. Cubic approximation - Equation 11. A cubic approximation exists as a direct result of the method to calculate the theoretical correlation profile (van Cittert-Zernike theorem) [27]. Note that although this function is a cubic, it only has three parameters. This makes it possible to fit this curve with only 3 known points.

$$\begin{aligned} \mu_{12}\left(x\right) = \mu_0 \big(& -2\left|x - 2\kappa L_{RX}\right|^3 + \left|x + \left(2 - 2\kappa\right) L_{RX}\right|^3 + \left|x + \left(-2 - 2\kappa\right) L_{RX}\right|^3 \\ & -2\left|x + 2L_{RX}\right|^3 + 4\left|x\right|^3 - 2\left|x - 2L_{RX}\right|^3 + \left|x + \left(2 + 2\kappa\right) L_{RX}\right|^3 \\ & + \left|x + \left(-2 + 2\kappa\right) L_{RX}\right|^3 - 2\left|x + 2\kappa L_{RX}\right|^3 \big) \end{aligned} \tag{11}$$

   The inverse of this function is not known. It may be possible to find the inverse by algebra, but it is impractical. Therefore, some form of iterative computational approach would be required. Also, unlike the Gaussian or quadratic, it is not clear what effects changing the inputs gives to the cubic approximation.

4.8.2 Fitting a Correlation Profile

Figure 25 shows that the quadratic approximation is the least accurate and the cubic approximation is the most accurate. The quadratic approximation will not be used due to the high errors. While the cubic approximation is the most accurate, the inverse is not known and finding it is impractical. Therefore, the Gaussian approximation will be used as it has the best balance between accuracy and ease of use. The associated inaccuracy will be negligible compared to the imperfect shape of the real correlation profile.

The parameter values for each approximation are also given in Figure 25. The curve fit procedure (using the scipy library optimize function) has estimated the Gaussian offset, ζ, as being non-zero. This is unexpected as the theoretical profile variation is centred about x = 0. The value of ζ is very small however, so this is likely an outcome of the curve fit procedure. When using this Gaussian approximation, zeta will be assumed to be 0.

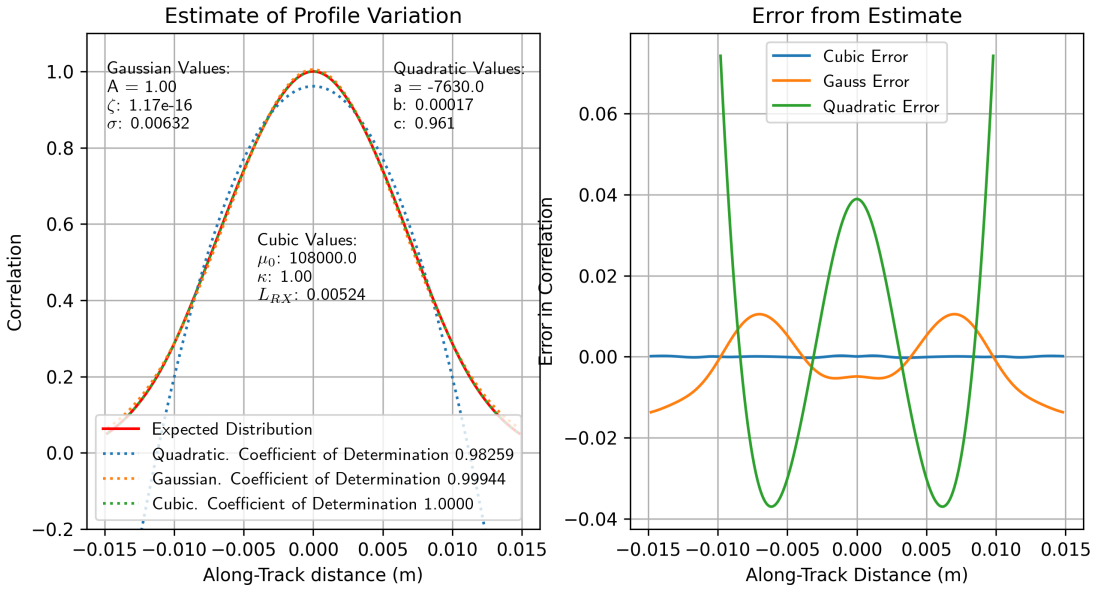

Figure 25: Three curve fits of approximations of correlation variation

4.8.3 Estimating Distance Using Two Sensors

For this simulation a single correlation peak is shown centred at a distance of 50mm. The correlation peak has been calculated using the Gaussian with values defined in the previous section. This profile is analogous to data from a single sensor being correlated with data 50mm ahead of the start position. This may occur in an array of 2 sensors separated by 50mm. Data from the first sensor is selected as the datum. This simulation shows what may be seen if data from the second sensor is correlated with the datum as it moves in the direction of the first sensor. The results are shown in Figure 26. Distance can be estimated using the correlation value as an input to the inverse Gaussian function. This has been done for each correlation value on the correlation profile. The resulting distance estimates are shown in red. As indicated by the scatter plot, the inverse Gaussian always produces two different distance estimates. If no other information is available, it is not possible to distinguish which estimate is correct. As a consequence it is not possible to determine the direction of movement. However, there is still value in this information. If this process is combined with another sensing method such as odometry, it will be possible to determine which distance estimate value is correct with relative ease.

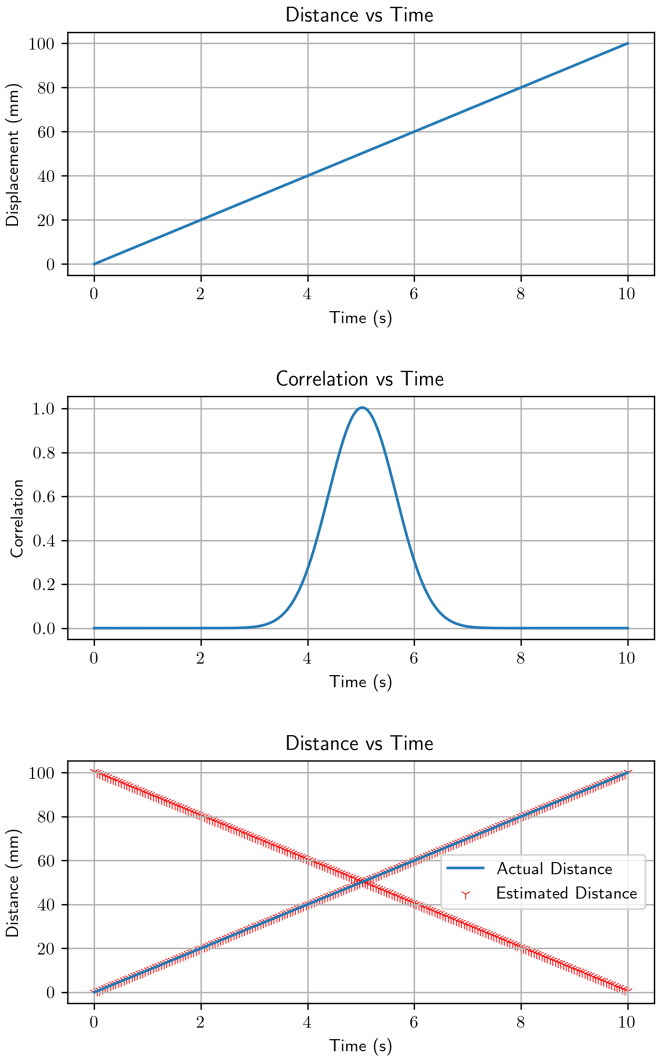

Figure 26: Simulated distance and coherence against time plot with estimates of distance using the inverse Gaussian function

4.8.4 Real Correlation Profile Imperfections

Due to real world conditions, there are 3 major effects on the correlation profile. To investigate the impact of these effects, each was simulated individually and any effects were observed. Figures showing individual effects are in Appendix 9.4.

1. Noise. This is caused by random noise in the radar signals, which corresponds to random noise in the correlation profile. It has been shown that this noise decreases as the number of averaged samples increases (Figure 19), but it is impractical to take very large numbers of samples, so this is an issue that may be present in real-world applications. Figure 41 shows that noise simply decreases the accuracy of a distance estimate. This effect is more significant at low correlation values where the magnitude of the noise is large with respect to the magnitude of the correlation profile.

2. Reduction in peak height. Because of poor correlation between sensors, the correlation peaks will have a lower maximum height than 1. Figure 42 shows that a reduced peak gives inaccurate distance estimates near the peak. This makes sense as the absolute error between the ideal and reduced peak profiles is maximum at the peak of the profile.

3. Upwards-bias floor. Because correlation is an upwards-bias estimator, there will be a base correlation value that no sample will be below. ie: correlation will never decrease to 0, but to a particular value where there will only be noise. Figure 43 shows that raising the floor of the correlation profile makes distance estimates inaccurate at low correlation values.

Figure 27 shows all effects at once. All effects described above can be seen in this figure.

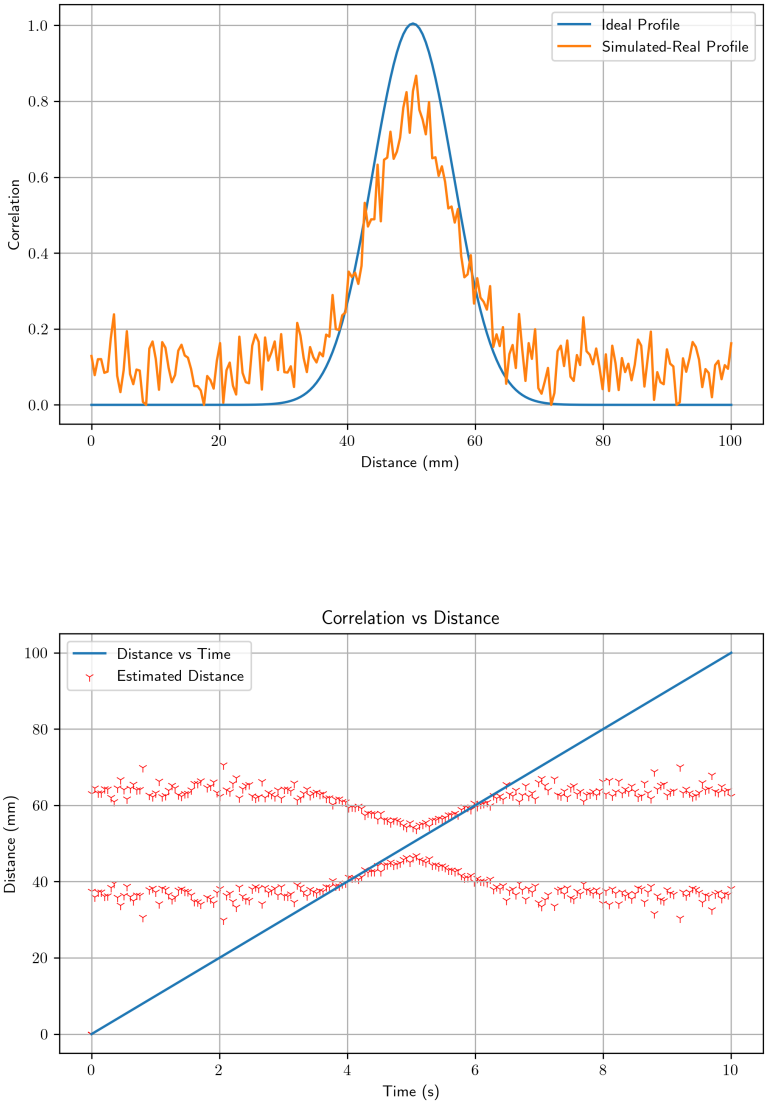

Figure 27: Simulated correlation profile with all negative effects

4.8.5 Compensating Real Correlation Profile Imperfections

It has been seen that imperfections in real-world correlation peaks have negative effects on any distance estimates. The negative impact of a lower peak can be mitigated by refitting a Gaussian curve to the real data. Refitting the curve means accurate estimates can be found when near the peak of the profile. Another improvement is to not estimate distance for low correlation values. The raised floor and noise both have significant negative impacts on distance estimates at low correlation values. Therefore, it is sensible to restrict the correlation values for which to estimate distance. There is no way to compensate for noise in the correlation peak, due to the random nature of noise. Figure 28, shows that compensating for the irregular correlation peak results in a estimates that more closely match the actual distance travelled.

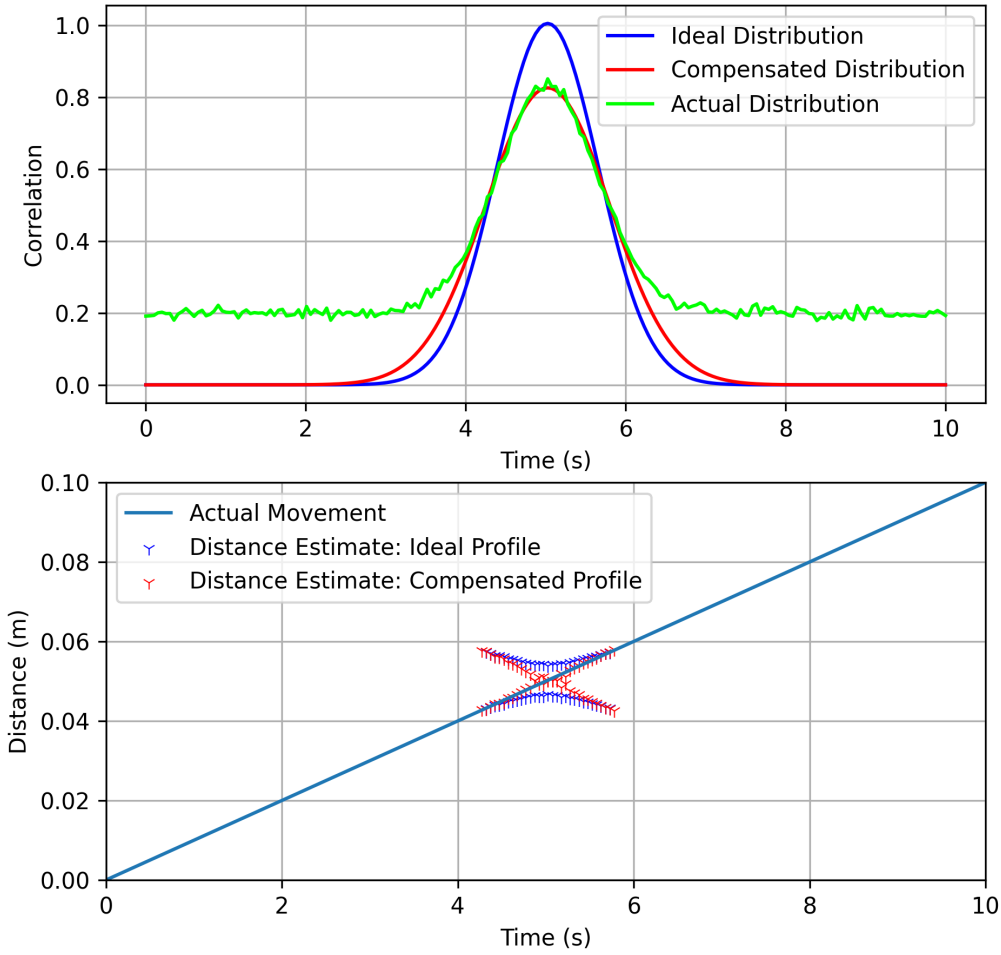

Figure 28: Comparison of estimating distance using compensated and non-compensated inverse-Gaussian functions

4.8.6 Unambiguous Distance Estimation Using 3 Sensors

When estimating the distance travelled by using the inverse-gauss function, two distance estimates are always returned. It has been noted that additional sensing methods such as odometry could be used to reject the incorrect estimate. However, it is also possible to reject false estimates by using an additional

radar sensor.

Firstly, consider a 3 sensor array with sensors at separations of 0mm, 50mm, 100mm. Figure 29 shows the two correlation peaks from correlating each sensor with the one in front of it. The distance estimates on the bottom left are based on the ideal correlation profile, whereas those on the bottom right are based on a simulated imperfect correlation profile and a compensated Gaussian. The ideal correlation distance estimates overlap exactly, whereas the real distance estimates do not overlap exactly due to the random noise. Having two correlation peaks to estimate distance still results in two distinct estimates for each point.

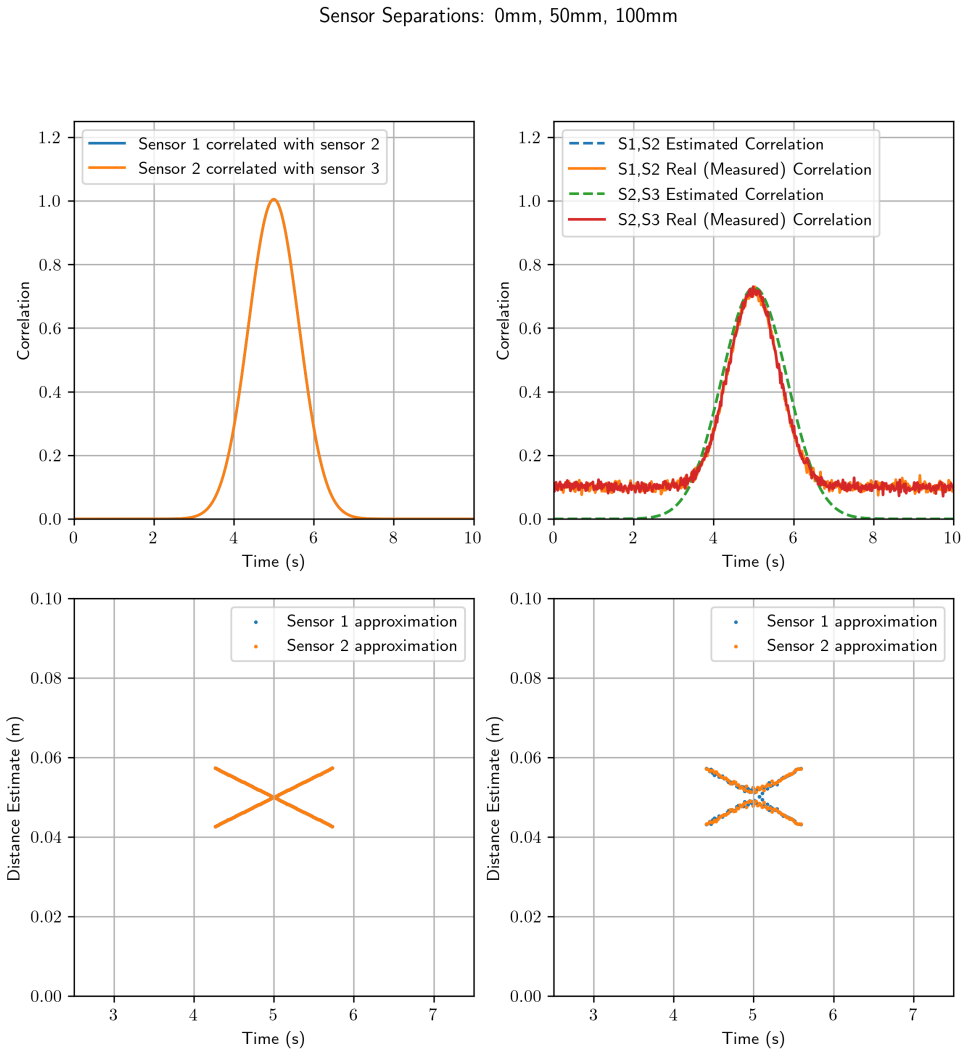

Figure 29: Three sensor array with uniform offsets inter-sensor correlation simulation

It is possible to get a single distance estimate with no ambiguity by offsetting the spacing of the sensors. Figure 30 shows the same simulation as previously, but with sensors at separations of 0mm, 50mm, 105mm. Now, instead of the inter-sensor correlations being completely aligned, they are slightly offset by 5mm. This means that by considering estimates from both correlation peaks, basic logic can be used to reject false values and find an exact position.

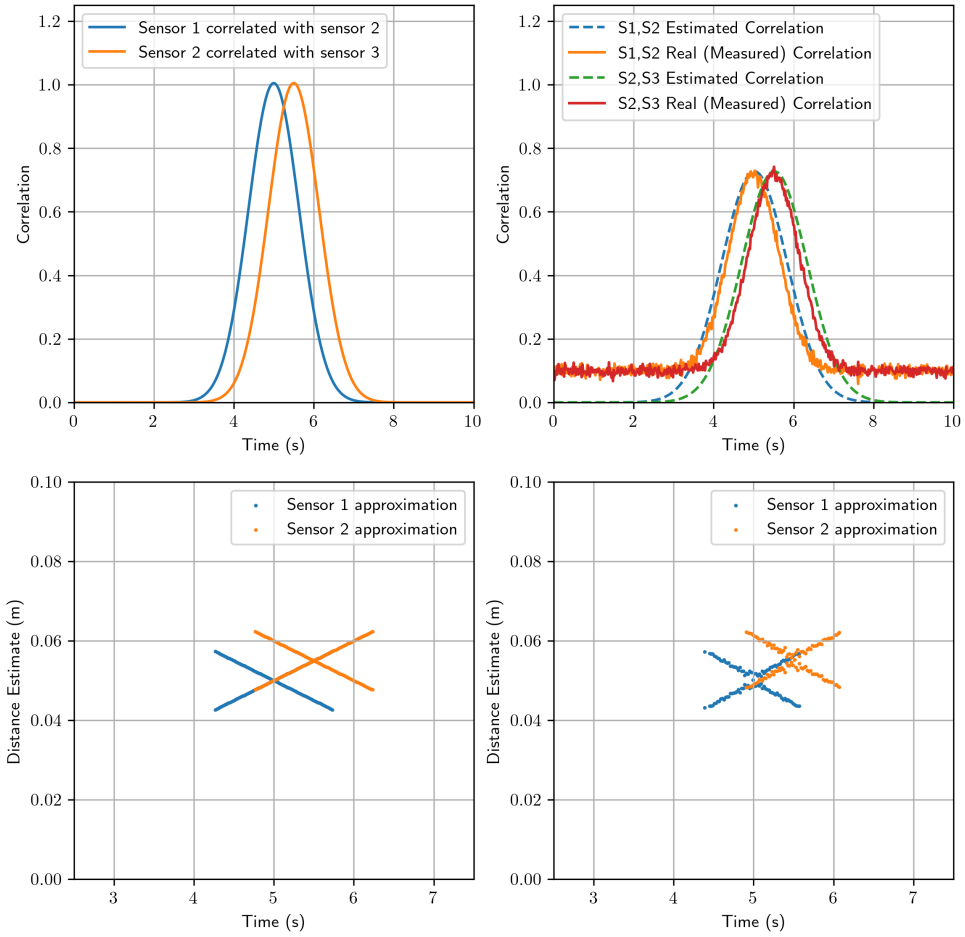

Figure 30: Three sensor array with non-uniform offsets inter-sensor correlation simulation

In a 3 sensor array, when examining the correlations of each sensor with the sensor in front of it, a maximum of four distance estimates will be found. This can be seen by inspecting the distance estimates from $5s < t < 5.5s$ in Figure 30. Two of these estimates will be similar, and the other two will be dissimilar. For example, in an ideal case, these four values might be estimated:

$$\left[\begin{bmatrix}45 & 55\end{bmatrix}, \begin{bmatrix}55 & 65\end{bmatrix}\right]$$

In this case it is clear from inspection that the array is at a 55mm displacement. However, when a noisy signal is analysed the distance estimate may be:

$$\left[\begin{bmatrix}43.7 & 59.6\end{bmatrix}, \begin{bmatrix}54.2 & 67.2\end{bmatrix}\right]$$

The outlier measurements convey no useful information. By examining the estimates, the outliers can be discarded. The two similar estimates can be averaged to produce a single, unambiguous estimate. This method works by placing two sensors on different correlation peaks, but approximating the peaks as having the same shape. This creates a virtual correlation peak, and it is possible to place as many sensors as desired on this peak.

4.8.7 Full Distance Estimation

Recall that correlation profiles are always associated with a datum position. When navigation starts, the datum position is selected as being the first sample with a displacement of zero. It has been shown that an array of 3 sensors with non-uniform offsets can be used to obtain multiple, unambiguous distance estimates. These distance estimates can only occur when the array has moved approximately 50mm from the initial starting position. It has been shown that each datum can lead to multiple distance estimates. This process can therefore be repeated until a distance estimate is available for each new set of data. This concept is shown in Figure 31.

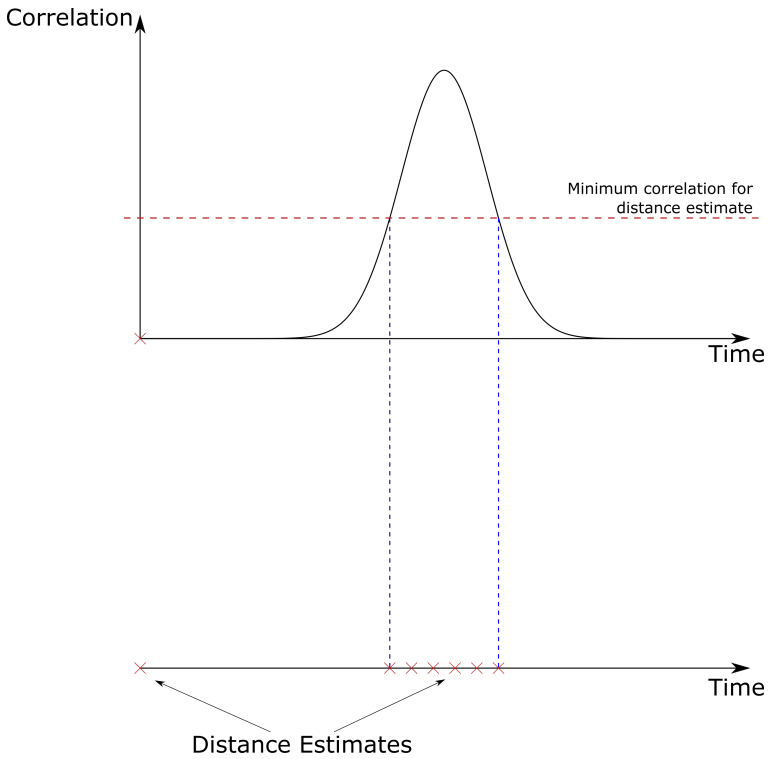

Figure 31: A diagram showing how a single datum point can be used to generate distance estimates at multiple points in space

A list of estimates is created. The first set of data gathered by the sensors is added to the list of estimates with a displacement of 0. For each subsequent set of data gathered, the new data is correlated with each set of data in the estimates list. If these correlation values are high enough, a distance estimate is produced. This is added to the list of distance estimates.

Applying this method to a simulation gives Figure 32. It can be seen that distance estimates are initially sparse. The first new distance estimates occur at an approximate distance of 50mm. As expected, the number of distance grow over time until distance estimation is possible with any new sample.

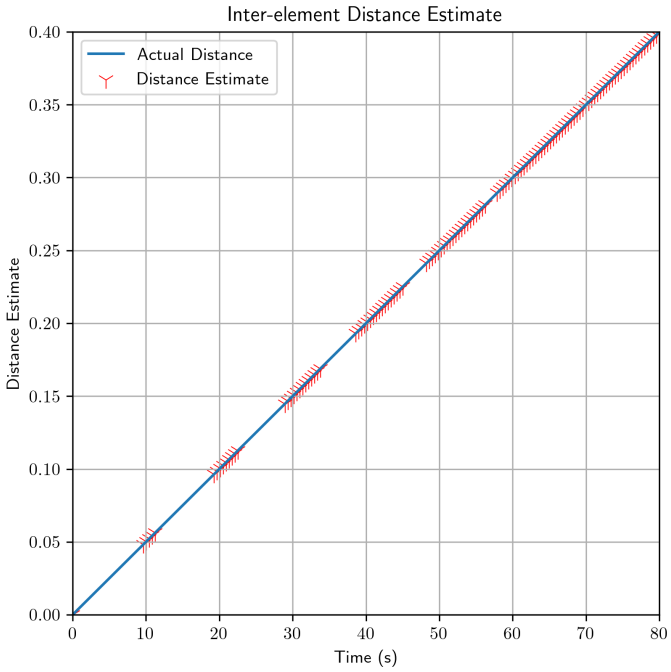

Figure 32: Distance estimation with ideal correlation and non-uniform offsets

4.9 Applying Distance Estimation to Real Data

The previous section has presented a technique for estimating distance travelled using 3 sensors spaced at irregular offsets. This has been applied to real data gathered by the A121 sensors. A new sensor array was designed for this purpose. The modules are offset by 50mm and 55mm. A set of data was then gathered at different along-track positions with a scene of steel wool. The distance estimation algorithm was then applied to this dataset. Distance estimation was limited to samples with over 0.8 correlation. This value was chosen from inspection of Table 6. This ensures that only estimates with high probabilities of being correct are calculated. Figure 33 shows the output of this algorithm. It can be seen that the distance estimation method tracks the motion of the sensors relatively well. The average error per sample is 3.4mm. This value represents the uncertainty in any single estimate. It can be seen that there are occasional outlier estimates that have a high degree of error. These occur at approximately sample 150 and sample 210.

Due to this, a moving average has been calculated, averaging the estimates from the previous and next 10 samples. This smooths out the outlier samples to give a better distance estimate than the individual estimates. However, this moving average gives a poor distance estimation when applied to sparse estimates.

A line of best fit has been plotted to determine the rate at which the estimates diverge from the actual distance travelled. The requirement to intersect the origin (0,0) has been specified. The final error in this line is 4.81mm. The total distance travelled is 650mm, meaning that error is 0.74% of distance travelled (ie: for every 1m travelled, an error of 7.4mm is introduced).

Significant oscillations in distance estimate occur at around sample 400 and continue, appearing to increase in amplitude, until the end of the data. It is hypothesised that these artefacts are caused by outlier estimates. Any outlier estimates will subsequently be used for further distance estimation, meaning that any error present in this outlier will be carried forward into a next estimate. This would have the effect of creating over/under estimates approximately 50mm from the outlier. This effect may continue long after the outlier was first present. The oscillations in Figure 33 appear to occur approximately every 50mm, so it is likely that this is what is happening. Whether these errors would continue to grow is unknown. An

experiment with a longer range of along-track movement would be desirable to investigate the longer term effects of these issues.

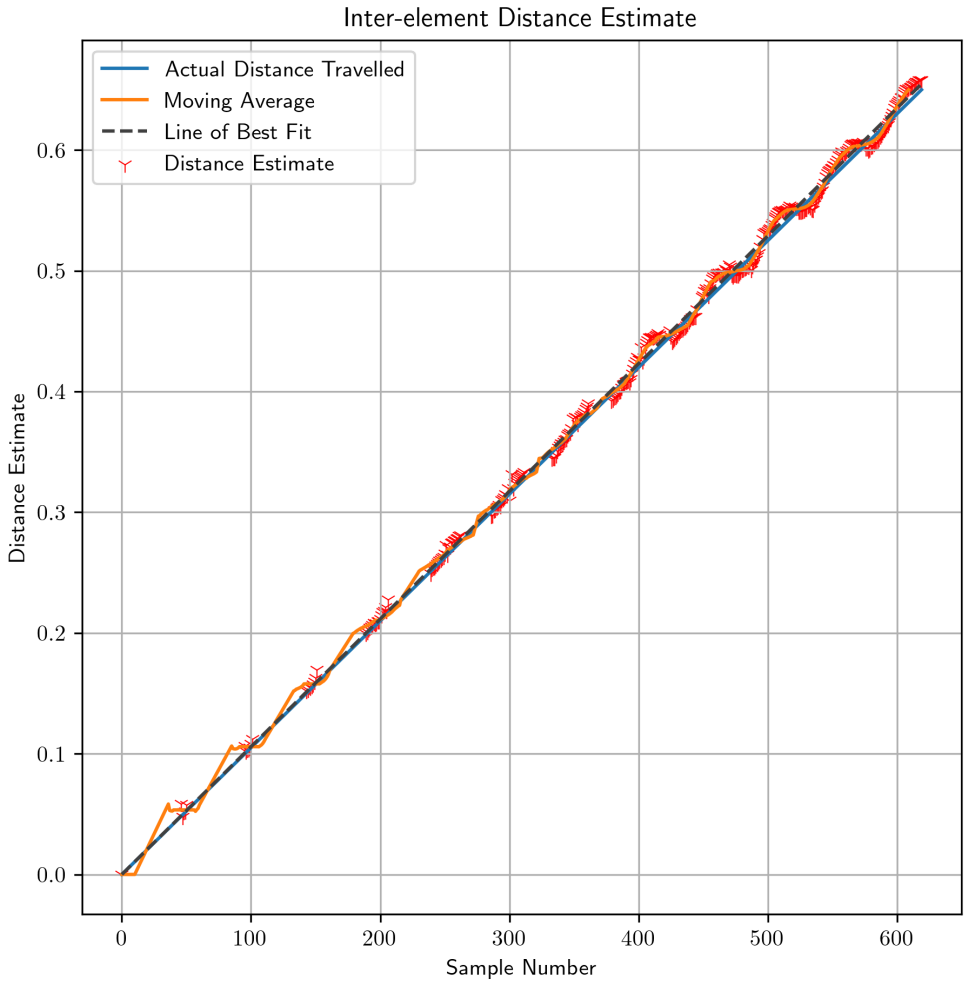

Figure 33: Distance estimation for a three sensor array with a steel wool scene

This same experiment was run for a scene with no steel wool and with fewer samples. This was done to simulate an environment with no strong reflectors or features. In this case, the footprint of the sensors only includes a carpet with no objects present. It was found that the previous distance estimation algorithm was not able to produce a single estimate. This is because the threshold for distance estimation (0.8 correlation) is too high. Lowering this limit to 0.6 gives the distance estimates in Figure 34. The average error per sample is 23.1mm.

The line of best fit has a final error of 35.3mm, corresponding to an error of 5.44% of distance travelled. This is significantly larger than the error growth in Figure 33. This is expected as when scene quality is low, correlation profiles between the sensors are poorly defined, which degrades the quality of any distance estimates.

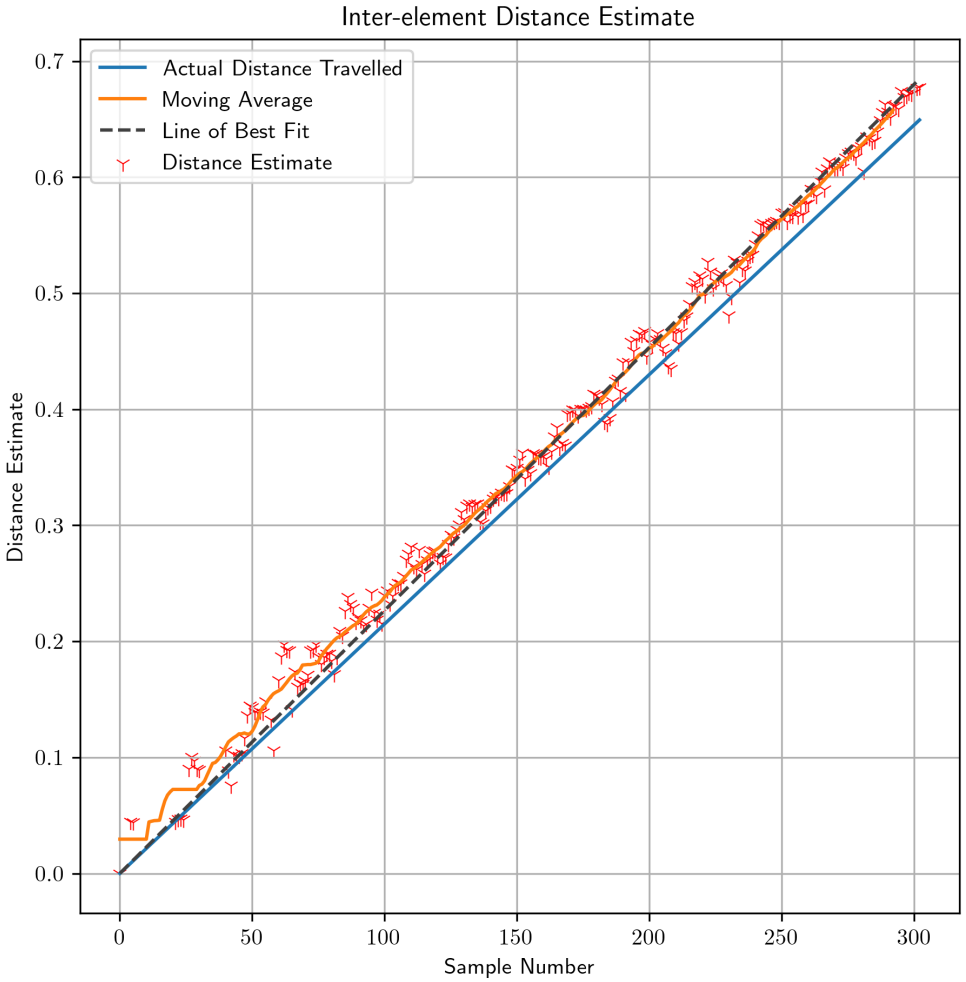

Figure 34: Distance estimation for a three sensor array with an empty scene

4.10 Modified Sonar Distance Estimate Method

It is not possible to utilise the distance estimation method discussed in the literature review due to the correlation profile for these sensors being too narrow [13]. However, a modified method is presented here, which allows use of this method on narrow correlation peaks with one additional sensor.

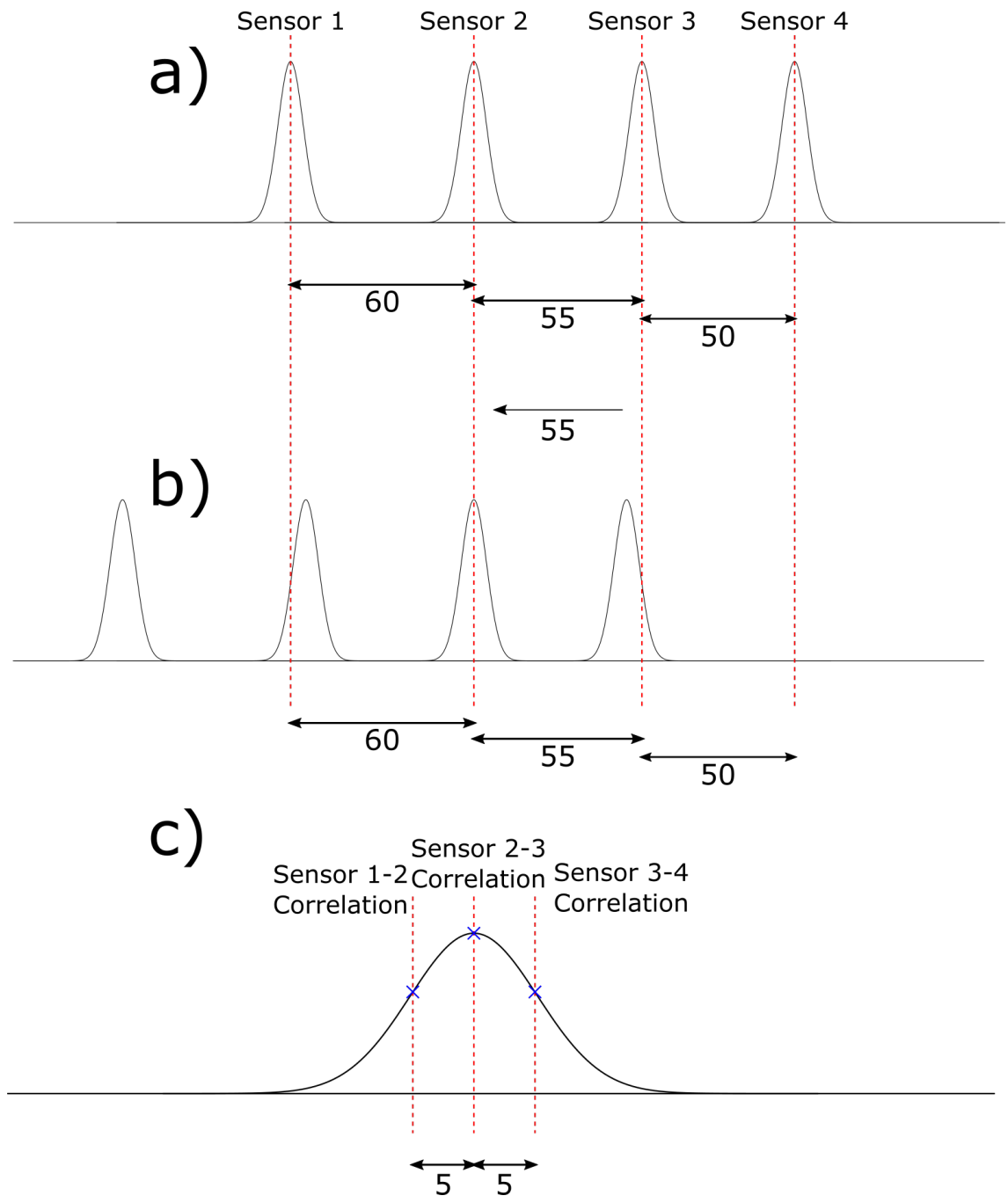

Figure 35: A method to combine correlation values from three distinct correlation peaks (a) & (b) to form a virtual correlation peak (c)

The vertical red lines in Figure 35 (a) represent an array of four sensors. These are offset by uneven distances, where the separation between sensors is 5mm less than the previous separation. The four correlation peaks in Figure 35 (a) correspond to how correlation would vary as the sensors move away from the initial positions. Figure 35 (b) shows the same array moved forward 55mm. It can be seen that because of the offsets, each sensor is correlating differently with the sensor in front of it. These 3 correlation values are on different correlation peaks. However, if it is assumed that the shape of all correlation peaks is identical, the correlation values can be used to recreate the shape of this correlation peak, and thus estimate distance. This is shown in Figure 35 (c). This method was developed late in the project, alongside the method in Section 4.8. Unfortunately, only 3 A121 sensors were available at this point in time and it was not feasible

to procure more due to time constrains. Therefore, this method has not been tested on experimental data.

4.11 SAR Imaging

SAR images were processed for the A121 sensors with a 12° beamwidth. The process for creating SAR images is discussed in Section 2.7. The output of this processing technique is an array of numbers. The position of a number in the array corresponds to the position in space and the magnitude of the number corresponds to the strength of reflections at this point.

The first scene consists of a steel hemisphere placed in the centre of the footprint of the radar, shown in Figure 36. The diameter of this hemisphere is 130mm. The maximum range of the footprint was 1.5m. Therefore, the minimum length requirement was calculated to be 0.315m using Equation 1. The data was gathered over a length of 700mm. As explained in Section 2.7.1, this means that 385mm of along-track image data will be fully focused. The effective antenna length was calculated to be 20.8mm using Equation 2. As mentioned in Section 2.7.2, spacing between samples should be below $L/4$. The data was gathered with sample spacings of 1.15mm, which is well within this requirement.

Figure 36: The first scene used in creating a SAR image - a metal hemisphere

Figure 37 shows the raw data (left), and the focused SAR image (right).

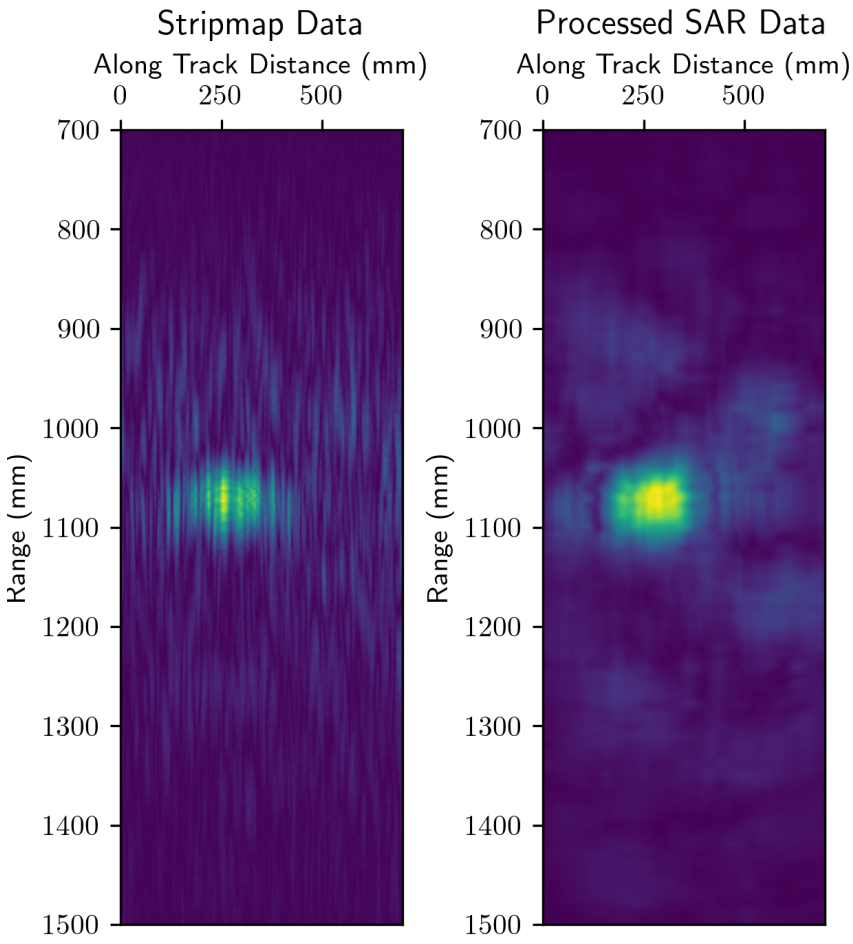

Figure 37: SAR image of a hemisphere reflector

The second scene consists of the mirror reflector placed closer to the sensor array, as shown in Figure 38. The mirror is 260mm in length. The processed SAR image is shown in Figure 39.

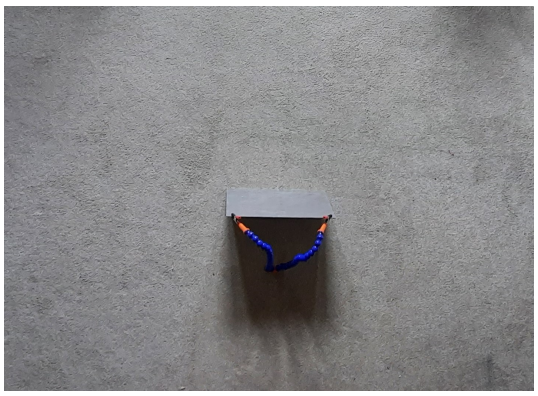

Figure 38: The second scene used in creating a SAR image - a metal mirror

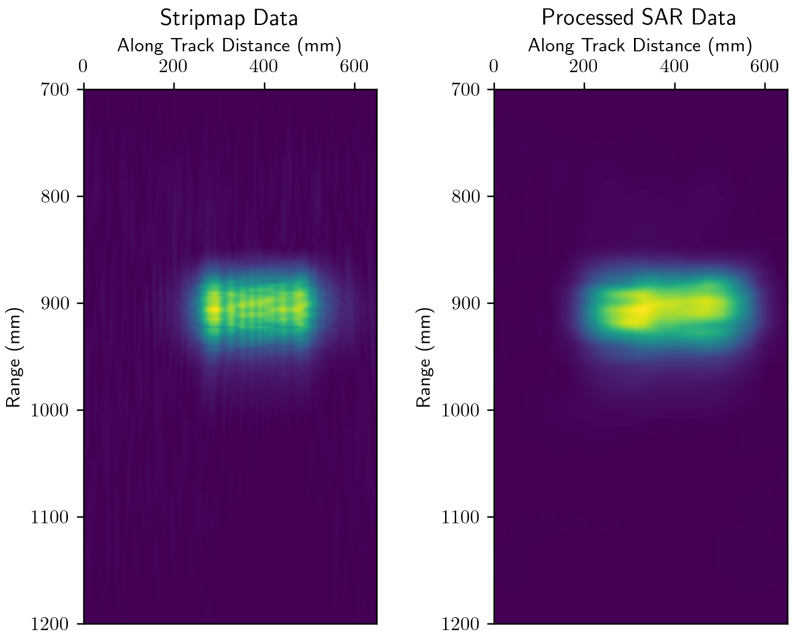

Figure 39: SAR image of a mirror reflector

5 Discussion

5.1 Sensor Types

Throughout these experiments, the A121 sensors have performed significantly better than the A111 sensors. It was shown that the A111 sensor inbuilt averaging feature creates significant artefacts in the data, which makes the data unsuitable for distance estimation or SAR imaging. While it is possible to eliminate this issue by taking samples manually, this is undesirable as it is slower. Assuming that a robot is constantly moving, faster sampling means that distance travelled between samples is lower. Averaging closely-spaced samples gives a better representation of the surroundings, and so faster sampling means data is more representative of the surroundings. The A121 sensors do not suffer from these same artefacts. The quality of signal for the A121 sensors does not significantly improve with more samples, meaning that a low number of samples gives the same high quality data as many samples.

5.2 Correlation Profile

The correlation profile of the A111 sensors poorly follows the theoretical correlation profile. This is likely due to high noise in the signal. It may also be due to differences in transmitted signal or signal pre-processing, that modify the data received.

It can be seen from Figure 19 that increasing the number of samples can improve the quality of the correlation peak for the A111 sensors. However, this may not be possible in the desired application for the reasons discussed in Section 5.1.

The A121 sensors follow the theoretical profile closely for high values. Two distinct behaviours are observed. Firstly, the real correlation profile is not symmetrical. This may be caused by random variation in the distribution of steel wool in the scene; some regions (such as those with a strong phase pattern) may correlate better than others.

Secondly, small peaks in the correlation profile can be observed far from the initial correlation peak. This is likely cause be a repeating pattern in the scene - potentially caused by a repeating raised bump in

the steel wool scene, causing an amplitude peak.

5.3 Inter-Sensor Correlation

Table 5 shows that correlation between different A111 sensors is poor. It is hypothesised that this is due to noise in the signal, as well as slight manufacturing differences between sensors leading to different data being collected. The A121 sensors perform significantly better; the lowest correlating pair of A121 sensors correlate more highly than the highest correlating pair of A111 sensors (Tables 5 & 6). This difference in correlation between different sensors is further reflected in the inter-sensor correlation against distance plot (Figure 24); correlation between A121 sensors show high peaks in correlation at the correct places, while A111 sensors are missing peaks. It is not possible to distinguish any significant peak in correlation between A111 sensors 3 & 4 or 1 & 4. This makes any form of distance estimation impossible. It was for this reason that no distance estimation was attempted using the A111 sensors.

5.4 Distance Estimation Method from Sonar Literature

The method for distance estimation detailed in the literature review [13] was not possible with the sensors used in this project. This is due to the correlation profile (which is dependent on the sensors) not being wide enough, which prevents 3 sensors in an array all correlating highly with a single previous data sample. This issue may be overcome by using custom modules, lenses, and lens holders. As the sensors are approximately 5mm wide, it is theoretically possible to position them at approximately 5mm separations. The distance from the first sensor to last sensor in this theoretical array is 10mm. The correlation profile of the A121 sensors is well-defined up to 10mm either direction from a datum (Figure 18), which is twice the minimum width required. A custom circuit board and custom lenses would need to be designed for this purpose. A system such as this could provide an extremely compact sensing solution for distance estimation.

A modified version of this distance estimation method that can work with large sensor separations is presented in Section 4.10. Testing of this method was not possible in this project due to a combination of time and equipment constraints.

5.5 Custom Distance Estimation Method

The custom distance estimation algorithm developed in this report was shown to produce accurate distance estimates, even in the presence of disturbances such reduced correlation peak height. This algorithm was designed to work with 3 sensors, as this was the number of A121 sensors available in this project. This algorithm has potential to work with more than 3 sensors. More sensors would increase the number of estimates averaged, thereby decreasing outlier estimates. Furthermore, more sensors would increase the rate at which estimates can be found, reducing the time for which estimates are sparse when navigation begins. Beginning of navigation is a potential weakness of this algorithm; an array must to travel several integer multiples of 50mm before distance estimates for every new sample of data is available. Navigation on a very small scale using this algorithm may be infeasible.

The errors seen when applying this method to real data are relatively low; a 4.81mm average error when in a favourable scene and a 23.1mm average error when presented with an unfavourable scene. This difference shows that accuracy is highly dependent on scene quality, similar to optical sensors requiring a clear, well-lit scene to gather accurate data. However, in the context of radar, *high quality* refers to a scene populated with many strong and complex reflectors. This is a significantly different requirement than those for cameras or lidar, thus showing that radar may be used to improve performance of existing systems in low-visibility conditions.

The distance estimation method developed in this report has significant scope for expansion. Estimates arise from correlating sensor data with a known previous point. These estimates are then used to create more estimates. In this project, this has been used on a sample-by-sample basis; each new set of data is analysed for estimates before moving on to the next. However, this method could be applied repeatedly.

Once new estimates are found, these can then be used to generate more estimates for previous samples. This process could be repeated, essentially comparing every sample to every other sample in order to extract as much information about the data as possible. This may be computationally intensive, but could significantly increase the accuracy of estimates.

5.6 Consequence of 1-Dimension Simplification

These experiments have shown that distance estimation in 1-dimension with no change in orientation can be done using the methods and technology put forward. Navigation in 3 dimensions may be possible by applying this method with 3 arrays each facing different directions. A system such as this would be reliable by itself, provided that the system did not change orientation. Changes in orientation would significantly change the data that the sensors receive, and therefore would likely make distance estimation impossible. There exist few specialist applications where this change of orientation does not occur, although one application is robots that run on linear tracks, such as those used in some warehouses [28]. Therefore, while the simplification to 1-dimension does not limit the applications of this method, the simplification of no changes in orientation does significantly. For this reason, it is likely that applications using this method would need to combine radar sensors with other sensing methods.

5.7 Synthetic Aperture Radar Imaging

Figures 37 & 39 both compare a the raw data from the sensor (left) to the focussed SAR image (right). In both cases, it can be seen that SAR processing improves the quality of the image - showing clusters of high signal amplitude where the objects are. In Figure 37, the cluster of high amplitude pixels measures approximately 110mm across - corresponding to the 130mm actual diameter of the hemisphere. In Figure 39, the length of the mirror appears to be approximately 280mm - corresponding to the 260mm actual length. Both estimates of dimensions are somewhat imprecise due to a smooth variation for low amplitude to high amplitude. More precise estimates could be achieved by fine-tuning the processing of the images.

It has been shown that these sensors are able to produce SAR images that are representative of the surroundings. SAR imaging requires accurate knowledge of where data was taken. It is unlikely that the distance estimates are able to provide the necessary accuracy, and so if SAR imaging were to be incorporated into a navigation system, additional sensing methods would be required to produce this accurate location data. It should be noted that SAR imaging does not require the sensors to be travelling in a straight line facing a direction perpendicular to the direction of travel, as the distance estimation method does. However, accurate position and orientation of the sensors is required to focus SAR images.

6 Conclusions

This project has shown that using coherent radar sensors for navigation in 1 dimension is feasible. It was found that the current generation of radar sensors, the A111 sensors, do produce data that correlates well between sensors. This is likely caused by small differences in the physical construction of the sensors. This factor prevents use of A111 sensors to estimate distance travelled. However, the next generation of sensors, the A121 sensors, can be used to provide accurate distance estimates in 1 dimension on a small scale. A custom distance estimation method has been devised to work with off-the-shelf sensors and lens holders. Distance estimates using the A121 sensors have been shown to have a maximum error rate of 5.44% (ie: for every 1m travelled, 54.4mm of error in introduced into the estimate). However, this appears to be dependent on the environment, and thus error rates could be higher or lower depending on the environment. A favourable scene produced a significantly better error rate of 0.74%. Variation of accuracy with scene quality is a potential limitation of this method. More work needs to be done to determine the scope of these effects. Due to the 1-dimensional nature of the distance estimation algorithm in this report, it is unlikely that many applications exist for this method alone. This algorithm could be extended into 3 dimensions

with relative ease. However, changes in orientation would severely reduce the accuracy of this method. Therefore, application of this method would likely need to be integrated with other sensing methods in real-world applications. Synthetic aperture radar has shown to be possible with these sensors. However, it has not been fully investigated. This project has determined that SAR images can be used to detect the approximate size and shape of potential obstacles, potentially offering a solution to the mapping problem.

7 Future Work

7.1 Future Experiments

This project has shown that displacement estimation on a small scale is feasible using an array of Acconeer A121 sensors. Further work should be done to estimate the viability of this method.

Testing the displacement estimation over a longer distance would give a more accurate indication of how estimates might increase in error over time. Building a longer testrig would be straightforward, but compliance of the belt would increase positioning errors with distance. Therefore, the next sensible step would be to mount an array of sensors on a vehicle. Accurate tracking of the location of the vehicle could be done using a system of Optitrack cameras, which have a range of 30m.

Future work should ideally test both distance estimation and SAR in a greater variety of environments. This would determine how much the accuracy of these systems is dependent on the environment. If accuracy is highly dependent on a specific environment, the potential application of this method will be limited. These experiments should also be carried out in low-visibility conditions to test that distance estimation is unaffected by smoke.

7.2 Integration Into Navigation Algorithms

It has been concluded that there are few applications where the method of navigation in this report is feasible without other supporting methods. Combining this sensing method with other methods is vital for a more versatile navigation algorithm. Work should be done to integrate this method of distance estimation into navigation algorithms such as SLAM. SAR could also be incorporated into this method to perform object detection or potentially collision avoidance, which would aid in path planning.

8 References

References

[1] S. Cyrill, *Robotic Mapping and Exploration*. 1st Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[2] Makarenko, A.A, S.B Williams, F. Bourgault, and H.F Durrant-Whyte, "An Experiment in Integrated Exploration" in IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002.

[3] J.M. Santos, M.S. Couceiro, D. Portugal, R.P. Rocha. "A Sensor Fusion Layer to Cope with Reduced Visibility in SLAM", *Journal of Intelligent Robotic Systems*, vol 80.3-4 2015

[4] C. Brunner, P. Thierry, V. Teresa, J. Underwood. "Selective Combination of Visual and Thermal Imaging for Resilient Localization in Adverse Conditions: Day and Night, Smoke and Fire", *Journal of Field Robotics*, vol 30.4, 2013.

[5] L. Hyukjung, J. Chun, K. Jeon. "Experimental Results and Posterior Cramér-Rao Bound Analysis of EKF-Based Radar SLAM With Odometer Bias Compensation", *IEEE Transactions on Aerospace and Electronic Systems*, vol 57.1, 2021

[6] A. Faiza, B. Georg, V. Martin. "A Rotating Synthetic Aperture Radar Imaging Concept for Robot Navigation", *IEEE Transactions on Microwave Theory Techniques*, vol 62.7, 2014

[7] NASA. (2019, Feb. 19). *Radio Spectrum* [Online]. Available: `https://www.nasa.gov/directorates/heo/scan/spectrum/radio_spectrum/`

[8] Meikle, Hamish, *Modern Radar Systems*, 2nd ed. Boston: Artech House, 2008

[9] S. Kingsley, S. Quegan. *Understanding Radar Systems*. Raleigh, NC: SciTech, 1999.

[10] Acconeer, "A111 - Pulsed Coherent Radar (PCR)", A111 datasheeet, 2021-10-22 [v2.6].

[11] J.P. Fitch. *Synthetic Aperture Radar*. Berlin: Springer-Verlag, 1988.

[12] E. Lai, *Practical Digital Signal Processing for Engineers and Technicians*. 1st ed. Amsterdam; London: Newnes, 2004

[13] D.C. Brown, I.D. Gerg, T.E. Blanford. "Interpolation Kernels for Synthetic Aperture Sonar Along-Track Motion Estimation", *IEEE Journal of Oceanic Engineering*, vol 45.4, 2020

[14] J. Oeschger, "Method of estimating along-track displacement of an underwater vehicle", U.S. Patent US7342847, 2008-03-11

[15] Henrik. (2015, Oct. 14). *Homemade synthetic aperture radar* [Online]. Available: `https://hforsten.com/homemade-synthetic-aperture-radar.html`

[16] G.L. Charvat, J.H. Williams, A.J. Fenn, S. Kogon, J.S. Herd. (2011, Jan.). *Build a Small Radar System Capable of Sensing Range, Doppler, and Synthetic Aperture Radar Imaging* [Online]. Available `https://ocw.mit.edu/courses/res-ll-003-build-a-small-radar-system-capable-of-sensing-range-doppler-and-synthetic-apertu`

[17] J. Opretzka, M. Vogt, H. Ermert. "A Model-based Synthetic Aperture Image Reconstruction Technique for High-frequency Ultrasound", *IEEE International Ultrasonics Symposium*, 2009

[18] A.J. Hunter, "Underwater Acoustic Modelling for Synthetic Aperture Sonar", Ph.D. dissertation, Dept. Electrical and Computer Engineering, University of Canterbury, 2006.

[19] Acconeer. *Products* [Online]. Available: `https://www.acconeer.com/products/`

[20] Mouser. *XC112/XR112 Evaluation Kits* [Online]. Available: `https://www.mouser.co.uk/c/?marcom=163458703`

[21] Acconeer, "Getting Started Guide Lens Evaluation Kit LH112/122/132", LH112/122/132. 2020-10

[22] RS Components, "RS NUMBER: 1805279", 180-5279, 2018

[23] Acconeer. Acconeer Exploration Tool docs [Online]. Available: `https://docs.acconeer.com/en/latest/`

[24] Acconeer. Radar sensor introduction [Online]. Available: `https://docs.acconeer.com/en/latest/sensor_introduction.html`

[25] L.D. Haugh. "Checking the Independence of Two Covariance-Stationary Time Series: A Univariate Residual Cross-Correlation Approach" *Journal of the American Statistical Association* vol 71.354, 1976

[26] Neha Jirafe, Analytics Vidhya. (2019, Dec. 19). *How to filter noise with a low pass filter - Python* [Online]. Available: `https://medium.com/analytics-vidhya/how-to-filter-noise-with-a-low-pass-filter-python-885223e5e9b7`

[27] B. Thomas, A.J. Hunter. "Coherence-Induced Bias Reduction in Synthetic Aperture Sonar Along-Track Micronavigation", *IEEE Journal of Oceanic Engineering*, Vol. 47, No. 1, Pp. 162-178, 2022

[28] BBC News. (2017-02-7). *The Ocado warehouse run by robots* [Online]. Available: `https://www.bbc.co.uk/news/av/business-38897417`

9 Appendices

9.1 Test-rig Design Requirements

Table 7: Design requirements for the test-rig

ID	Specification	Must / Should / Could	Anticipated Difficulty (Easy / Medium / Hard)	Achieved (y/n)
1	Constrain sensor in 2 dimensions	Must	Easy	Y
2	Allow movement in 1 direction	Must	Easy	Y
7	Have moveable range of over 0.5m	Should	Easy	Y
8	Adjustable sensor orientation	Should	Easy	Y
4	Accurate positioning (0.1mm)	Must	Medium	Y
3	Provide relative positioning in 1D	Must	Medium	Y
5	Provide absolute positioning in 1D	Should	Medium	N
6	Be portable/portable power source	Could	Hard	N

9.2 Schematic For Testrig Electrical Circuit

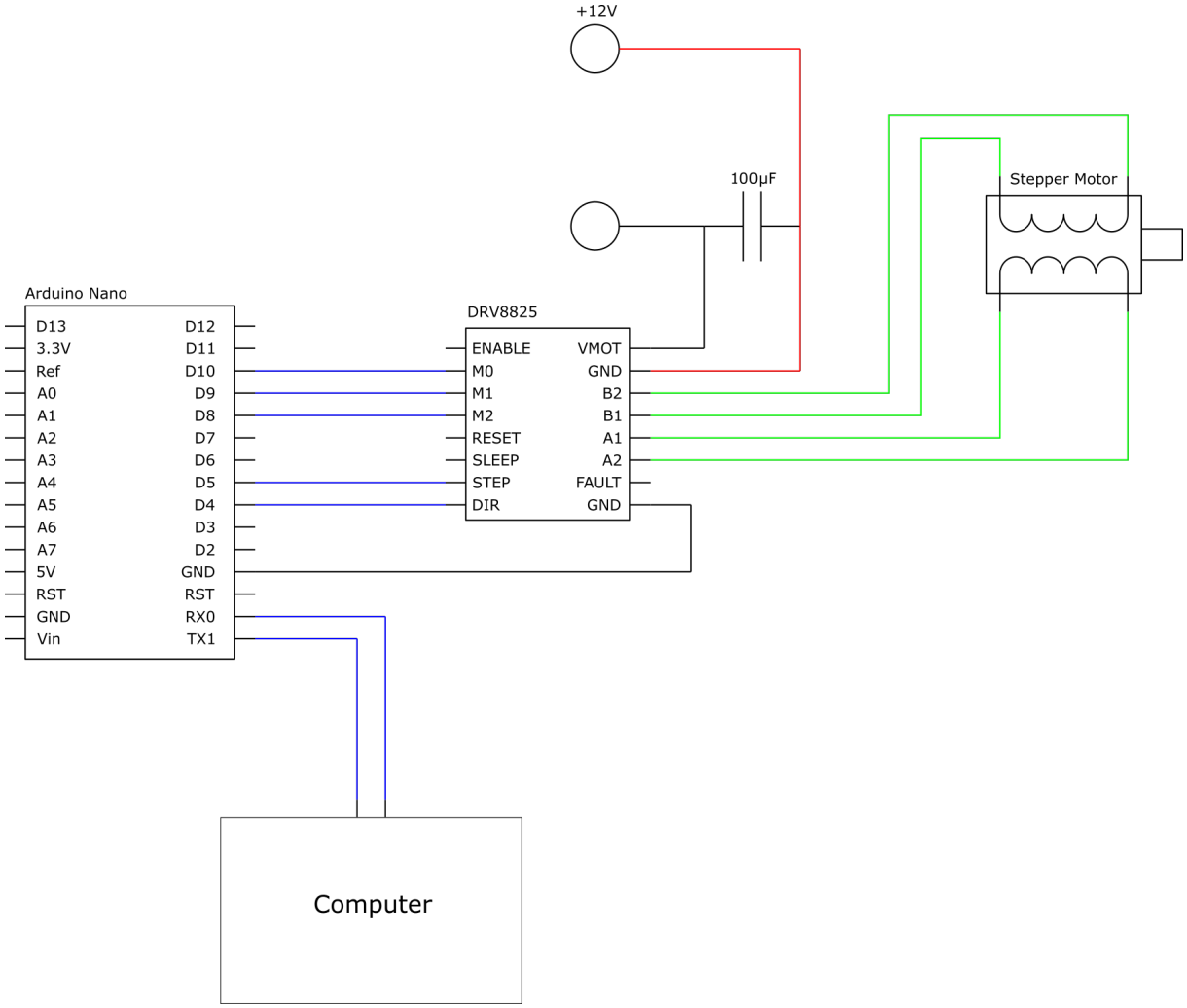

Figure 40: Circuit diagram for the test-rig

9.3 Radar Footprint Geometry Equations

These equations are associated with Figures 11 and 12.

$$x_R = \frac{h}{\tan(\theta)} \tag{12}$$

$$x_R - x_{min} = h\left(\frac{1}{\tan(\theta)} - \frac{1}{\tan(\theta+\psi)}\right) \tag{13}$$

$$x_{max} - x_R = h\left(\frac{1}{\tan(\theta-\psi)} - \frac{1}{\tan(\theta)}\right) \tag{14}$$

Note that:

$$x_R - x_{min} \neq x_{max} - x_R \tag{15}$$

therefore:

$$\Delta x = x_{max} - x_{min} = h\left(\frac{1}{\tan(\theta-\psi)} - \frac{1}{\tan(\theta+\psi)}\right) \tag{16}$$

Also the following range equations:

$$R_{min} = \frac{h}{\sin(\theta + \psi)} \tag{17}$$

$$R = \frac{h}{\sin(\theta)} \tag{18}$$

$$R_{max} = \frac{h}{\sin(\theta - \psi)} \tag{19}$$

$$y_R = 2R \cdot \tan(\psi_H) \tag{20}$$

9.4 Real Correlation Profile Imperfections - Individual Effects

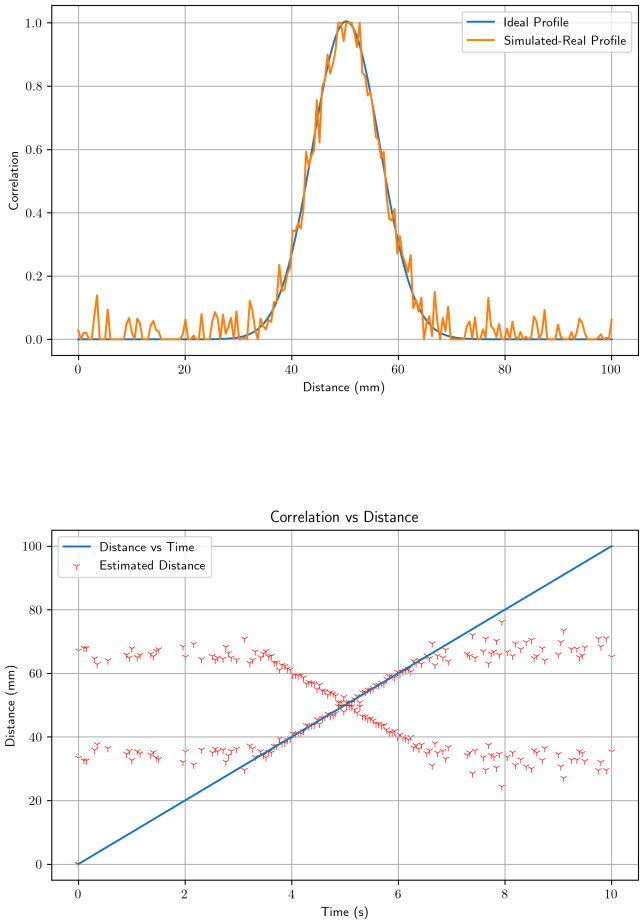

Figure 41: Effect of noise on a correlation profile

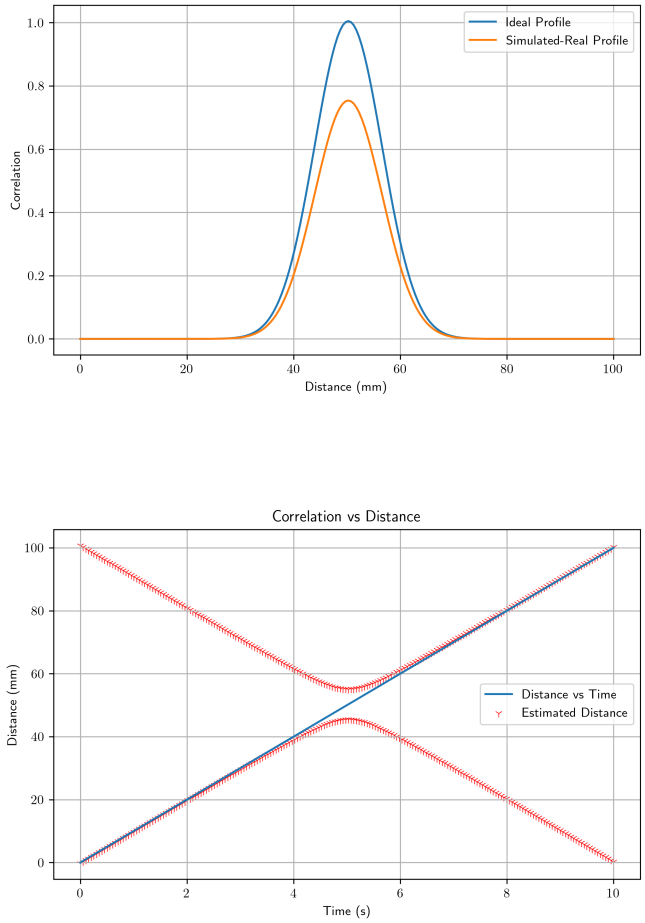

Figure 42: Effect of a reduced peak height on a correlation profile

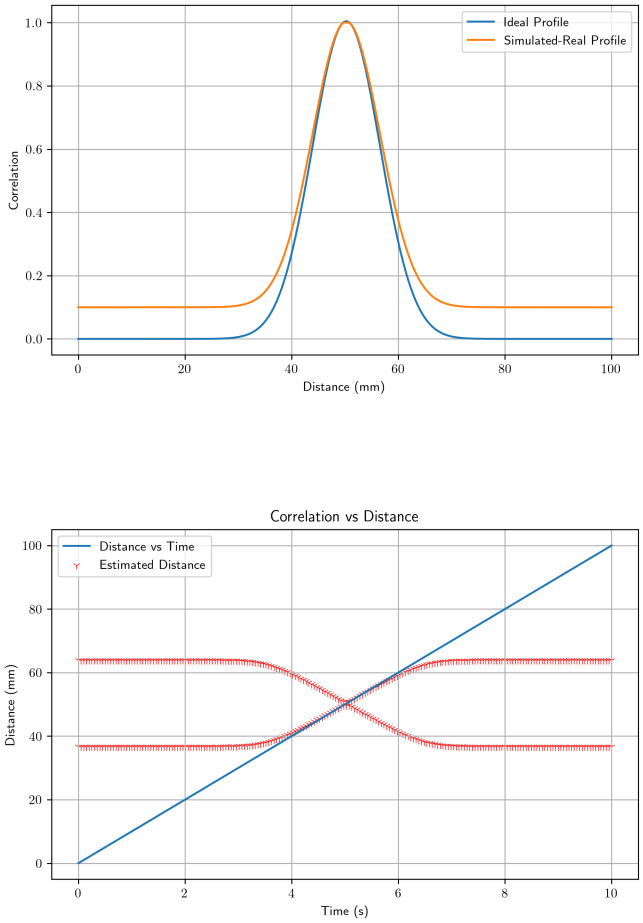

Figure 43: Effects of raised floor height on a correlation profile

9.5 SAR Processing Computational Method

Consider a signal received by a radar sensor. This signal is projected by an angle over an area, as shown in Figure 44.

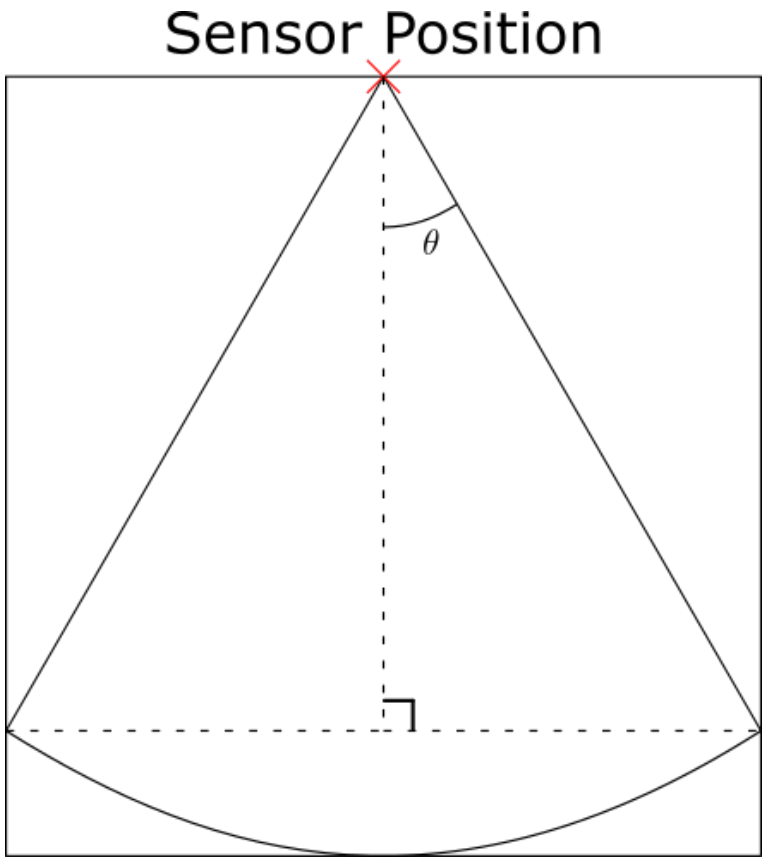

Figure 44: SAR imaging signal projection

The signal is simply a list of complex numbers. The geometry in Figure 44 can be used to generate a list of co-ordinates in 2D space and the index of associated signal value that is projected there. In a 2D matrix, this would look like so:

$$\begin{bmatrix} \times & \times & 1 & \times & \times \\ \times & 2 & 1 & 2 & \times \\ 3 & 2 & 2 & 2 & 3 \\ 3 & 3 & 2 & 3 & 3 \\ \times & 3 & 3 & 3 & \times \end{bmatrix}$$

Note that this is not to scale. X's denote positions where no signal data should be projected onto the image. Numbers denote the index of the value in the signal that should be project to that position. This template can be generated once, but used multiple times with different x axis offsets to overlay different signals at different x positions on a large 2D array. The addition of projected signals to a grid can then be achieved in a single line of code:

```
imageArray[[listOfXCords], [listOfYCords]] += signal[indexList]
```

A further processing step is weighting the signal data points instead of either just adding them or not. This more accurately represents the radar. Modelling the beamwidth strength function (can be done using a hamming window) and using this to weight the data points improves the accuracy of the image.

9.6 Python Module Code

There are 3 parts to the module written for this report.

- Main functions, used for experiments, loading & saving data, and basic analysis. Located in "init.py"

- Simulation functions, used for creating and running simulations. Located in “sim.py”
- Mapping functions, used to create SAR images. Located in “map.py”

9.6.1 Main

```
# Hugo Weston 2022 hw876@bath.ac.uk

# Import other parts of this library:
import fyp_package.map
import fyp_package.sim

import numpy as np
import math
import acconeer.exptool as et

import scipy.io as scio # used to load matlab files

import serial
import time
import os
import pickle

import matplotlib.pyplot as plt


from scipy import signal

import copy


def reload_modules():
    import imp
    imp.reload(fyp_package.sim)
    imp.reload(fyp_package.map)
    imp.reload(fyp_package)


# Classes ------------------------------------------------------------------------------
class linearMetadata:
    # This class contains all data for doing a linear run
    def __init__(self,directory,lens="unspecified",rangeInterval=[0.06,7.0],steps=range(0,100,10),serialPort="COM8",IP="192.168.1.75",sensorType="default"):
        # Set up class with lots of default parameters
        self.sensorType = sensorType
        self.A121_start_point = 0
        self.A121_num_points = 0
        self.lens = lens # Exists in name only, doesn't change anything in the code
        self.directory = directory
        self.IP =  IP
        self.update_rate = 30
        self.range = rangeInterval
        self.steps = steps
        self.HWAAS = 30 # Hardware Accelerated Average Samples
        self.stepmode = "stepmode_1"
        self.gain = 0.5
        self.serialPort = serialPort
        self.stepFactor = 0.20
        self.lowPassFilter = 0.5 # 0.5 = Turn filter off by default
        self.samples = 20 # Number of samples to take per data point
        self.stepPause = 0.5 # Time to pause inbetween steps
        self.datapointPause = 0.5 # time to wait after getting a set of data. Does not affect getting multiple samples, only affects calls to getnewdata()
        self.sensors = [1]
        self._saveMode = "samples"
        self.profile = ""

        if self.sensorType == "A121":
            self.create_depths()

    def create_depths(self):
        # Range for A121 is determined by a number of points, no acutal distance
        # Find number of points assocaited with chosen distance
        spacing = 2.5e-3

        self.A121_start_point = int(self.range[0]/spacing)

        current_position = self.A121_start_point * spacing
        num_points = 1
        while True:
            current_position += spacing
            if current_position > self.range[1]:
                break
            num_points += 1

        self.A121_num_points = num_points

    def numberOfSteps(self):
        # return the number of steps in the run
        return len(self.steps)

    def stepsFromDistance(self, startPos, stopPos, samples):
        # Generate a list of positions (in steps) from a start
        # and end position and a number of samples

        # Because of the conversion from distance to steps, stoppos
        # and number of samples output may not exactly match input
```

```
    startFullSteps = startPos / self.stepFactor
    stopFullSteps = stopPos / self.stepFactor

    startSteps = startFullSteps * self.stepModeFactor()
    stopSteps = stopFullSteps * self.stepModeFactor()

    startSteps = int(startSteps)
    stopSteps = int(stopSteps)

    stepsBetweenSamples = int((stopSteps - startSteps) / samples)

    self.steps =  [*range(startSteps,stopSteps,stepsBetweenSamples)]

def setStepMode(self, inputStepMode):
    if inputStepMode == 1:
        self.stepmode = "stepmode_1"
    elif inputStepMode == 2:
        self.stepmode = "stepmode_2"
    elif inputStepMode == 4:
        self.stepmode = "stepmode_4"
    elif inputStepMode == 8:
        self.stepmode = "stepmode_8"
    elif inputStepMode == 16:
        self.stepmode = "stepmode_16"
    elif inputStepMode == 32:
        self.stepmode = "stepmode_32"
    else:
        raise Exception("Incorrect step mode selection: '" + str(inputStepMode) + "'")

def stepModeFactor(self):
    # The factor by which the chosen stepmode reduces the step
    return int(self.stepmode.replace("stepmode_",""))

def alongTrackDirection(self):
    # Return a list of distances to take data at (mm)
    return np.array(self.steps) * self.stepFactor * (1/self.stepModeFactor())

def setProfile(self,inputProfile):
    if self.sensorType == "A121":
        # No such thing as profile 1 or 5 for new sensors
        if inputProfile == 2:
            self._profile = et.configs.A121SparseIQConfig.Profile.PROFILE_2
        elif inputProfile == 3:
            self._profile = et.configs.A121SparseIQConfig.Profile.PROFILE_3
        elif inputProfile == 4:
            self._profile = et.configs.A121SparseIQConfig.Profile.PROFILE_4
        else:
            raise Exception("Incorrect profile selection: '" + str(inputProfile) + \
                            "' \n" + "Remember that there is no Profile 1 or 5 for the A121 Sensors" )
    else:
        if inputProfile == 1:
            self._profile = et.configs.IQServiceConfig.Profile.PROFILE_1
        elif inputProfile == 2:
            self._profile = et.configs.IQServiceConfig.Profile.PROFILE_2
        elif inputProfile == 3:
            self._profile = et.configs.IQServiceConfig.Profile.PROFILE_3
        elif inputProfile == 4:
            self._profile = et.configs.IQServiceConfig.Profile.PROFILE_4
        elif inputProfile == 5:
            self._profile = et.configs.IQServiceConfig.Profile.PROFILE_5
        else:
            raise Exception("Incorrect profile selection: '" + str(inputProfile) + "'")
    self.profile = "Profile " + str(inputProfile)

def savetxt(self):
    # Save the run info to a file in a human-readable manner
    touchDir(self.directory) # create directory if it doesn't already exist
    selfDict = vars(self)

    f = open("./" + self.directory + "/run_info_readable.txt","w")
    f.write("Data saved for reading only.\n")
    f.write("TIMESTAMP of file creation (not time of run):\n")
    f.write(str(time.ctime(time.time())))
    f.write("\n")

    f.write("Calculated Information:")
    f.write("\n")
    f.write("Number of datapoints: " + str(len(self.steps)))
    f.write("\n")
    f.write("Min Distance: " + str(self.steps[0] * self.stepFactor / self.stepModeFactor()) + " mm")
    f.write("\n")
    f.write("Max Distance: " + str(self.steps[-1] * self.stepFactor / self.stepModeFactor()) + " mm")
    f.write("\n")

    f.write("Variable Information: \n")
    for key in selfDict:
        f.write(str(key) + ": " +  str(selfDict[key]))
        f.write("\n")
    f.close()



def saveInfo(self):
    touchDir(self.directory) # create directory if it doesn't already exist
    #Replacement for the old saveInfo. Does not use pickle and therefore does not require consistent library/module/class names
    if os.path.exists(self.directory + "/run_info.hugo"):
```

```
            print("Warning. File '" + str(self.directory + "/run_info.p") + "' already exists")
            userCheck = input("overwrite? (y/n) ")
            if userCheck.upper() != "Y":
                raise Exception("User chose not to overwrite Data")


        # convert to dictionary
        infoToSave = vars(self)
        f = open("./" + self.directory + "/run_info.hugo","w")
        for key in infoToSave.keys():
            f.write(key)
            f.write(";")
            f.write(str(infoToSave[key]))
            f.write("\n")
        f.close()

    def saveInfoOld(self):
        # Save the run info to a file so it can be loaded by a computer at a later date
        if os.path.exists(self.directory + "/run_info.p"):
            print("Warning. File '" + str(self.directory + "/run_info.p") + "' already exists")
            userCheck = input("overwrite? (y/n) ")
            if userCheck.upper() != "Y":
                raise Exception("User chose not to overwrite Data")


        touchDir(self.directory) # create directory if it doesn't already exist
        f = open("./" + self.directory + "/run_info.p","wb") #.p = pickle file
        pickle.dump(self, f)
        f.close()

# Signal & Data Processing ------------------------------------------------------------------------

def average(inputVector): # averages a series of vectors
    # Designed for input vector to be a 1D array of 1D vectors
    # Will average for each item in the list

    # Used to average multiple samples of data
    avgVector = inputVector[0]
    for i in range(1,len(inputVector)):
        avgVector = avgVector + inputVector[i]
    avgVector = avgVector / len(inputVector)
    return avgVector

# Filtering functions are commonly used, but were taken from following sources:
# https://medium.com/analytics-vidhya/how-to-filter-noise-with-a-low-pass-filter-python-885223e5e9b7
# https://www.delftstack.com/howto/python/low-pass-filter-python/
def butter_lowpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = signal.butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

def butter_lowpass_filter(data, cutoff, fs, order=5):
    # data = vector of data
    # cuttoff = cuttoff frequency to filter out
    # fs = sampling frequency
    b, a = butter_lowpass(cutoff, fs, order=order)
    y = signal.lfilter(b, a, data)
    return y


def ncrossCorr(dataset1, dataset2,mode="normal"):
    # Return the normalised cross correlation between dataset1 and dataset2

    if mode == "phase":
        # Phase only cross correlation
        for i in range(len(dataset1)):
            # Normalise the magnitude of the vectors without modifying the angle
            # This means the only differences between datapoints is the phase
            if np.abs(dataset1[i]) != 0:
                dataset1[i] = dataset1[i] / np.abs(dataset1[i])
            if np.abs(dataset2[i]) != 0:
                dataset2[i] = dataset2[i] / np.abs(dataset2[i])

    acorr1 = signal.correlate(dataset1,dataset1,mode="full") # auto correlation of dataset 1
    acorr2 = signal.correlate(dataset2,dataset2,mode="full") # auto correlation of dataset 2

    # Cross correlation:
    ccorr = signal.correlate(dataset1,dataset2,mode="full")

    # Normalise the Cross correlation
    ccorr = ccorr / np.sqrt(np.abs(max(np.abs(acorr1))) * np.abs(max(np.abs(acorr2))))

    return ccorr

def peak_value(dataset1,dataset2,mode="both"):
    cross_correlation = ncrossCorr(dataset1,dataset2,mode)
    #middleindex = int((len(cross_correlation) - 1)/2)
    #print(middleindex)
    #return np.abs(cross_correlation[middleindex])
    return max(np.abs(cross_correlation))


# Gathering Data ----------------------------------------------------------------------------------

def getDepths(runInfo):
```

```
    # Return the depths bins for a input sensor setup
    [client,config] = setupClient(runInfo)
    session_info = client.setup_session(config)

    if runInfo.sensorType == "A121":
        depths = et.utils.approx_r3_range_m(config)[0]
    else:
        depths = et.utils.get_range_depths(config, session_info)
    return depths

def getSamplingFreq(depths):
    # Calculate sample frequency:
    c = 299792458 # speed of light to a unecessary level of accuracy
    rangeBin = (depths[-1] - depths[0]) / len(depths)
    ts = (2*rangeBin)/c # time = distance/speed
    sr = 1/ts # sampling rate
    return sr

def getNewData(runInfo):

    [client,config] = setupClient(runInfo)

    client.connect()

    session_info = client.setup_session(config)
    #depths = et.utils.get_range_depths(config, session_info)
    #print("Session info:\n", session_info, "\n")
    client.start_session()
    dataset = [] # data to return
    for i in range(runInfo.samples):
        dataset.append(client.get_next()[1])

    client.stop_session()
    time.sleep(0.1)
    client.disconnect()

    # pause after getting data
    # Sometimes the pi is still connected afterwards. Pause to give it time to disconnect before reconnecting
    time.sleep(runInfo.datapointPause)
    return dataset

def setupClient(runInfo):

    client = et.SocketClient(runInfo.IP)

    if runInfo.sensorType == "A121":
        config = et.configs.A121SparseIQConfig()
        config.sweeps_per_frame = 1
        config.start_point = runInfo.A121_start_point
        config.step_length = 1
        config.num_points = runInfo.A121_num_points

        config.frame_rate = runInfo.update_rate
        config.receiver_gain  = runInfo.gain
        config.hwaas = runInfo.HWAAS
    else:
        config = et.IQServiceConfig()
        config.range_interval = runInfo.range
        config.asynchronous_measurement = False # slower, but prevents interference between sensors
        config.depth_lowpass_cutoff_ratio = runInfo.lowPassFilter
        config.noise_level_normalization = False # added 27/02/2022
        config.update_rate = runInfo.update_rate
        config.gain = runInfo.gain
        config.hw_accelerated_average_samples = runInfo.HWAAS

    config.sensor = runInfo.sensors

    config.profile = runInfo._profile



    return [client,config]

def LinearRun(listOfRuns):
    # Do a run and get data at points along the run

    # Metadata about the run is saved when this function is called - it is not requried to save this seperately

    # Will do separate runs in parallel. Therefore all runs must have the same:
    # - Lens
    # - Steps/Number of dataPoints/Start and end position
    # - Scene

    # Serial connection is one-way, so need to wait between sending signals
    arduino=serial.Serial(listOfRuns[0].serialPort, 9600)
    time.sleep(0.5)
    print("Arudino Connection Established")
    arduino.write("0".encode()) # write it to position 0 (should be current position)
    time.sleep(1)
    arduino.write(listOfRuns[0].stepmode.encode()) # Transmit desired microstep mode
    time.sleep(1)
    dataset = []

    # Save metadata
    for i in range(len(listOfRuns)): # loop through differnt runs
        # Create directory if it doesn't already exist
```

```
        touchDir(listOfRuns[i].directory)
        # Save setup
        #listOfRuns[i].saveInfo()
        #listOfRuns[i].savetxt()

        # Save depths
        depths = getDepths(listOfRuns[i])
        np.save("./" + listOfRuns[i].directory + "/depths",depths)

    for i in range(len(listOfRuns)):
        dataset.append([]) # add new empty list for data for each run

    for i in range(len(listOfRuns[0].steps)):
        # Loop through step positions
        arduino.write(str(listOfRuns[0].steps[i]).encode())
        time.sleep(listOfRuns[0].stepPause)
        print(str(i+1) + "/" + str(len(listOfRuns[0].steps)))

        for j in range(len(listOfRuns)):
            #remove later:
            print("Profile: " + str(listOfRuns[j]._profile) + " HWAAS: " + str(listOfRuns[j].HWAAS))
            # Runs loop
            data = getNewData(listOfRuns[j])
            """
            if len(listOfRuns[j]) == 1:
                data = [data]
            """
            dataset[j].append(data)

            runInfo = listOfRuns[j]

            # save data
            if runInfo._saveMode == "samples" or runInfo._saveMode == "both":
                touchDir("./" + runInfo.directory + "/data/samples/")

                # Save the full sampled data
                # modified DATASET is the list of samples (not HWAAS)
                if len(runInfo.sensors) == 1: # Store the data in a 1-item list if only using 1 sensor
                    if runInfo.sensorType == "A121":
                        modifiedAverage = average(dataset[j][i])
                        modifiedDataset = dataset[j][i]
                    else:
                        modifiedAverage = [average(dataset[j][i])]
                        modifiedDataset = [dataset[j][i]]
                else: # Data for multiple sensors is stored in a list by default
                    modifiedDataset = dataset[j][i]
                    modifiedAverage = average(dataset[j][i])

                np.save("./" + runInfo.directory + "/data/samples/samples datapoint " + str(i),
                        modifiedDataset) #dtype=object tells numpy that this is a complex data structure

            if runInfo._saveMode == "average" or runInfo._saveMode == "both":
                touchDir("./" + runInfo.directory + "/data/average/")
                np.save("./" + runInfo.directory + "/data/average/average datapoint " + str(i),
                        modifiedAverage) #dtype=object tells numpy that this is a complex data structure

    # Reset Position back to 0
    arduino.write(str(listOfRuns[0].steps[0]).encode())


# Loading and Saving Data ---------------------------------------------------------------------

def loadRunInfoOld(directory):
        # Load a previously saved run setup

        f = open(directory + "/run_info.p","rb")
        data = pickle.load(f)
        f.close()
        return data

def loadRunInfo(directory):
    s = linearMetadata(directory) # will initially set the wrong directory, this will be overwritten
    f = open(directory + "/run_info.hugo","r")
    for line in f.readlines():
        [key,value] = line.split(";")
        value = value[:-1] # remove newline char
        #print(str(key) + ", " + str(value))

        # Why doesn't python have a switch?!?
        if key == "lens":
            s.lens = value
        elif key == "directory":
            s.directory = value
        elif key == "IP":
            s.IP = value
        elif key == "update_rate":
            s.update_rate = int(value)
        elif key == "range":
            listValues = value[1:-1].split(",")
            if len(listValues) > 2:
                raise Exception("Unexpected error. Range list longer than 2")
            s.range = [float(listValues[0]),float(listValues[1])]
        elif key == "steps":
            listValues = value[1:-1].split(",")
            s.steps = []
            for listValue in listValues:
```

```
                s.steps.append(int(listValue))
        elif key == "HWAAS":
            s.HWAAS = int(value)
        elif key == "stepmode":
            s.stepmode = value
        elif key == "gain":
            s.gain = float(value)
        elif key == "serialPort":
            s.serialPort = value
        elif key == "stepFactor":
            s.stepFactor = float(value)
        elif key == "lowPassFilter":
            s.lowPassFilter = float(value)
        elif key == "samples":
            s.samples = int(value)
        elif key == "stepPause":
            s.stepPause = float(value)
        elif key == "sensors":
            s.sensors = []
            listValues = value[1:-1].split(",")
            for listValue in listValues:
                s.sensors.append(int(listValue))
        elif key == "_saveMode":
            s._saveMode = value
        elif key == "profile":
            profileValue = value.replace("Profile ","")
            #s.setProfile(int(profileValue))
            s.profile = str(int(profileValue))
        elif key == "_profile":
            # do nothing
            pass
        elif key == "sensorType":
            s.sensorType = value
        elif key == "A121_start_point":
            s.A121_start_point = value
        elif key == "A121_num_points":
            s.A121_num_points = value
        elif key == "datapointPause":
            s.datapointPause = value
        else:
            raise Exception("Unexpected metadata: " + str(key))
    f.close()
    return s

def loadRunsInfo(fileListDirectory,directoryPrefix="."):
    f = open(directoryPrefix + "/" + fileListDirectory + "/list_of_runs.txt","r")
    runsInfo = []

    for line in f.readlines():
        if directoryPrefix == ".":
            run = loadRunInfo("./" + line[:-1]) # remove the \n char
        else:
            run = loadRunInfo(directoryPrefix + "/" + line[:-1]) # remove the \n char
            run.directory = directoryPrefix + "/" + line[:-1] # set to new directory
        runsInfo.append(run)
    f.close

    return runsInfo

def saveListOfRuns(listOfRuns,directory):
    touchDir(directory)
    f = open("./" + directory + "/list_of_runs.txt","w")
    for run in listOfRuns:
        run.saveInfo()
        run.savetxt()
        f.write(run.directory)
        f.write("\n")
    f.close()


def loadDataFromListOfRuns(listOfRuns,mode="average"):
    data = []
    for run in listOfRuns:
        data.append(loadDataFromRun(run,mode))
    return data

def loadDataFromRun(runInfo,mode="average"):
    # Load data saved in the directory in runInfo
    # mode determines what data to load: samples, averages, or both
    # loading only averages is much faster, so that is recommended unless you really need samples

    class dataSet:
        def __init__(self):
            self.samples = []
            self.average = []

    data = [] # variable to return data in
    for i in range(len(runInfo.sensors)): # append item for each sensor data to load
        data.append(dataSet())

    if mode == "samples" or mode == "both":
        for i in range(len(runInfo.steps)):
            fileData = np.load("./" + runInfo.directory + "/data/samples/" + "samples datapoint " + str(i) + ".npy",allow_pickle=True)

            # need to unpack values if only using 1 sensor
            if len(runInfo.sensors) == 1 and runInfo.sensorType != "A121":
```

```
                fileData = fileData[0]
                samples = []

                #if runInfo.sensorType == "A121" and len(fileData) == 1: # new sensors store data differently. need to unpack data
                #    fileData = fileData[0]

                for k in range(runInfo.samples):
                    samples.append(fileData[k])
                data[0].samples.append(samples)
            else:
                for j in range(len(runInfo.sensors)):
                    samples = []
                    for k in range(runInfo.samples):
                        samples.append(fileData[k][j])
                    data[j].samples.append(samples) #data[j].samples.append(fileData[j])

    if mode == "average" or mode == "both":
        for i in range(len(runInfo.steps)):
            fileData = np.load("./" + runInfo.directory + "/data/average/" + "average datapoint " + str(i) + ".npy",allow_pickle=True)
            for j in range(len(runInfo.sensors)):
                data[j].average.append(fileData[j]) #data[j].samples.append(fileData[j])

    if runInfo._saveMode != mode:
        print("FYI: Data loading mode '" + str(mode) + "' does not match data collection mode '" + str(runInfo._saveMode) + "'")

    for i in range(len(runInfo.sensors)):
        data[i].length = max(len(data[i].samples),len(data[i].average))

        data[i].depths = np.load("./" + runInfo.directory + "/depths.npy",allow_pickle=True)
    return data


def loadMatlabData(filepath):
    # Load esitmated correlation profile from a matlab file


    fileData = scio.loadmat(filepath)# import the expected profile variation
    tempDis = fileData["surgeVec"][0] # unpack data
    tempDis = list(tempDis) # unpack data

    # Each item in the array is stored in a 1-element array
    # Need to unpack these values
    expectedDis = []
    expectedRho = []
    for i in range(len(fileData["rho"])):
        if not(np.isnan(fileData["rho"][i][0])):
            expectedRho.append(fileData["rho"][i][0]) # 0th item in 1-element array
            expectedDis.append(tempDis[i])

    # The data from matlab is single-sided
    # Make it double sided
    doubleSideRho = []
    doubleSideDis = []

    for i in range(len(expectedRho) - 1):
        doubleSideRho.insert(0,expectedRho[i+1])
        doubleSideDis.insert(0,-expectedDis[i+1])

    expectedRho = np.array(doubleSideRho + expectedRho)
    expectedDis = np.array(doubleSideDis + expectedDis)

    return [expectedDis, expectedRho]

# Misc ------------------------------------------------------------------------------------

def touchDir(directory):
    # directory is a (relative) directory path using "/" as seperators
    # Will create a directory path if it doesn't already exist
    folderNames = directory.split("/") # get name of each folder

    # Remove first character if it references the current directory
    if folderNames[0] == ".":
        folderNames.pop(0)

    currentDirectory = "./"
    for folderName in folderNames:
        if not(os.path.isdir(currentDirectory + folderName + "/")):
            os.mkdir(currentDirectory + folderName + "/")
        currentDirectory = currentDirectory + folderName + "/"


# Data Analysis Functions  ----------------------------------------------------------------------

def compareSignals(dataset1, dataset2, depths, saveDir = "",filename="",pltLabels = ["",""],title="",correlationMode="normal"):
    autoCorrelation1 = ncrossCorr(dataset1,dataset1,mode=correlationMode)
    autoCorrelation2 = ncrossCorr(dataset2,dataset2,mode=correlationMode)
    crossCorrelation = ncrossCorr(dataset1,dataset2,mode=correlationMode)

    fig, axs = plt.subplots(5,1,figsize=(20,25))
    fig.subplots_adjust(hspace=0.45) # give more hspace between plots to make axes labels visible


    fig.suptitle(title)
    axs[0].plot(depths,np.abs(dataset1), label=pltLabels[0])
    axs[0].plot(depths,np.abs(dataset2), label=pltLabels[1])
    axs[0].set_xlabel("Distance (m)")
```

```
    axs[0].set_ylabel("Amplitude")
    axs[0].set_title("Amplitude vs Distance")
    axs[0].grid(visible=True)
    axs[0].legend()

    axs[1].plot(depths,np.angle(dataset1), label=pltLabels[0])
    axs[1].plot(depths,np.angle(dataset2), label=pltLabels[1])
    axs[1].set_xlabel("Distance (m)")
    axs[1].set_ylabel("Phase")
    axs[1].set_title("Phase vs Distance")
    axs[1].grid(visible=True)
    axs[1].legend()

    distance = depths[-1] - depths[0]
    tmp1 = np.linspace(-distance,distance,len(autoCorrelation1)) # temporary x vector

    axs[2].plot(tmp1,np.abs(autoCorrelation1), label=pltLabels[0] + " AutoCorrelation")
    axs[2].set_xlabel("lag (m)")
    axs[2].set_ylabel("Correlation Magnitude")
    axs[2].set_title("Datum AutoCorrelation")
    axs[2].grid(visible=True)
    axs[2].legend()


    axs[3].plot(tmp1,np.abs(autoCorrelation2), label=pltLabels[1] + "AutoCorrelation")
    axs[3].set_xlabel("Distance (m)")
    axs[3].set_ylabel("Correlation Magnitude")
    axs[3].set_title("Sample AutoCorrelation")
    axs[3].grid(visible=True)

    axs[4].plot(tmp1,np.abs(crossCorrelation), label="Cross Correlation")
    axs[4].set_xlabel("Distance (m)")
    axs[4].set_ylabel("Correlation Magnitude")
    axs[4].set_title("Datum-Sample Cross Correlation. Max: " + str(max(np.abs(crossCorrelation))))
    axs[4].grid(visible=True)
    axs[4].set_ylim(0,1)

    if saveDir != "":
        touchDir(saveDir + "/saveDataPointFigures/")
        plt.savefig(saveDir + "/saveDataPointFigures/" + filename + ".png",dpi=300)
        plt.close("all")



def saveDataPointFigures(runInfo,data,depths,datum):
    # For a given dataset, this will save a figure for each averaged datapoint
    # data = list of AVERAGED datapoint values. Each datapoint should have a single vector in it
    # Datum = usually just the first item in the data eg: data[0], but could be anything (like the centre point)

    print("Saving figures for each datapoint, this may take a while...")


    # loop through data points
    for i in range(len(data)): # dataPOINT loop
        print("Datapoint " + str(i+1) + "/" + str(len(data)))

        autoCorrelation1 = ncrossCorr(datum,datum)
        autoCorrelation2 = ncrossCorr(data[i],data[i])
        #crossCorrelation = correlationVector
        crossCorrelation = ncrossCorr(datum,data[i])

        fig1, axs = plt.subplots(5,1,figsize=(20,25))
        fig1.subplots_adjust(hspace=0.45) # give more hspace between plots to make axes labels visible
        fig1.suptitle("Distance: " + str(runInfo.alongTrackDirection()[i] * runInfo.stepFactor * (1/runInfo.stepModeFactor())) \
                      + "mm")
        axs[0].plot(depths,np.abs(datum), label="Datum (average)")
        axs[0].plot(depths,np.abs(data[i]), label="Sample (average)")
        axs[0].set_xlabel("Distance (m)")
        axs[0].set_ylabel("Amplitude")
        axs[0].set_title("Amplitude vs Distance")
        axs[0].grid(visible=True)
        axs[0].legend()

        axs[1].plot(depths,np.angle(datum), label="Datum (average)")
        axs[1].plot(depths,np.angle(data[i]), label="Sample (average)")
        axs[1].set_xlabel("Distance (m)")
        axs[1].set_ylabel("Phase")
        axs[1].set_title("Phase vs Distance")
        axs[1].grid(visible=True)
        axs[1].legend()

        tmp1 = np.linspace(-1,1,len(autoCorrelation1)) # temporary x vector

        axs[2].plot(tmp1,np.abs(autoCorrelation1), label="Datum AutoCorrelation")
        axs[2].set_xlabel("Distance (m)")
        axs[2].set_ylabel("Correlation Magnitude")
        axs[2].set_title("Datum AutoCorrelation")
        axs[2].grid(visible=True)
        axs[2].legend()


        axs[3].plot(tmp1,np.abs(autoCorrelation2), label="Sample AutoCorrelation")
        axs[3].set_xlabel("Distance (m)")
        axs[3].set_ylabel("Correlation Magnitude")
        axs[3].set_title("Sample AutoCorrelation")
        axs[3].grid(visible=True)
```

```
        axs[4].plot(tmp1,np.abs(crossCorrelation), label="Datum-Sample Cross Correlation")
        axs[4].set_xlabel("Distance (m)")
        axs[4].set_ylabel("Correlation Magnitude")
        axs[4].set_title("Datum-Sample Cross Correlation. Max: " + str(max(np.abs(crossCorrelation))))
        axs[4].grid(visible=True)
        axs[4].set_ylim(0,1)

        touchDir(runInfo.directory + "/saveDataPointFigures/")
        plt.savefig(runInfo.directory + "/saveDataPointFigures/" + str(i) + ".png",dpi=300)
        plt.close("all")


def saveDataSampleFigures(runInfo,data,depths,datum):
    # HIGHLY UNRECOMMENDED

    # Generates a plot for each data sample
    # Will take ages to run
    # Will create gigabytes worth of figures

    touchDir(runInfo.directory + "/saveDataPointFigures/")

    for i in range(len(data)):
        touchDir(runInfo.directory + "/saveDataPointFigures/" + str(i) + "/")
        for j in range(len(data[i])):
            acorr1 = ncrossCorr(datum,datum)
            ccorr = ncrossCorr(datum,data[i][j])
            acorr2 = ncrossCorr(data[i][j],data[i][j])

            fig0, axs = plt.subplots(3,2,figsize=(20,14))
            fig0.suptitle("Distance: " + str(runInfo.alongTrackDirection()[i] * runInfo.stepFactor * (1/runInfo.stepModeFactor())) \
                          + "mm")

            tmp1 = np.linspace(-3.14,3.14,len(acorr1))
            axs[0,0].plot(depths,np.abs(datum),color= 'red')
            axs[0,0].set_title("Datum Signal")
            axs[0,0].set_xlabel("Distance")
            axs[0,0].set_ylabel("Amplitude",color= 'red')
            axs[0,0].tick_params(axis ='y', labelcolor = 'red')

            tmpaxs = axs[0,0].twinx()
            tmpaxs.plot(depths,np.angle(datum),color= 'blue')
            tmpaxs.set_ylabel("Phase",color= 'blue')
            tmpaxs.tick_params(axis ='y', labelcolor = 'blue')


            axs[0,1].plot(depths,np.abs(data[i][j]),color= 'red')
            axs[0,1].set_title("Current Signal")
            axs[0,1].set_xlabel("Distance")
            axs[0,1].set_ylabel("Amplitude",color= 'red')
            axs[0,1].tick_params(axis ='y', labelcolor = 'red')

            tmpaxs = axs[0,1].twinx()
            tmpaxs.plot(depths,np.angle(data[i][j]),color= 'blue')
            tmpaxs.set_ylabel("Phase",color= 'blue')
            tmpaxs.tick_params(axis ='y', labelcolor = 'blue')

            axs[1,0].plot(tmp1,np.abs(acorr1))
            axs[1,0].set_title("Autocorrelation of Datum (Amplitude)")

            axs[1,1].plot(tmp1,np.abs(acorr2))
            axs[1,1].set_title("Autocorrelation of Current (Amplitude)")

            axs[2,1].plot(tmp1,np.abs(ccorr))
            axs[2,1].set_title("Cross Correlation (Amplitude): Peak Value: " + str(np.abs(max(np.abs(ccorr)))))
            axs[2,1].set_ylim(0,1)

            plt.savefig(runInfo.directory + "/saveDataPointFigures/" + str(i) + "/" + str(j) + ".png")
            plt.close("all")
```

9.6.2 Simulations

```
# Simulation code for my fyp
# Hugo Weston 2022 hw876@bath.ac.uk

import scipy.io as scio # used to load matlab files

import numpy as np

import os

# class to store a simulated sensor array
class simSensorArray ():
    def __init__(self,sensor_separations):
        self.sensors = []
        for sensor_separation in sensor_separations:
            self.sensors.append(simSensor(sensor_separation))
        #self.estimate_mode = ""
        #self.correlation_mode = ""



    def inter_correlation(self,distance_vector, base_sensor, target_sensor, target_sensor_position, mode="ideal"):
        # position is how far the target sensor has moved!
        if mode == "ideal":
            int_correlation = self.sensors[base_sensor].ideal_correlation(distance_vector, target_sensor_position + self.sensors[target_sensor].separation)
```

```
        elif mode == "real":
            int_correlation = self.sensors[base_sensor].real_correlation(distance_vector, target_sensor_position + self.sensors[target_sensor].separation)
        elif mode == "estimated":
            int_correlation = self.sensors[base_sensor].estimated_correlation(distance_vector, target_sensor_position + self.sensors[target_sensor].separation)
        else:
            raise Exception("Incorrect mode: " + str(mode))
        return int_correlation

    def inter_displacement_estimate(self,distance_vector,time_vector,base_sensor, target_sensor, target_sensor_position, cor_mode="ideal",est_mode="default"):

        int_correlation = self.inter_correlation(distance_vector, base_sensor, target_sensor, target_sensor_position,mode=cor_mode)
        [estTime,estDist] = self.sensors[base_sensor].estimate_displacement(int_correlation,time_vector,vals=est_mode) #gauss_parameters_mitigated


        distance_between_sensors = self.sensors[target_sensor].separation - self.sensors[base_sensor].separation
        estDist = estDist + distance_between_sensors + target_sensor_position
        return [estTime,estDist]

    def inter_distance_estimate(self,distance_vector,time_vector,base_sensor, target_sensor, target_sensor_position):
        [estTime,estDist] = self.inter_displacement_estimate(distance_vector,time_vector,base_sensor, target_sensor, target_sensor_position)
        estDist = estDist + self.sensors[base_sensor].separation
        return [estTime,estDist]






# class to store a simulated sensor
class simSensor():
    def __init__(self,sensor_separation):
        self.separation = sensor_separation
        self.estimatedDistance = []
        self.estimatedTime = []

    # These are all ways of getting a correlation profile
    def ideal_correlation(self,distanceVector,datum): # datum is a distance
        idealCorrelaion = gauss(distanceVector - datum + self.separation)
        #unused (for now)
        return idealCorrelaion

    def real_correlation(self,distanceVector,datum):
        idealCorrelaion = self.ideal_correlation(distanceVector,datum)
        #noise = 0.05 * np.random.normal(size=(len(distanceVector)))
        noise = 0.01 * np.random.normal(size=(len(distanceVector)))
        realCorrelation = 0.909090909 * 0.75 * 0.9 * idealCorrelaion + noise + 0.1
        return realCorrelation

    def custom_correlation():
        return

    def estimated_correlation(self,distanceVector,datum,vals="gauss_parameters_mitigated"):
        return gauss(distanceVector - datum + self.separation,vals=vals)


    # Estimating displacement/distance DOES NOT TAKE INTO ACOUNT THE DATUM POSITION
    # In your code you MUST REMEMBER TO ADD THE DATUM BACK ON
    def estimate_displacement(self,correlation,timeVector,vals="default",minCorrelation=0.5):
        # Alternatively use "gauss_parameters_mitigated"


        estimated_distance_1 = np.array([]) # store low estimate values
        estimated_distance_2 = np.array([]) # store high estimate values
        estimatedTime = np.array([])
        #estimated_time = np.zeros(len(correlation)) # time vector corresponding to estimated distances

        for i in range(len(correlation)):
            if correlation[i] > minCorrelation:
                estimated_distance_1 = np.append(estimated_distance_1, invGauss(correlation[i],vals=vals)[0])
                estimated_distance_2 = np.append(estimated_distance_2, invGauss(correlation[i],vals=vals)[1])
                estimatedTime = np.append(estimatedTime, timeVector[i])
        return [np.concatenate((estimatedTime, estimatedTime)), np.concatenate((estimated_distance_1, estimated_distance_2))]

    def estimate_distance(self, correlation, timeVector, vals="default"):
        [estTime, estDist] = self.estimate_displacement(correlation, timeVector,vals=vals)
        estDist = estDist + self.separation
        return [estTime, estDist]



def getRsquared(data, fittedCurve):
    # Find r squared (coefficient of determination) between two datasets
    SSR = 0 # sum of squares regression
    SST = 0 # sum of squares total
    mean = np.mean(data)
    for i in range(len(data)):

        SSR = SSR + (data[i] - fittedCurve[i])**2
        SST = SST + (data[i] - mean)**2

    R_squared = 1 - SSR/SST # definition of r squared
    return R_squared

# Cubic Distribution (from Ben)
def cubicDistro(x,mu_0,k,L):
```

```
    # This function is very sensitive to inputs and scipy will no be able to optimise it without good starting parameters
    # Start curve fitting with the following parameters:
    # mu_0 = 200000
    # k = 1
    # L = -0.1
    return mu_0 * (-2 * np.abs(x - 2*k*L)**3 + np.abs(x + (2-2*k)*L)**3 + \
                   np.abs(x + (-2-2*k)*L)**3 - 2*np.abs(x+2*L)**3 + 4 * np.abs(x)**3 - \
                   2 * np.abs(x-2*L)**3 + np.abs(x + (2 + 2*k)*L)**3 + np.abs(x + (-2+2*k)*L)**3 - \
                   2 * np.abs(x + 2*k*L)**3)

# gauss distribution
def gaussDistro(x,amplitude,mean,sigma):
    return amplitude * np.exp(-(x - mean)**2/(2*sigma**2))

def quadratic(x,a,b,c):
    return a * x**2 + b * x + c

def gauss(xcor, vals="default"):
    # get parameters

    params = loadFunctionParameters(vals,defaultValue="I:/My Drive/my-emacs/University of Bath/Year 4/ME40321/Code/simulation/gauss_parameters.npy")

    amplitude = params[0]
    mean = params[1]
    sigma = params[2]

    return gaussDistro(xcor,amplitude,mean,sigma)

def sigFig(x, sig, returnType="string"):
    # Round to sig number of significant figures
    # if a string output is desired, 0's will be added to the end to show the sig figs
    output = round(x, sig-int(np.floor(np.log10(abs(x))))-1)

    if returnType == "string":
        stringOutput = str(output)
        while len(stringOutput.replace(".","")) < sig:
            stringOutput = stringOutput + "0"
        return stringOutput
    elif returnType == "float":
        return output
    else:
        raise Exception("Incorrect returnType mode '" + str(returnType) +"'")

def invGauss(ycor,vals="default",):
    # Inverse of the gauss function

    # vals is either a sttring ("default") or a list of parameters for the gaussian function

    # get parameters
    params = loadFunctionParameters(vals,defaultValue="I:/My Drive/my-emacs/University of Bath/Year 4/ME40321/Code/simulation/gauss_parameters.npy")

    amplitude = params[0]
    zeta = params[1]
    sigma = params[2]


    underRoot = 2*sigma**2*(np.log(amplitude) - np.log(ycor))
    if underRoot < 0:
        #print("Warning, approximating -ve square root as 0")
        underRoot = 0

    x1 = zeta - np.sqrt(underRoot)
    x2 = zeta + np.sqrt(underRoot)
    return [x1,x2]

def loadFunctionParameters(inputValue,defaultValue=""):
    if defaultValue == "":
        raise Exception("Default value not defined. Default value must be defined")

    if type(inputValue) == str:
        if inputValue == "default":
            params = np.load(defaultValue)
        else:
            filepath = "I:/My Drive/my-emacs/University of Bath/Year 4/ME40321/Code/simulation/" + str(inputValue) + ".npy"
            if os.path.exists(filepath) == False:
                raise Exception("Cannot load gauss paramters. File: " + str(filepath) + " doees not exist")
            params = np.load(filepath)

    else:
        params = inputValue
    return params

"""
# not needed anymore
def inter_element_distance_estimate(sensor_1, sensor_2,distanceVector, timeVector, datumIndex, correlationFunction,vals="default"):

    # CorrelationFunction is the function useed to generate the points entered into the distance estimation function
    # Vals is the gaussian approximation values to use to estimate distance

    inter_correlation = inter_element_correlation(sensor_1, sensor_2,distanceVector,datumIndex,correlationFunction)

    distancetonextsensor = sensor_2.separation - sensor_1.separation
    [estTime, estDist] = sensor_1.estimate_displacement(inter_correlation, timeVector,vals=vals)
    estDist = estDist + distancetonextsensor
    return [estTime, estDist]
"""
```

9.6.3 Mapping

```
# Mapping code for my fyp
# Hugo Weston 2022 hw876@bath.ac.uk

import numpy as np
import copy

def normaliseArray(arr):
    #normalise 2D matrix
    #Also convert it to only amplitude

    #get max value:
    currentMax = np.abs(arr[0][0])
    for x in range(len(arr)):
        for y in range(len(arr[0])):
            if np.abs(arr[x][y]) > currentMax:
                currentMax = np.abs(arr[x][y])

    if currentMax == 0:
        raise Exception("Found that the maximum value in the array is 0. Cannot normalise array")

    for x in range(len(arr)):
        for y in range(len(arr[0])):
            arr[x][y] = np.abs(arr[x][y]) / currentMax
    return arr

def averageSignalArray(signalArray,countArray):
    for y in range(len(signalArray)):
        for x in range(len(signalArray[0])):
            #print("x = " + str(x) + " y = " + str(y))
            if countArray[y][x] > 0: # Don't mess with anything if no signal is present in this pixel
             signalArray[y][x] = signalArray[y][x] / countArray[y][x]
    return signalArray


def create_canvas(sizex,sizey,canvasType="signal"):
    # canvasType = signal returns a 2D array of zeros
    # canvasType = image returns a 2D array of [0.0, 0.0, 0.0] lists (rgb)

    canvas = []
    for i in range(sizey):
        canvas.append([])
        for j in range(sizex):
            if canvasType == "signal":
                canvas[i].append(0.0)
            elif canvasType == "image":
                canvas[i].append([0.0,0.0,0.0]) # must be float
            else:
                raise Exception("Incorrect canvas type selection: '" + str(canvasType) + "'")
    if canvasType == "signal":
        return np.array(canvas,dtype="cfloat")
    elif canvasType == "image":
        return canvas
    else:
        raise Exception("Incorrect canvas type selection: '" + str(canvasType) + "'")


def indexMapSwath(signalLength,halfSwathWidthDeg):
    # This function maps the datapoints from the 1-D signal array the 2-D image array

    # Half Swath Width is half the angle of the projected signal
    # Signal length is the length of the signal vector
    halfSwathWidthRad = halfSwathWidthDeg * np.pi/180

    arrayHeight = signalLength
    arrayWidth = int(np.ceil(2 * signalLength * np.sin(halfSwathWidthRad)))

    # need the width to the odd
    if arrayWidth % 2 == 0:
        arrayWidth += 1


    imageArray = create_canvas(arrayWidth,arrayHeight,canvasType="signal")

    sensorXPos = int(np.ceil(arrayWidth/2))

    class signalMap():
        def __init__(self,arrayWidth,arrayHeight):
            self.targetX = []
            self.targetY = []
            self.signal = []
            self.arrayWidth = arrayWidth
            self.arrayHeight = arrayHeight
            self.sensorXPos = sensorXPos
            self.weightings_basic = []

    output_signalMap = signalMap(arrayWidth,arrayHeight)

    # iterate through array
    for x in range(imageArray.shape[1]):
        for y in range(imageArray.shape[0]):
            xDist = sensorXPos - x
            yDist = y


            verticalAngle = np.arctan2(xDist,yDist) # angle between vector and vertical
```

```
            # Check the angle is within requested
            if np.abs(verticalAngle) <= halfSwathWidthRad:
                r = (xDist**2 + yDist**2)**0.5 # pythagorean theorem
                # r is the distance in samples between [x,y] and the array position


                # Check that you can actually populate the data
                if int(r) < signalLength:
                    output_signalMap.targetX.append(x)
                    output_signalMap.targetY.append(y)
                    output_signalMap.signal.append(int(r)) # round to the nearest whole number
                    output_signalMap.weightings_basic.append(np.exp(-(5*verticalAngle)**2))

    output_signalMap.targetX = np.array(output_signalMap.targetX)
    output_signalMap.targetY = np.array(output_signalMap.targetY)
    return output_signalMap

def processSARImage(run,data,userResolution,halfSwathWidth,filterVal=0):
    # Calculate Resolutions:
    alongResolution = (run.alongTrackDirection()[-1] - run.alongTrackDirection()[0]) / len(run.alongTrackDirection()) # mm per sample taken

    #acrossResolution = 1000 * (run.range[1] - run.range[0]) / len(data.depths) # mm per sample (across track)

    signalLength_mm = data.depths[-1] * 1000 # max depths (mm)
    signalLength_sa = int(signalLength_mm /userResolution) # max depths (sampes)

    extraWidth = np.sin(halfSwathWidth * np.pi/180) * signalLength_sa # this is the width caused by the swath expanding outwards

    # + 10 as a tolerance as the indexing may go to above the width
    imageWidth = int(len(data.average) * alongResolution * (1/userResolution)  + 2*extraWidth + 10)   # convert to have the same sample resolution as the range


    signalArray = create_canvas(imageWidth,signalLength_sa,canvasType="signal")
    countArray = create_canvas(imageWidth, signalLength_sa,canvasType="signal")
    # Low Pass Filter data
    c = 299792458
    D = (data.depths[-1] - data.depths[0])/len(data.depths)
    d = 2*D/c

    print("Creating Image Index Map")
    indexMap = indexMapSwath(signalLength_sa,halfSwathWidth) # map signal to 2D array

    print("Creating Signal Index Map")
    pointFrom = []
    pointTo = []
    for i in range(signalLength_sa):
        #print(str(i))
        pos = i * userResolution / 1000
        for j in range(len(data.depths) - 1):
            if data.depths[j] < pos and data.depths[j+1] >= pos:
                pointFrom.append(j)
                pointTo.append(i)

    for i in range(len(data.average)):
        print(str(i+1) + "/" + str(len(data.average))) # Progress Bar

        signal = data.average[i]

        # Change comment to add/remove filtering
        #signal = fyp.butter_lowpass_filter(imageData.average[i],10e9,1/d)
        # remove close signal data
        #signal[0:250] = 0

        modifiedSignal = np.zeros(signalLength_sa,dtype="cfloat")
        modifiedSignal[pointTo] = signal[pointFrom] # insert signal

        #modifiedSignal = np.abs(modifiedSignal) # Amplitude only map (not SAR)

        xOffset = int(run.alongTrackDirection()[i]/userResolution)
        signalArray[indexMap.targetY,indexMap.targetX + xOffset] += modifiedSignal[indexMap.signal] * indexMap.weightings_basic
        countArray[indexMap.targetY,indexMap.targetX + xOffset] += 1

    croppedArray = copy.deepcopy(signalArray[:,int(extraWidth):imageWidth-int(extraWidth)])
    return croppedArray
```