The University of York Department of Computer Science

Submitted in part fulfilment for the degree of BEng in Computer Science.

A System to Reduce the Effects of Crosstalk in Ultrasonic Sensor Data Using Neural Networks

Jack Knott

27th April 2015

Supervisor: Dr. Mike Freeman

Number of words = 14491, as counted by TeXcount.
This includes the body of the report only.

Abstract

This project looks at investigating a variety of methods to reduce noise in the application of ultrasonic sensors in autonomous robot environments. In systems with multiple ultrasonic transmitters, co-channel noise, or crosstalk, can often result in inaccurate sensor readings and hinder the overall performance of the system. This project explores methods of encoding the signals transmitted by each transmitter, before subsequently decoding the messages once received so that the origin of the signal can be determined. A novel system is developed which coded signals are generated, corresponding to one code per transmitter. These signals are then intelligently decoded with the aid of classification techniques enabled by an artificial neural network. The results indicate that the system can correctly identify 90% of the codes, even in the presence of various types of noise. However, the system does not perform well when subject to observations containing multiple signals. The project does not involve building an autonomous robot; the work is intended to build on existing techniques to develop a system that could be adapted to be used as part of an autonomous robotic system.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

Ultrasonic sensors are used in a host of range-finding applications. However, the signal that the sensors provide is often subject to noise and interference, creating difficulties especially when it is crucial for this data to be accurate. This report aims to investigate methods of improving ultrasonic sensor data reliability.

Autonomous robots need to collect data from their surroundings using sensors before interpreting this data in order to be autonomous. If the data collected from these sensors is not accurate or subject to noise, the system may not perform correctly, or may not function at all. A robot will often utilise multiple sensors in order to gather more information about its surroundings. A common form of noise found in multi-sensor robots is crosstalk, and this project aims to investigate methods of reducing the impact crosstalk has on the performance of an ultrasonic robot. Based on this research, a system will then be developed to try and combat the detrimental effects of crosstalk in such robots.

Ultrasonic sensors have been chosen over alternatives due to the fact that they are cheap, powerful and versatile. The range-finding applications of ultrasonic sensors enables them to be used in almost any autonomous robot that needs to know where it is with precision. Various features of ultrasonic sensors will be explored in detail, as well as some of the issues associated with them, and potential solutions to those issues.

1.2 Project Objectives

The aim of this project is to develop a system that can increase the reliability of ultrasonic sensor data where noise, specifically crosstalk, is present. The specific objectives that have been identified are as follows:

- to research the field of autonomous robotics and investigate various methods of improving ultrasonic data reliability
- to apply the results of the investigation to develop a system able to extract information from ultrasonic signals and process this information in order to reduce uncertainties in the signals
- to evaluate the developed system through a series of tests to determine whether it functions as intended and is a practical, viable solution

1.3 Ethical Statement

This project does not involve human participants. As a result, the project contains no physical risk to any persons and no confidential information is collected. As the project makes use of electronic equipment, there are some safety issues that must be considered, but all lab work will be performed during lab hours during which members of staff will be present and able to supervise.

1.4 Report Structure

The remaining sections of this report are structured as follows:

- **Chapter 2** investigates the background information required to understand the concepts involved in the project. An introduction is provided to many topics in computer science and electronics that may be exploited during the course of the project.
- **Chapter 3** outlines the problem that this project attempts to solve in detail, referring back to the literature review where relevant. It also specifies the project requirements and tests that will be used to evaluate the system.
- **Chapter 4** introduces the proposed design of the system. The system's implementation is discussed, various design options are considered and the design choices made during the proccess are explained and justified.
- **Chapter 5** analyses the performance of the system through a series of tests, and evaluates the system against the initial system requirements.
- **Chapter 6** summarises the work covered in the project, before outlining how the work could be used in future projects.

2 Literature Review

2.1 Autonomous Robotics

An autonomous robot is a robot that performs tasks and operates by itself, without the need for human instructions. Autonomous robots are often used for a variety of tasks in which they navigate their way around a location. It therefore must have the ability to sense its surroundings and the environment in which it is present in, as well as the capability to detect and avoid obstacles. This can be achieved with the use of sensors, data capture, processing that data and consequently controlling the movement of the robot. Figure 2.1 shows how an autonomous robot might function.

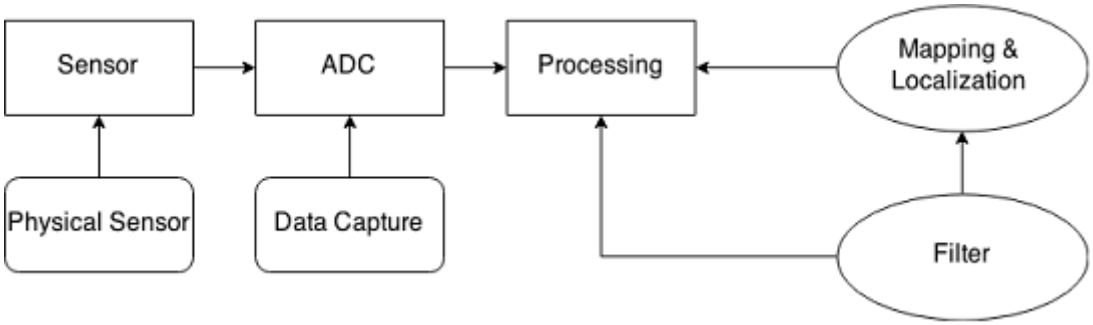


Figure 2.1: An example autonomous robot system overview

2.1.1 Mapping and Localization

A robot can receive sensor data containing information about where objects are in relation to itself, but it must also interpret this data in a way that allows it to know where it is and where it needs to be going in relation. It needs some way of representing its surroundings in a format that it can understand. There are different approaches that look to solve this problem, known as map representation. The metric framework considers a two-dimensional space and locates the robot in terms of co-ordinates. An example of this is the global positioning system (GPS), a commonly-used satellite navigation system designed by the US military that became fully operational in 1995 [1]. In contrast, the topological framework does not consider a predefined space in which to locate the robot, only the local distances between objects, which can be used to construct a graph, where the objects are nodes and the distances are edges.

The two aforementioned mapping frameworks are built around how the robot accesses the information which it uses to understand its position, receiving data from both ideothetic and allothetic sources. Whereas ideothetic sources are used to determine the exact location of the robot, including many dead reckoning techniques, allothetic sources make use of objects other than the robot such as walls; the allothetic sources in the case of a robot are the sensors attached. Both types of source have associated issues. The primary issue with idiothetic sources is cumulative error. For example, if a robot thinks it is travelling at 10 centimetres per second, but is actually travelling at 9 centimetres per second, every second that passes means the robot's estimated location is a centimetre away from its actual location. The primary issue with allothetic sources is what is known as perceptual aliasing: the idea that two separate locations

can be perceived as the same. If a robot arrives at a location and receives the same data from its sensors that it received at another location, it may process that data and come to the conclusion that it is at the first location. Chrisman offers a potential solution to the perceptual aliasing problem [2] which utilises reinforcement learning techniques. In this method, a predictive model of the environment is created, and probabilistic techniques are used to estimate the likelihood of a current location based on the state of the model. Reinforcement learning is then used to teach the system which control action to take in each possible state.

2.1.2 Sensors

In order to understand what is happening in the world around it, an autonomous robot needs sensors to capture data about its surroundings. There are many types of sensors that can be used to calculate the distance to objects in the environment, each of them collecting different types of data. Four types of sensor will be investigated in this section: cameras, lasers, infrared sensors and ultrasonic sensors.

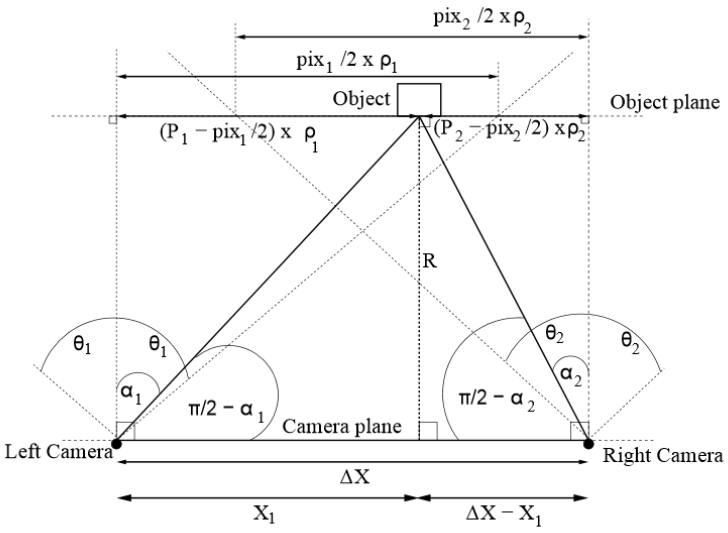
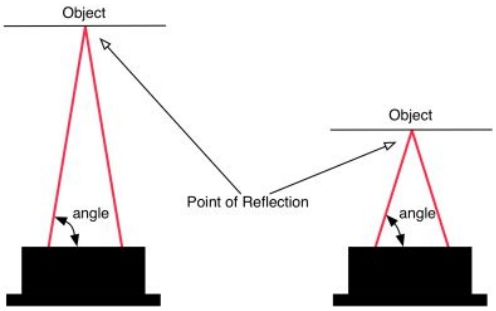


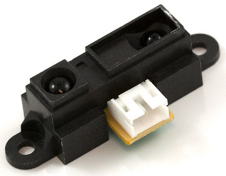
Figure 2.2: Geometry of Stereoscopic Range-finding [3]

Cameras

Cameras can be effective tools for identifying objects and using in range-finding applications. In stereoscopic range-finders, two cameras are positioned to view the same object and the two resulting images can be compared in order to determine the distance of the object. This is achieved by calculating the measurements on the stereoscopic image of the object, measuring the angular disposition of the two cameras and using the distance between the two cameras, as well as the extra technical information such as the focal lengths of the cameras, in order to estimate the distance to the object [4]. However, extracting this information from the images requires a great deal of processing, and in the case of an autonomous robot, this could hinder the performance of the system and result in slow movement. One proposed method [5] attempts to offer a stereoscopic localisation system that does not require reconstructing a complete geometric model of the environment, in order to reduce the amount of processing that needs to be performed.



(a) Infrared range estimation method [6]



(b) A Sharp infrared proximity sensor [7]

Figure 2.3: Infrared Sensors

Infrared

Infrared sensors are a cheap way to calculate distances to objects and consist of an LED light and a detector. A beam of infrared light is emitted from the LED. When this infrared light collides with an object and is reflected back, the signal is output by the detector. There are two categories of infrared sensors: range finders and proximity sensors. Proximity sensors work via a very simple principle and make use of digital detectors [6]. A constant beam is output with a limited range, and if any reflected light is detected within this range, a binary value of 1 is output by the sensor. However, with range finders, the angle at which the reflected beam returns is dependent on the distance of the object. Range finders output an analog signal that represents the relationship between the distance and the angle. The distance can be calculated using the angle of the reflected beam, a triangulation technique, as shown with a diagram in Figure 2.3a. However the relation between the angles at different distances is not linear so a linearisation equation must be used to convert the output to a distance value.

One of the most commonly used infrared proximity sensors is the Sharp GP2Y0A21YK0F [7]. It has a range of between 10 and 80cm and can be purchased for approximately £10. Although usually cheaper to implement than some ultrasonic sensors, they tend to be less accurate. One of the issues associated with using infrared sensors is that they can cause problems if used outside in direct sunlight, or exposed to light from a tungsten lamp, as the infrared light produced by these sources can contribute to incorrect readings [7]. It can also sometimes be difficult to detect objects that are dark in colour as dark objects absorb more infrared light than lighter ones. Infrared sensors have a much shorter range on average than ultrasonic sensors. An advantage to infrared sensors is that they can work well even if the angle the object is facing is not directly back at the sensor.

Infrared sensors are widely used in autonomous robotics for obstacle detection and range estimation. However, because of their dependence on the reflectance of surrounding objects, range estimation measurements can often be imprecise. As a result, maps made from measurements taken with these sensors are often inaccurate, and in many cases it is much more appropriate to use proximity detectors [8]. It may be because of this, that many autonomous robot systems utilise a combination of ultrasonic sensors and infrared proximity sensors [9, 10]; seldom are infrared sensors used exclusively in such systems. By combining the two, the readings of each sensor type can used together in what is known as sensor fusion [11]. Sensor fusion enables more information to a system, which can in turn give a more accurate representation of the environment.

Lasers

Lasers can be used to identify the distance of objects and operate in a similar principle to ultrasonic sensors, but instead of transmitting sound waves, narrow beams of light are transmitted and the time taken for the beam to be reflected back to the sender can determine the distance of the object. Light travels at approximately $3 \times 10^8 ms^{-1}$ and as a result, range-finding systems that utilise lasers can estimate distances much faster than ultrasonic sensors. The maximum range of these systems can also be much larger than other sensor types, exceeding 1km in some cases, because the beam of a laser is so narrow and not conoidal like some sensor types. However, lasers can be very expensive and the benefits of implementing a localisation system that uses lasers can be outweighed by the fact that there are other sensor types available that perform nearly as well for a fraction of the cost, and laser systems may be overkill for small-scale applications. Laser range finding does have its place in autonomous robotics, and have uses in outdoor scenarios, where infrared sensors may not work optimally, as demonstrated by Guivant et. al [12].

There are multiple techniques for estimating distances using lasers. A common method for achieving this is known as Lidar. Lidar, a word created by combining the words 'light' and 'radar', as described in the first published mention of the word [13], works by shining an object with a laser and analysing the reflected light. However, there are different techniques to analyse this light. The first technique, incoherent detection, also known as direct energy detection, measures the amplitude of the reflected light. Coherent detection however, is a measure of doppler or phase sensitive features of the analysed light. Lidar systems that implement coherent detection generally use optical heterodyne detection and as a result they can operate at much lower power than incoherent detection systems, although [14]. As Lidar systems consists of multiple components including lasers, scanners and optics, photodetectors and receivers, the cost of such a system is driven up and can be very expensive to implement.

Ultrasonic

Ultrasonic sensors are a cheap and effective way to implement a localization system. Whereas the three other suggested sensor types utilise electromagnetic waves at different parts of the electromagnetic spectrum, ultrasonic sensors emit sound waves outside the range that humans can hear and measure the time taken for the wave to return. Sound waves are vibrations of air molecules so can be affected by factors such as the humidity and temperature of the air they are present in. Ultrasonic sensors can detect distances to objects using the time-of-flight principle, in which an ultrasonic signal is transmitted, and the time taken for the signal to return can be used to calculate the distance.

Issues do arise with objects that absorb ultrasonic waves, such as human beings. Other issues include problems associated with detecting certain obstacles such as corners and doorways. This is due to the path that the reflected signals take. The next section will explore these issues in greater detail.

Summary

Ultrasonic sensors have been chosen for this project primarily due to the fact that they are cheap, they perform well and they are straightforward to use. Although infrared sensors can be cheaper, the performance drop is significant and although lasers perform better, the price increase is significant. Ultrasonic sensors appear to have the best performance to implementation cost ratio. Table 2.1 compares the identified sensor types in terms of both range and price.

Sensor Type	Typical Range	Typical Price
Ultrasonic	2cm - 3m	£25-30
Laser	1000m	£100+
Infrared	3cm - 5.5m	£10-15
Camera	100m	£60

Table 2.1: Comparison of various sensor types

2.2 Ultrasonic Sensors

Ultrasonic sensors consist of a sender and receiver, which in the context of sound waves can be thought of as a loudspeaker and a microphone. The loudspeaker emits a sound wave, which when reflected by an object, will be picked up by the microphone. We know the time delay between the emission and reception of the pulse, the speed of sound in dry air (0% humidity) can be calculated using the equation:

$$speed = 331.3 + (0.606 \times \theta) ms^{-1}$$

Where θ is the air temperature in degrees celsius. We can then calculate the distance of the object using the straightforward equation:

$$distance = \frac{speed \times time}{2}$$

Ultrasonic sensors have a wide variety of applications and are commonly used: in cars to detect proximity to obstacles to assist with parking [15], for map building and localisation in autonomous robots [16, 17] and in medicine to take pictures of the body, enabling diagnosis of diseases such as cancer [18], a process called medical ultrasonography.

The majority of manufactured ultrasonic sensors function by use of piezoelectric crystals. When a current is applied to these crystals, they quickly change shape, and the vibrations cause them to emit sound waves. They also are able to receive ultrasonic waves, as when the waves come into contact with the crystals, it causes them to emit an electrical current. This current can be measured in order to identify the features of the wave that has been received. This is how many ultrasonic sensors can both transmit and receive signals, and a common feature of many types of transducers.

There is another type of ultrasonic sensor available, known as capacitive micromachined ultrasonic transducers (CMUT). They were first reported in the late 1980s and early 1990, however the fabrication process was difficult and their performance could not match that of the traditional piezoelectric models. It was not until 1994 when Haller and Khuri-Yakub demonstrated their CMUT with higher performance and an improved fabrication technology [19] that the devices started to become a viable solution in some applications. These devices are capacitors, but with a moveable electrode. Applying a voltage causes the electrode to vibrate and produce ultrasonic energy. Like piezoelectric sensors, they can also be used as receivers;

when an ultrasonic signal is received, this moves the electrode and consequently causes a change in capacitance which can be measured. CMUTs are a relatively new technology, and it has been suggested that many of the inherent features will allow great advances in the field of ultrasonic imaging [20]. These features include the ability to easily create arrays of sensors, as the sensors are created using micromachining. This in turn can enable sensor systems which have a much greater bandwidth than other technologies such as the aforementioned piezoelectric transducers.

2.2.1 Beam Pattern

The beam pattern of an ultrasonic transducer is defined as the relative sensitivity of the transducer as a function of spatial angle [21]. The width of the beam pattern, or beam width, is determined by the diameter of the radiating surface and the wavelength of the transmitted sound wave. By increasing the diameter of the transducer radiating surface, the beam of sound transmitted will be narrower, and vice versa. This enables ultrasonic transducers to be designed with varying ranges, to suit a variety of different applications.

Manufacturers often publish the beam pattern of their sensors, enabling people to select which sensor is appropriate for their application. Figure 2.4 shows the beam pattern for the MaxBotix HRLV-EZ0 Range Finder, created by positioning various-sized dowels around the module. As expected, the larger the dowel, the more ultrasonic energy reflected back. A narrower beam will be more appropriate for range finding at further distances, because as the signal travels, less energy is dispersed, so will travel for longer.

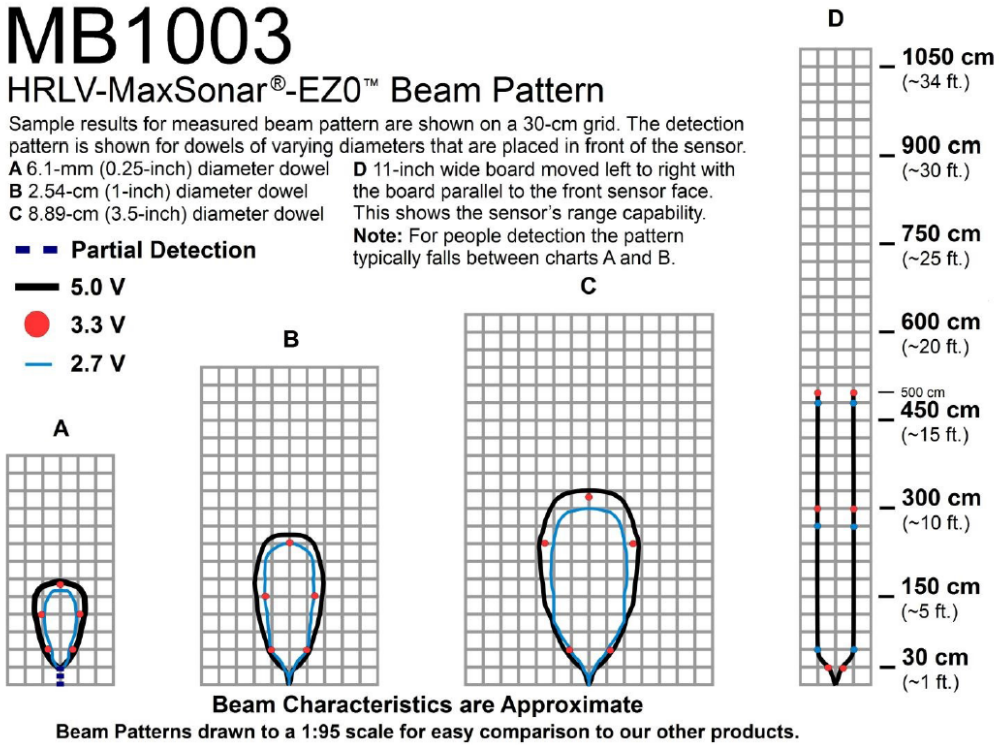


Figure 2.4: Beam pattern of the MaxBotix HRLV-EZ0 Range Finder [22]

2.2.2 Range Finders vs Transducers

There are a multitude of different ultrasonic sensors available, but they can be split into two distinct categories: range finders and transducers. Range finders output a digital signal whereas transducers output an analogue signal. When building an autonomous robot, the type of sensor used is an important decision and will determine how the data is processed.

Transducers

On receiving an ultrasonic wave, a transducer will output the analogue signal that it receives. This means that there is more information available about the received signal and consequently, this extra information can be used to learn more about where the wave has come from and extra properties about the received signal. In order to digitally manipulate the analogue data received from the transducer, an ADC (analogue-to-digital converter) will be required.

Figure 2.5: A 40kHz ultrasonic transducer

When choosing an ultrasonic transducer, one must consider its specifications and the implications of those specifications. The range of the sensor is an important factor to consider, some modules will be designed to work at much higher distances than others, and this may factor into the cost. Table 2.2 compares some of the different ultrasonic transducers available and identifies the properties of each.

Sensor	**Beam Width (°)**	**Operating Frequency**	**Price** [1]
Navo 16mm Diameter	60 ± 15	$40.0 \pm 1.0kHz$	£2.20
Multicomp MCUSD16A40S12RO	50	$40.0 \pm 1.0kHz$	£1.81
Prowave 400ST/R100	72	$40.0 \pm 1.0kHz$	£3.90
Murata MA40S4R	80	40kHz	£4.86

Table 2.2: Comparison of ultrasonic transducers

Range Finders

Range-finders consist of two transducers, a sender and a receiver. Containing a built-in digital interface, by analysing the received ultrasonic wave, they attempt to identify the time-of-flight between sender and receiver. As a result, most of the processing that would otherwise need to be performed manually is abstracted away from the system designer. Digital range-finders vary broadly and there are many different models available. Some can be found as cheap as £0.99 for the HC-SR04 module [23] whereas higher quality modules such as the Maxbotix HRLV-EZ0 [22] often cost closer to £30.
There are a number of reasons why range finders may be selected to be used in robotics, the first of which is their ease of use due to their digital interface. To transmit an ultrasonic signal, a digital high signal, usually 3.3 or 5 volts, is sent to the transmit pin on the module and when

[1] Prices from Farnell UK, 17/03/2015

Figure 2.6: HC-SR04 Range-Finder (left) and Maxbotix HRLV-EZ0 Range-Finder (right)

an ultrasonic signal is received, the receive pin will output a digital signal with a pulse width proportional to the distance of the object. As an example, a range finder has been connected to an oscilloscope and the trace of the output is displayed in 2.7. A second related advantage is that individual digital pulses reduce the amount of data used substantially meaning that much less data processing is required of a system incorporating range finders.

There are also several drawbacks to using range finders in robotic systems. The first of which is the lack of information the received signal contains. Although considered an advantage to the extent that less data processing is required, this means that the strength and duration of the received signal is unknown. Hence, there is a distinct lack of versatility in the applications of range finder modules, and multiple techniques that cannot be used as a result. For example, the method which received signals are identified is limited to the method used by the module, usually a simple thresholding algorithm.

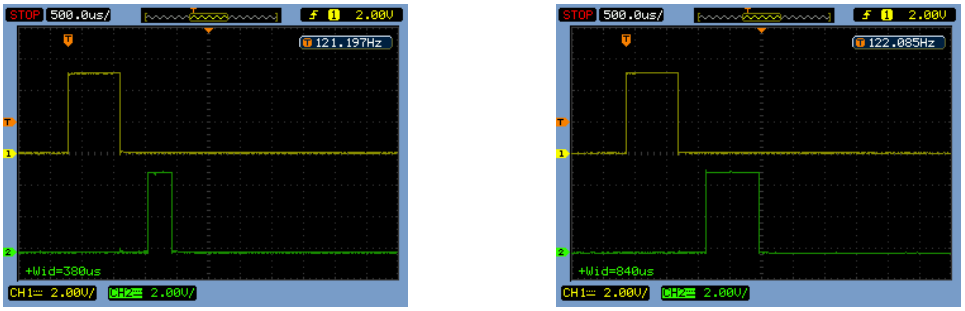


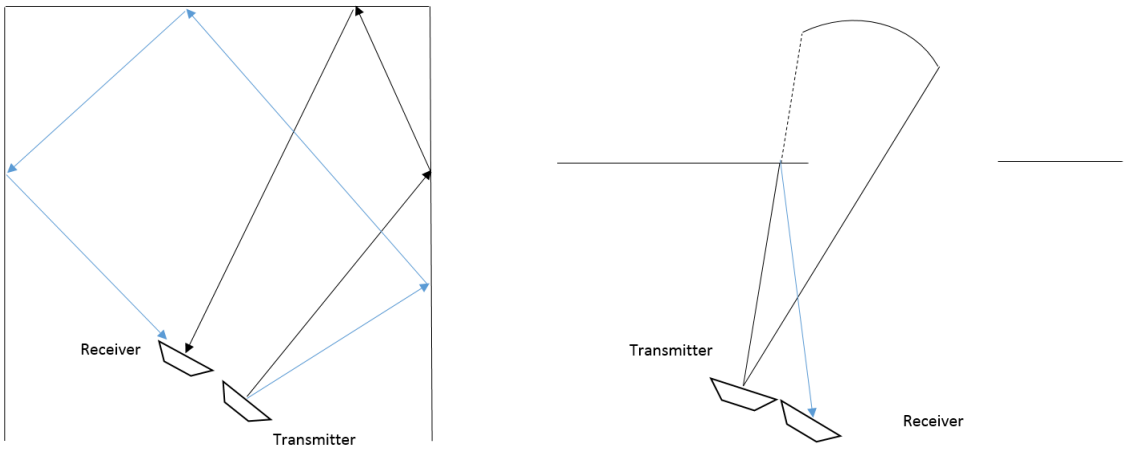
Figure 2.7: Range-finder output trace at 5cm (left) and 15cm (right)

2.2.3 Issues

In the previous section, it was mentioned that ultrasonic sensors run into problems when the waves are absorbed by soft objects. This causes issues as it makes soft objects much harder to detect than objects with hard surfaces. This is not the only issue that needs consideration. The beam width produced by ultrasonic transmitters is often too wide, or sometimes too narrow. To overcome this problem, the beam width of an ultrasonic transmitter should be researched before purchased, so that a model with a beam width appropriate for the application can be selected.

One of the main issues ultrasonic sensors face is detecting corners in walls. If a signal reflects of multiple walls, the time-of-flight detected by the receiver will be longer than the actual distance to the wall and therefore the measurement will be inaccurate. Figure 2.8a shows an example of signals which are reflected. The first signal (black) is reflected off two walls and the second (blue) is reflected off three. Had the top wall not been present, the signals should have reflected away and not been detected by the receiver. However, because the top wall is present, the signals reflect back round to the receiver and a false reading is recorded.

Another major issue with ultrasonic sensors is the detection of walls with doorways. An ultrasonic beam is transmitted as a cone, not a perfectly straight line. So in a situation where the beam meets a doorway, the signal may either simply pass straight through, or as in Figure



(a) Corner issue faced by ultrasonic sensors (b) Doorway issue

Figure 2.8: Ultrasonic reflection issues

2.8b, only partially reflect off the wall. In this scenario, some of the ultrasonic energy will still be reflected, so the receiver will still detect that the wall is present, another false reading.

2.3 Data Capture and Processing

This section looks to explore how an autonomous robot can use the data it receives from its sensors, covering the concepts of sampling, resolution and processing.

2.3.1 Sampling

Before the data is processed, it must first be captured to the system. Sampling is the process of converting an analogue signal to a digital one. A single value that can be represented digitally is recorded each time a sample is taken. How often a sample is taken is determined by the sampling rate. This can be achieved with an analogue-to-digital converter (ADC).

There are many different types of ADC available, designed for different applications. These include direct-conversion ADCs, sigma-delta ADCs, pipelined ADCs and integrating ADCs. Pipelined ADCs are the most popular ADC architecture used in applications that require a high sample rate, up to 100 megasamples/s whereas integrating ADCs offer low power consumption, high resolution and good noise rejection [24]. Sigma-delta ADCs offer sample rates which range between 48 kSPS and 192 kSPS [25].

The Nyquist Sampling Theorem, first implied by Nyquist [26], and later proved by Shannon [27], states that the sampling frequency should be at least twice as high as the frequency contained in the signal. So, for a 40kHz ultrasonic sensor, the sampling frequency of the ADC should be at the very least 80kHz. As a result, a sigma-delta ADC should be sufficient for sampling data from a sensor of these specifications.

2.3.2 Resolution

The resolution, also known as bit depth, is the measure of how much information is recorded per sample. By increasing the resolution, each the precision of each individual measurement improves. From this, it may appear that the higher the resolution is, the better the system, however the more information in each sample, the more storage space and memory required by the system. When the sample rate is high and the resolution is high, the performance rate of the system could drop significantly. This is because a higher sample rate indicates more samples will be taken and a high resolution means each sample contains more information, requiring more processing power to process the data. As a result, there is a trade-off between accuracy and performance. A good system will use only as much resolution as required, in order to maximise performance. The resolution is also affected by noise. If any form of noise is present, the advertised resolution will not be as stated, but distorted by the presence of noise.

2.3.3 Processing

Once the data has been captured and converted to a digital format, the system will need a means to extract and manipulate it. It must also be able to control the signals which are being transmitted to ensure synchronisation between the send and receive phases. While this could be done via a serial link to a laptop or desktop computer, that would remove the sense of portability, a crucial component in autonomous robotics. There are many microcontroller-based mobile platforms that are very capable but also able to run on battery power. Table 2.3 compares some of the available options in terms of their processing capabilities and retail value. While the Raspberry Pi appears to have the best price-performance ratio, it also must run a full desktop operating system, which creates considerable processing overheads, and is unnecessary for an autonomous robot, which does not even require a graphical user interface, let alone a full operating system.

In an autonomous ultrasonic robot system, there will be a very large stream of data constantly being delivered to the processor on board. Much of the data received will be useless, especially

Table 2.3: Comparison of available microcontrollers

Microcontroller	Processor	Memory	Storage	Price
Raspberry Pi	700 MHz single-core	512MB RAM	SD card	£30
mbed LPC1768	96MHz Cortex-M3	64KB RAM	512KB flash storage	£55
Arduino Duemilanove	16 MHz ATmega328	2KB RAM	32KB flash storage	£17

in a system that uses a high resolution. The more data that is received, the more the processor will have to work. Some of this useless data can be discarded before the primary processing actually takes place, through an operation called principle component analysis (PCA). Principle component analysis transforms a set of data into a set of identified principle components, where the number of principle components is less than or equal to the number of values in the original dataset. PCA is very useful in signal processing as it allows the signal to be compressed to a much smaller size. PCA is often utilised in signal processing and has been used in the context of ultrasonic signals by Jiminez et al. [28]. Their method involves encoding ultrasonic pulses in two separate transducers outputting simultaneously and using PCA to analyse and decode the signals, in order to calculate the time-of-flight of these signals. This information then allows the location of the transducers to be approximated.

2.4 Noise

Noise can be defined as any unwanted modification to a signal as seen on reception of the signal, but the modification can take place at any point during the lifespan of the signal, whether during the signal's transmission or during the capture process. Figure 2.9 shows Shannon's simple explanation [27] of how noise affects the transmitted signal so that the received signal is is altered. An information source passes a message to a transmitter, this transmitter transmits a signal to a receiver, but before the signal reaches the receiver, a noise source has interfered with the signal, leading the signal received by the receiver having been altered.

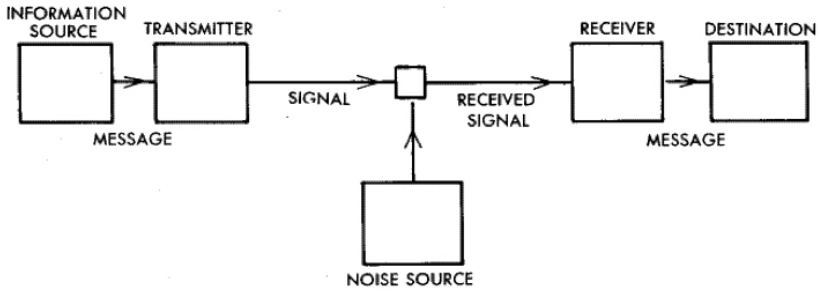


Figure 2.9: Noise in a simple communication system

2.4.1 Sources of Noise

To be able to solve issues surrounding noise in communication channels, the cause of the noise must first be understood. There are two broad categories that sources of noise can be divided into: artificial noise and natural noise.

Artificial noise can be considered as any signal affecting the communication channel that is man-made whereas natural noise is signals which which are present that are not created by any man-made object. In the case of ultrasonic signals, artificial noise could include a number of devices which are advertised and intended to emit ultrasonic waves, as well as devices which

transmit ultrasonic waves as an unintentional result of their operation [29]. More specifically, ultrasonic robots could receive noise from old reflected signals which they have transmitted, or signals transmitted by other ultrasonic robots present in the same environment. Natural ultrasonic noise could include any natural causes of changes in the vibration of air molecules, such as changes in temperature or humidity.

2.4.2 Interference

Before explaining the two types of interference experienced in ultrasonic systems in greater detail, it is first worth noting the definition of a *channel*. A channel, or communication channel, is used to transfer information from one or more senders to a receiver or multiple receivers. A channel can refer either to a physical channel such as a wire, or a multiplexed transmission such as those over radio waves. In the context of ultrasonic transmissions, a channel is determined by the frequency at which the information is transmitted.

Co-channel Interference (CCI)

The cause of noise in systems which use ultrasonic sensors can partly be attributed to co-channel interference, also known as crosstalk, especially in implementations where arrays of ultrasonic sensors are used. Although it is relatively trivial to transmit a signal, receive a signal and measure the time-of-flight of this signal, it is not easy to recognise that the received signal is the same signal that was transmitted, and has not come from a separate transmitter. It is also not easy to recognise that a received signal is the same as the transmitted signal, but has reflected off multiple surfaces and will not give an accurate representation of the distance to the initial surface. Co-channel interference is the presence of an unwanted signal that utilises the same channel as the expected signal [30].

Adjacent-channel Interference (ACI)

Adjacent-channel interference is caused by signals transmitting in neighbouring channels to the expected channel. This will be familiar to many people who have attempted to manually tune a radio before; by tuning the radio to the adjacent channel to a known radio station, the known radio station will usually still be heard, albeit a very poor quality signal. The cause of this type of interference is that a receiver is never set to receive signals that are the exact frequency of the transmitted signal, but allow a certain tolerance around this value. Consequently, signals from neighbouring channels will sometimes be picked up by the receiver which will affect the signal that it is trying to receive.

2.4.3 Electrical Noise

Thermal noise, also known as Johnson-Nyquist noise, is a form of noise that is unavoidable. Discovered by John B. Johnson in 1926, but explained by Nyquist in 1928 [31], thermal noise is generated by the random thermal motion of charge carriers in an electrical conductor, which happens with any applied voltage, regardless of strength. The noise is approximately white in distribution, which means that the spectral density is almost constant throughout the frequency spectrum. As a result, when sampling data using an ADC, every single sampled value will contain some amount of thermal noise, and consequently the resolution of the ADC is limited, so even if the claimed resolution is very high, it will always be limited by the thermal noise present. Methods to overcome this include oversampling at a much higher rate than necessary and averaging the values to the original desired sample rate. As thermal noise is random, the variations in the signal caused by this noise should average 0 overall. So by averaging the samples around the desired sample, this should reduce the effect caused by thermal noise.

Another potential solution to this problem is to reduce the heat present in the ADC circuit by use of a cooling system, therefore reducing the amount of thermal noise present.

2.5 Noise Reduction Techniques

The noise reduction techniques that have been identified include two spread-spectrum transmission methods and a technique to encode the transmissions so that each signal can be identified as originating from a specific sensor.

2.5.1 Spread Spectrum Techniques

Frequency-hop spread spectrum (FHSS) is a method of transmitting signals over more than one frequency channel. It is one of many types of spread-spectrum transmission techniques. This is in contrast to fixed-frequency transmission, where just one frequency channel is used. In FHSS, a carrier signal jumps between a set of frequencies within the available bandwidth, using a pseudorandom sequence known to both the transmitter and receiver [32]. Because each pulse will be using a different frequency, the risk of collisions between signals will be reduced and consequently, the likelihood of interference between signals is reduced. This method can be used with ultrasonic sensors as Saad and Bleakly have demonstrated [33]. Their method uses a wideband frequency-hop spread spectrum ultrasonic signal with the aim of increasing robustness to noise and reverberation. They created a FHSS signal with 16 separate frequency slots between 28kHz and 36kHz, with 460Hz separation between them. A range estimation algorithm is then presented, utilising cross-correlation techniques. The assumption is made that the velocity of sound is constant throughout, as the effects of variation in the air temperature and humidity is assumed to be negligible due to the measurements being taken over a short period of time. In this instance, this assumption would appear to be correct, but it is an important consideration in all ultrasonic systems and should not be assumed to be true in all cases.

Direct-sequence spread spectrum (DSSS) is another technique that falls under the category of spread-spectrum transmission. It is similar to FHSS, however the data-modulated carrier signal is further modulated by a pseudorandom binary sequence [34]. Hazaz and Ward use DSSS with ultrasonic sensors in an attempt to achieve multiple-access properties and robustness towards noise [35]. Although they report success, their algorithm had an update rate of five location estimates per second, due to the limits imposed by the computational power of their workstation PC. As this was the case running the algorithm on a computer, it seems impractical that it could be implemented on an autonomous robot, with a microcontroller that is likely to be significantly less powerful than a workstation computer.

2.5.2 Coded Signals

A key idea in information theory is the concept of encoding information. In order to encode some information, a set of symbols, or alphabet, known by both a sender and receiver must be established. These symbols correspond to some source alphabet. A message can be encoded by concatenating symbols sequentially. Once this code has been generated, it can be transmitted to one or more receiver. In order for a receiver to understand the contents of the message, it must be decoded [36]. An example of codes that are very commonly used is the American Standard Code for Information Interchange (ASCII) [37], which assigns each alphabetical letter to a binary number, and is used by keyboard interfaces to send letters to a computer via a serial link. Each symbol is a binary sequence of length 7, enabling 128 unique characters and consequently using 7 bits of information per code.

One proposed method to overcome the problems associated with crosstalk in ultrasonic transmissions is to use coded signals. In this method, each transmitted pulse is assigned a unique code, so that the transmitter can be identified even if the signal's echo is received by another receiver. The unique code can be determined by the length of the pulse, for example, a pulse transmitted from one transducer could last for 1 microsecond, a pulse from another could last 2 microseconds and so on. The method proposed by Shoval and Borenstein [38] assigns each burst of pulses with a code, so that the receiver can unambiguously determine which transmitter the signal originated from. Their results indicate that short codes provide more accurate data for reflections returning from obstacles within close proximity, and consequently a variety different length codes may be required to cover a wide range of measurements. A similar technique was also used by Jiminez et al. [28] as described in Section 2.5. Their results indicate success however their system only measures the first echo received, leading to a redundancy in a large proportion of the recorded data.

2.6 Pattern Recognition

Pattern recognition is the concept of attempting to extract certain patterns from sets of data, and inferring ideas by processing the data. Pattern recognition can be used to solve problems such as speech recognition, face detection in images, handwriting recognition and computer-aided diagnosis in medicine. Pattern recognition is used in supervised learning, where labelled training data exists, as well as unsupervised learning, extracting hidden patterns from unlabelled data.

There are a number of different algorithms and methods that have been used to implement pattern recognition, but each method is designed to solve a different branch of pattern recognition. A specific branch of pattern recognition, known as classification, is the problem of identifying which category, or class, a new observation belongs to based on previous observations contained within training data.

The previous section identifies coded signals as a means to distinguish between ultrasonic signals from multiple transmitters. Whereas the identified method [38] recognises specific signal codes by constructing circular and elliptical arcs and analysing their corresponding intersections, this is an example of a classification problem, where each code corresponds to a separate class. Consequently, the problem of identifying codes could be solved using alternative pattern recognition techniques. Examples of alternative methods include classification algorithms, clustering algorithms, regression algorithms and neural networks.

2.7 Neural Networks

Neural networks are statistical models used to estimate and approximate functions to map input data to target data, that are inspired by the structure of the biological neural networks found in the brains of humans and animals. Artificial neural networks are often used in pattern recognition problems as biological neural networks are very good at recognising patterns. For example, every time a human reads a piece of text, they are able to read it because they recognise the visual patterns that each character consists of and can easily distinguish between letters at an impressive speed. Artificial neural networks have been used successfully in a host of pattern recognition applications such as face detection [39].

There are several different types of neural network architecture, the most common of which is the feedforward network. In a feedforward network, the data moves in one direction and the structure of the nodes in the network never forms a directed cycle. This is in contrast to recurrent neural networks, in which the flow of data is bi-directional.

The single-layer perceptron (SLP) is an example of a feedforward network, based on the

perceptron model invented by Rosenblatt [40]. A perceptron takes multiple inputs, applies a set of weights corresponding to those inputs and performs some function using the weighted sum of the inputs to give an output. The SLP consists of a single output layer of perceptrons, mapped to a layer of source nodes. This simple model is able to solve classification problems if the data is linearly separable. However, when the number of classes is greater than 2, the problem becomes more complex and this type of network is no longer suitable.

The multi-layer perceptron (MLP) can solve more complex problems that are not linearly separable. The structure of a MLP is more complex than that of the single-layer-perceptron, consisting of an input layer, one or more hidden layers and an output layer. Although multiple hidden layers can be used for problems with a very high complexity, it has been proved that networks with just one hidden layer are capable of computing any measurable function [41].

Another common neural network architecture is radial basis function (RBF) networks. Unlike the multi-layer perceptron, radial basis functions are likely to contain more than one hidden layer. RBF networks are simply neural networks which use radial basis functions as activation functions. Neither multi-layer perceptrons nor radial basis function networks tend to perform well with noisy data but the performance hit is greater in radial basis function networks [42].

There are a number of examples where artificial neural networks have been used in autonomous robotics previously. Pomerleau presents ALVINN (Autonomous Land Vehicle In a Neural Network) [43], a system developed to train a neural network in real time to autonomously control a modified van on various road types. The aim is to train a network to operate the vehicle, based on observing a human drive. The system uses images of the road taken from a video camera as inputs to a neural network, and steering commands as its outputs. The network is trained on-the-fly, meaning the system learns which image features are relevant depending on the specific road type or environment. The claim is made that the system can be trained for a particular environment in just 5 minutes of observations.

As identified in the previous section, pattern recognition could be used to implement a system to classify coded ultrasonic signals. A multi-layer perceptron would be an appropriate network to use for classification tasks, as there is a large amount of information available regarding their use, as well as being able to solve any measurable function, they can be scaled to sizes of gigantic proportions in order to work with massive amounts of data if needed.

2.8 Summary

In summary, this chapter has explored and reviewed a variety of concepts and methods which could be applied in order to meet the aims of the project identified in the previous chapter.

Section 2.1 gives an overview of autonomous robotics and compared the various sensor types that can be used to help achieve autonomy. After the comparison, it is determined that ultrasonic sensors seem most appropriate for the task. The section also briefly covered the ideas of mapping and localisation, together with some of the design choices one has when implementing these features.

Section 2.2 looks at ultrasonic sensors in greater detail, explaining how they function, before comparing the differences between ultrasonic transducers and range-finder modules. The final part addresses some of the issues associated with ultrasonic sensors and potential solutions for these issues.

Section 2.3 investigates methods of capturing and processing data received by an autonomous robot's sensors. The theory behind data sampling is explained and the various types of analogue-to-digital converters are compared. The section then explores the concept of data resolution, identifying the inherent trade-off between accuracy and performance associated with increasing the resolution. Lastly, the concept of processing is explained and the associated issues are briefly covered, and examples of data processing are given.

Section 2.4 explains the the ideas behind noise, where noise originates and the different types of noise that could affect an autonomous robot. The section covers both co-channel interference and adjacent-channel interference, that can affect ultrasonic signals while they are travelling through the air, as well as electrical noise, which can occur while the signal is travelling through any electronic circuit.

Section 2.5 analyses various methods of reducing noise in signals. Of the methods identified, DSSS requires a lot of processing, FHSS appears to require bespoke hardware such as a bandwidth expansion circuit. Implementing coded ultrasonic signals seems most appropriate and achievable for an autonomous robot.

Section 2.6 expands on the coded signals method identified in section 2.5, suggesting using pattern recognition techniques as a means to decode coded ultrasonic signals. Pattern recognition is defined and explained, and examples of how pattern recognition can be used are identified. Different implementations of pattern recognition are investigated and potential solutions are discussed.

Section 2.7 explores artificial neural networks as a means to implement pattern recognition, giving a brief overview of how they work. Then the different types of neural network are compared and some of the design choices that must be made when implementing an artificial neural network are described in more detail.

3 Problem Analysis

This project specifically looks at the use of ultrasonic sensors as used in an autonomous robot system that navigates in an indoor environment. The previous chapter addressed some of the issues associated with such a system and this chapter looks to explore these issues in greater detail.

The robot is assumed to have 8 ultrasonic sensors, each with a beam width of 45 degrees, in order to cover the entire circumference of the robot's body, as shown in Figure 3.1. Many transducers have a beam width greater than 45 degrees, as identified in the previous section however, this will not be an issue as beam overlap can actually be advantageous to the system [44]. This should ideally enable a very high accuracy positioning system as the robot will receive data from 8 separate directions. The ultrasonic signals transmitted by the robot will bounce off any hard surfaces and the signals will be captured by the sensors and processed in order to determine the distance to the reflecting surfaces. Hard surfaces which do reflect the signals also do not absorb much of the sound energy. Consequently the chance of the signals reflecting of multiple surfaces before detection from the sensors increases and the likelihood of erroneous readings due to crosstalk increases as a result.

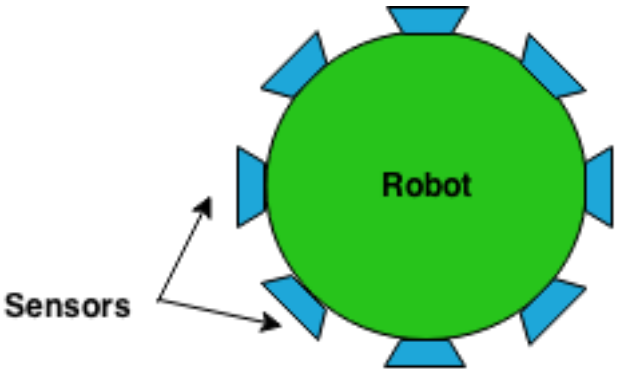


Figure 3.1: Sensor organisation

The previous chapter investigated a variety of methods that could eliminate the problems associated with crosstalk. The most viable of which appeared to be encoding the signals, assigning specific patterns for each transmitter. A system will be developed that is capable of generating coded signals, as well as the capability of observing these signals once reflected, and decoding the signals to determine their origin.

Ultrasonic transducers have been chosen over range finders, as they are far more versatile, allowing customised signals to be both generated and received. This is an important consideration for generating coded signals as it would not be possible to achieve with a traditional range finder module. This ultimately eliminates using range finders as an option, as they are not designed to be used for anything other than range finding.

Once a signal has been received by a transducer, the system must be able to identify it. Options to achieve this include simply setting a threshold value for the output from the transducer and identifying the start of a received signal as the point where this threshold is exceeded. An advantage to this method is that it requires very little processing power. However, there are plenty of alternative solutions, including a variety of statistical analysis techniques, such as peak-finding algorithms.

Decoding of the signals will be attempted using an artificial neural network, which will be provided with training data in order to learn which pattern corresponds to which code. It should then be able to successfully decode new signals provided to it automatically. Neural networks have been selected over other pattern recognition techniques as the literature reviewed suggests that neural networks can be used successfully in ultrasonic robot systems, as well as the fact that the network could eventually be trained to perform other tasks in the system such as controlling the movement of the robot.

3.1 System Requirements

The following requirements set out the criteria which enable the proposed system to be evaluated.

- The system must be able to classify 90% of observations
- The system must be able to identify the time of flight of each observation
- The system must correctly separate the reflected signal from the rest of the values within each observation
- The system must be able to correctly classify observations in the presence of white noise
- The system must be able to classify observations where multiple signals are present

3.2 System Tests

This section lays out a set of unique system tests in order to evaluate the performance of the system, allowing conclusions to be drawn about the overall success of the project.

An important measure of an autonomous robot's performance is the speed at which it can move. One of the largest factors affecting an autonomous robot's performance is the amount of data it must process. Therefore, it is key that the amount of data processing must be minimised as much as possible without jeopardising the function of the robot. Once a system has been developed that records data from an ultrasonic transducer and classifies this data, the resolution of each observation will be reduced and the performance evaluated.

The system must be capable of separating signals from observations, in order to then classify these signals. This will be tested by attempting to separate signal from observation for all recorded observations. If this can be achieved for all observations, the test can be considered a success.

As identified in the previous chapter, white noise is present in all electronics, so the system must be able to perform in conditions where white noise is present. White noise is typical in electrical components such as op-amps which are commonly used in amplifying circuits. White noise will be artificially added to a set of observations and fed into the developed system in order to evaluate the robustness of the system in the presence of white noise.

As the system is designed to deal with crosstalk scenarios, where signals from other transmitters are present, the system must be able to identify these situations and deal with them accordingly. In order to test the system under these circumstances, certain signal codes will be artificially added to a set of observations, and the system will be tested to see if it can identify these signals and pass them separately to be classified.

4 Design and Implementation

This section outlines the design of the proposed system and explains in detail the design choices made together with justification for these decisions.

4.1 Hardware

In order to generate and receive the ultrasonic sensors, two 40kHz ultrasonic transducers have been chosen. Transducers have been selected over digital rangefinders because on receiving a signal, a lot more data is available to manipulate. A digital rangefinder will simply give a digital signal when the wave is received threshold is met. The receiving transducer was connected to an amplifying circuit, provided by my supervisor, the schematics for which are displayed in Figure 4.1. In this circuit, the output signal from the transducer is connected to an inverting amplifier with a gain of 47, constructed using resistors and a 741 op-amp . The signal then passes through a summing amplifier, in which the gain is still 47, but a DC offset is added to the signal, which can be altered with a variable resistor. The offset is added so that the signal remains a positive voltage, as opposed to a alternating between positive and negative, so that the circuit will be compatible with microcontrollers, which do not have the capabilities to work with negative voltages. The total gain of the circuit is equal to the gain of the first amp multiplied by the gain of the second: $47^2 = 2209$. This was determined before the circuit was provided, after a series of tests settled that it was appropriate for the types of indoor ranges it would be used for.

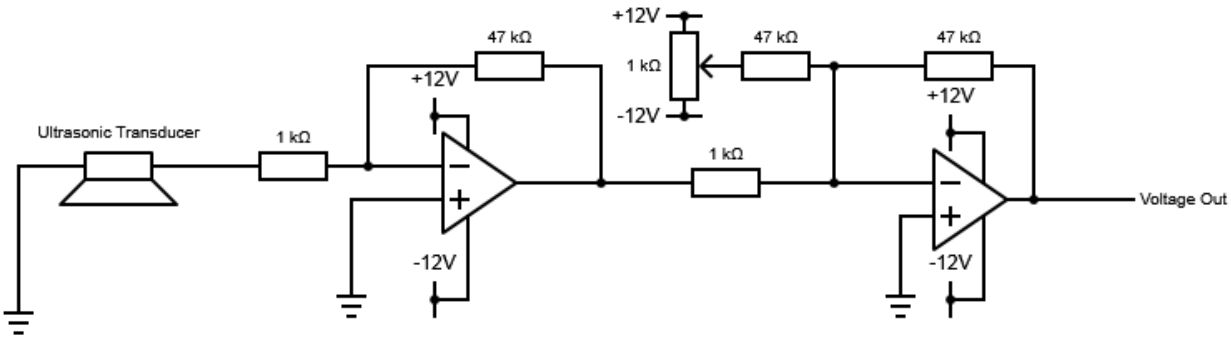


Figure 4.1: Circuit diagram of receiver amplifying circuit

A circuit was also provided for the transmitting transducer. Figure 4.2 shows the schematics for this circuit. It is a digital logic circuit, and its purpose is to enable altering the shape of the ultrasonic beam. The signal to drive the transmitter is connected to a series of inverters, using a standard 7404 TTL chip. Although the logical output would be identical if in place of the not gates in parallel, single not gates were used, connecting them in parallel allows more current to flow through to the transducer, and is common practice in these type of circuits.

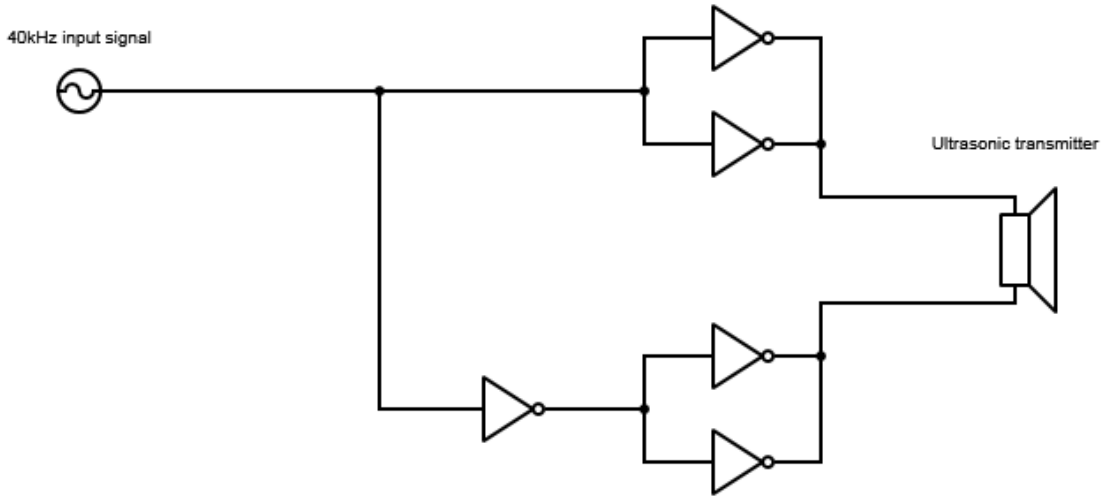


Figure 4.2: Circuit diagram of ultrasonic transmitter

In order to drive the transmitting transducer, an Arduino Duemilanove microcontroller [45] was used.This seemed like an appropriate choice of microcontroller because it is cheap, yet able to output high frequency square waves. It is easy to interface with, using a lightweight multi-platform desktop application and has a standard USB port to enable serial connectivity with a computer. The Arduino is also programmed using the C programming language. Both the mbed LPC1768 and the Raspberry Pi were considered but the Arduino had the advantage over the Raspberry Pi that it could be run on USB power as opposed to requiring a DC power supply, and the mbed didn't offer many extra features that warrant the higher price. The arduino has a 16 MHz ATmega328 processor, which is more than powerful enough to generate a 40kHz signal.

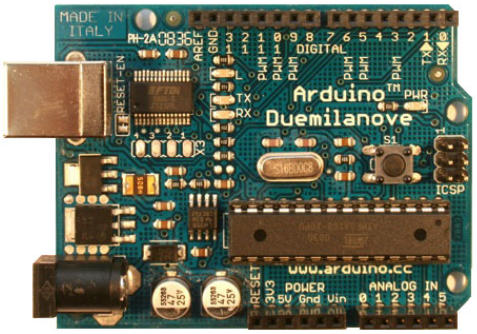


Figure 4.3: The Arduino Duemeilanove microcontroller [45]

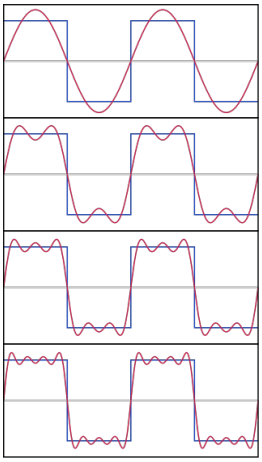


Figure 4.4: Square wave, first four partial sums of Fourier series [46]

The ideal signal to drive the transducer that emits the sound wave is a 40kHz sine wave. This is because the transducer is driven by a 40kHz alternating current (AC) signal, and a 40kHz square wave can be represented as an infinite summation of sine waves as explained by the Fourier series expansion, the first four partial sums of which are illustrated in Figure **??**. Therefore, the square wave signal includes multiple harmonics that are not within the range of frequencies required by the transducer, thus increasing the amount of noise present in the signal. Methods of generating sine waves using digital equipment were investigated. The first of these methods looked at using the analogue output capabilities of the microcontroller in order to manually create a sine wave using lookup tables. This was tested on both the Arduino Dueminalove and the mbed LPC1768 and neither were able to output sine waves anywhere close to the required 40kHz. Another method considered connecting an external DAC connected to the microcontroller. The DAC available for testing was able to output an analogue voltage at a rate of 200kHz. This meant that even if the DAC was performing at its theoretical best, each individual 40kHz wave would have to be represented with just 5 voltage changes, meaning the created sine wave would look very jagged and would definitely not be pure. Finally, a third method to create a sine wave was to output a square wave, but pass the signal through a series of filters to leave just a pure sine wave. Ultimately, it was decided that it wasn't necessary to use a sine wave to power the transducer and that a square wave, which could easily be output by the microcontroller [47], would suffice.

4.2 Signal codes

In order to encode the signals driving the transmitter, a code must be decided on. The number of codes available is determined by S^L, where S is the number of symbols and L is the length of the code. In this design task, the number of codes should be equal to the number of transmitting transducers positioned on the body of the robot. As indicated in the problem analysis, the system is designed to work with a robot with 8 transmitters. However, the method is designed so that it could be scaled to an arbitrarily-numbered sized array of sensors. 8 codes can be achieved with just two symbols with a message length of 3.

The microcontroller will be programmed to output a 40kHz signal to drive the ultrasonic transducer, and this frequency will remain constant. If the frequency deviates away from 40kHz, the strength of the signal creates by the transducer is weakened. Before describing how the signals can be encoded, there are certain terms that should be unambiguously defined. A *pulse* is considered as a single period of an ultrasonic wave and a *burst* is a sequence of one or more pulses. By varying the length of the pulses, they can be used to encode information. By setting a short pulse to represent one symbol X and a long pulse to represent another symbol

Y, this essentially allows a binary alphabet to be used to encode the signals. To enable 8 codes to be used, the symbols can simply be ordered in different patterns of length 3 as displayed in table 4.1 .

Code #	Code
1	XXX
2	XXY
3	XYX
4	XYY
5	YXX
6	YXY
7	YYX
8	YYY

Table 4.1: Table of Codes

It was decided that the symbol *X* would be represented by a 40kHz burst of 4 pulses and the symbol *Y* would be represented by a burst of 16 pulses. A program was written to the Arduino microcontroller to generate the signal codes. A function was written that takes in 3 values representing the 3 symbols that the signal code comprises of. For the symbol *X*, the function tells the output pin to generate a 40kHz square wave, wait for 29 microseconds, then stop outputting. This generates a single burst of 4 pulses. The same function generates the symbol *Y* using the same process but waiting for 165 microseconds. These values were determined by trial and error as the relationship between the delay and the number of pulses in the burst is not linear. In order to ensure the bursts within a code are distinguishable, they are separated by a delay of 500 microseconds. Figure 7.1 shows the program written to the microcontroller to generate the codes to drive the transmitters, and Figure 4.5 shows an example generated code traced on the oscilloscope.

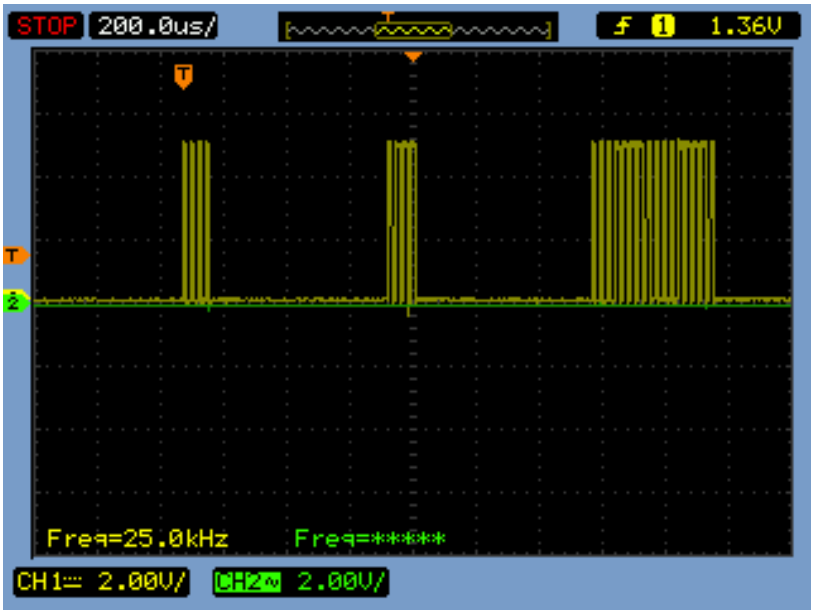


Figure 4.5: Code #2, generated by the microcontroller

4.3 Data Capture

An important consideration was how to capture the data from the sensors, in order to process it. The data must be converted into a digital format, in order to be stored. This could be performed using a analogue-to-digital converter (ADC). However, this project does not intend to actually build a robot and is simply a proof of concept, so for prototyping purposes, a the sensor data was captured using an oscilloscope. The oscilloscope model is the Agilent Technologies (now Keysight Technologies) DSO1002A [48]. It has a sample rate of 2GSa/s and a bandwidth of 60 MHz. This means that it is capable of measuring signals up to 60MHz and the sample rate can be set at a rate up to 2 billion samples per second. This meant that there was a high range of sample rates to choose from, and the oscilloscope also provides a graphical view of the data that was being captured.

The ultrasonic transmitter and receiver were positioned perpendicular to a flat, hard, reflective surface. The microcontroller was programmed with the code #1 (XXX). The distance between the sensors and the reflective surface was initially set at 30cm. 3 observations were recorded using this code at this distance. The reflective surface was then positioned 40cm from the sensors and 3 more observations were recorded. The distance was then increased another 10cm and this process was repeated until the reflective surface was 90cm away from the sensors. This gives 21 separate observations for one particular code. The microcontroller was reprogrammed with each of the other 7 codes and 21 observations were recorded for each, giving in total 168 separate observations. The reason the distance was varied is because a robot is not always going to be the same distance from the obstacles it is detecting and the signals containing the same code are not going to appear identical at different distances. By altering this distance, it aids with the generalisation of the training data so that a certain code can be recognised at a variety of distances. 3 observations were recorded at each distance to ensure that the data was reliable.

Each observation was saved as a comma-separated variable (CSV) containing all the values for each individual sample and the files were saved to a USB flash drive using the oscilloscope's save feature and USB port. 10240 samples were recorded per observation. The oscilloscope was set to record not just the reflected signal but the signal generated by the microcontroller also. The generated signal was recorded as a separate column in the CSV file. This enables the time-of-flight of each signal to be calculated. To illustrate the sampled received signal, an example observation, with the input signal removed, has been plotted and displayed in 4.6. Each observation was recorded at a sample rate of 250kSa/s. As each signal consists of 40kHz pulses, this is far greater than the 80kSa/s required when applying Nyquist's sampling theorem.

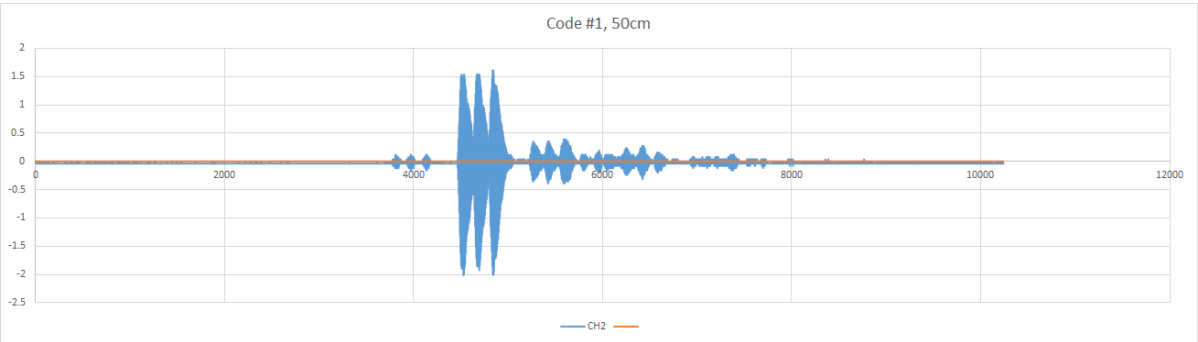


Figure 4.6: An example received signal sampled by the oscilloscope

4.4 Neural Network

Many decisions were made relating to the design and implementation of the neural network. This includes the software chosen to design and create the network, the structure and configuration of the network, and the preprocessing required to provide the necessary information to the network.

There are many software packages available today to aid the design and simulation of an artificial neural network. *MATLAB* [49] contains an integrated set of tools for graphically creating neural networks called the *Neural Network Toolbox* [50] and, coupled with the data processing capabilities already inherent in the MATLAB software, this makes it an ideal candidate to use to develop the system. It should be stated that MATLAB is not very easily portable, and it is not intended that this software be installed onto an autonomous robot, but merely for convenience during the development stage. The available microcontrollers are also not powerful enough to run such a system. A working neural network system could be ported to a mobile implementation in future work. An example of a more portable implementation would be for the network to be written in python using a library such as pybrain [51]. This could run on a high-performance microcontroller such as the quad-core Raspberry Pi 2 [52].

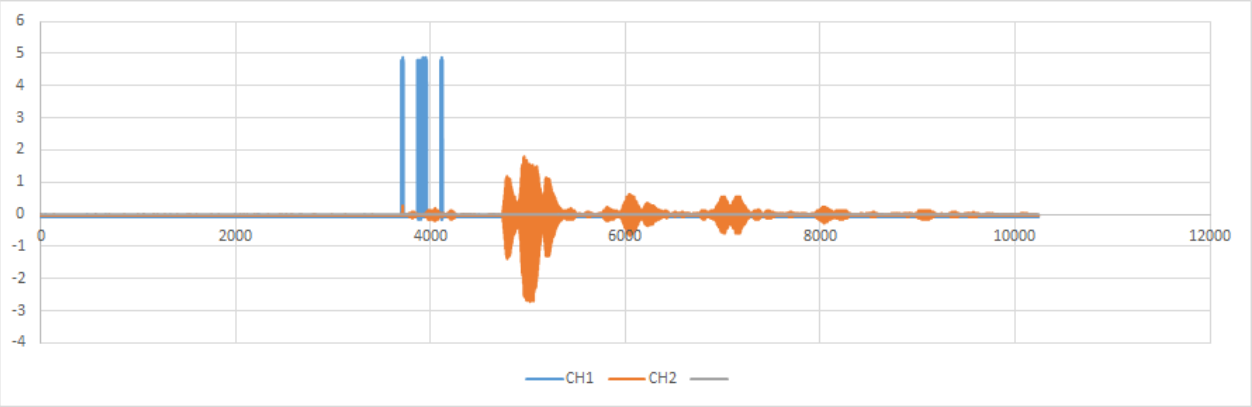


Figure 4.7: Observation at 70cm, code #3

An equal number of observations have been recorded per class. This is to prevent the data becoming unbalanced. Unbalanced data could impede the performance of the network, as the network would become better-trained to classes with more corresponding observations. The observations of each signal code will be used as training data for the neural network. However, each observation contains 10240 samples and therefore 10240 separate values. As there will be 168 inputs, 10240 is far too many values for the network to work successfully. This number needs to be significantly reduced before the data is fed into the network. One potential method to solve this problem is to take the average value for each 10 values in an observation in order to reduce the size by a factor of 10. This would solve the issue surrounding the size of the inputs, but it is not an optimal solution. This is because, in each observation, only the section containing the reflected signal is relevant to the neural network classifier, as can be seen in Figure 4.7. Also considered was to use principle component analysis to reduce the size of each observation, however this solution also suffers the same issue as the previous suggestion as redundant data is still used and fed into the network.

An third solution would be to detect the desired section of the observation and window this section, reducing the size of the input to the length of the reflected signal. An algorithm to achieve this could be written, which outputs just the desired section of each observation. This option has been chosen as it ideally will only leave useful data, and is therefore better fit for purpose than taking average values throughout the entire observation, coupled with the fact that this method could still be applied to the already-windowed observation, in order to

reduce the size further for speed optimisation.

4.4.1 Windowing Algorithm

Each of the 168 observations contains 10240 individual samples recorded by the oscilloscope. Of this data, only the section containing the initial reflected signal is relevant to the system; the majority of the data is not useful. In addition, 10240 samples per observation increases the processing required significantly and would likely hinder the performance of the system notably. Before the data can be input into the neural network, the useful section of each sample will be extracted, reducing the size of each sample and ensuring that only relevant data is used for training the system. One of the observations has been plotted and is displayed in Figure 4.7, showing how much data exists, and which data needs to be extracted.

An algorithm, displayed in Figure 7.2 was written and designed using Python. Python was chosen because it is a powerful high level language with useful features such as data structures capable of easily storing signal data as well as the csv module [53] which allows easy manipulation of comma-separated variable files. The purpose of the algorithm is to extract a window of data from each observation containing the reflected signal. An important decision was choosing the size of the window. Some messages will be longer than others; the message *XXX* will be shorter than the message *YYY* and as a result, the length of the message when reflected will vary. Across all observations, the size of the reflected signal ranges from approximately 500 values up to 750 values. To ensure all of the necessary values are extracted, the window size has been set at 768. The program scans the comma-separated file containing an individual observation for a threshold value. When the threshold value has been detected, the current value and the next 767 values are written into a new file. Multiple threshold values were tested for use in the algorithm. If the value was set too high, some of the observations of greater distances were not detected and if the value was too low, some false positives started to appear. Setting the threshold value to 0.8 enabled the algorithm to successfully window every recorded observation.

4.4.2 Network

Once the windowing algorithm has been applied to all 168 observations, each observation is imported into MATLAB as a single column vector. The observations are then collated into one 768x168 inputs matrix, where each column represents a single observation. The target data is manually created as a row vector of length 168, in which the first 21 values are the number 1, to represent code #1. The next 21 values are the number 2, and so on, up to the last 21 values which are the number 8. This in itself is not enough to use as target data in the neural network; the vector must be converted into a 8x168 matrix utilising a one-hot style encoding, so that the number 1 is represented as a column vector [10000000], the number 2 is represented by the column vector [01000000], up to the number 8 which is represented by [00000001]. To illustrate this, the class representing code #5 is shown in Figure 4.8

$$\begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

Figure 4.8: Code #5, formatted as a target class

The input data is divided into training, test and validation data at a ratio of 70:15:15. This means that 120 observations will be used to train the network, 24 observations will be used for validating the network and 24 observations will be used for testing the network. The number of input neurons is set to the number of values in each observation, 768. The number of output neurons is set to 8; the number of codes, or classes, that the data is to be classified into. The number of hidden neurons has been set to 32 as a provisional value, although there is no specific formula for calculating this value and is usually determined by testing multiple values and observing which performs best. The overall structure of the network is displayed as a diagram in Figure 4.9.

An important design consideration is deciding the activation function to be used by the neurons within the network. MATLAB offers 3 different functions: *logsig*; a log-sigmoid transfer function, *tansig*; a hyperbolic tangent sigmoid transfer function and *purelin*; a linear transfer function. *purelin* can immediately be discarded as an option as linear activation functions do not work well in classification scenarios [54]. It has been shown that *tansig* performs better than *logsig* for classification and pattern recognition tasks [55] so *tansig* has been selected for use in this network.

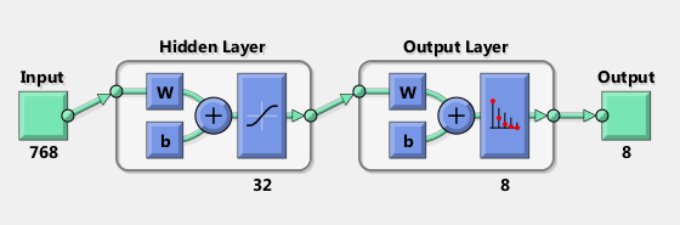


Figure 4.9: Structure of the Neural Network

4.4.3 Results

To determine the performance of the network, two methods have been used. The first is to look at how many observations in the training and validation datasets were correctly classified by the network. The second is to measure the cross entropy error.

With just one retraining of the network, it managed to correctly classify 100% of the observations. These results are displayed with a confusion matrix in Figure 4.10. This is very good as not a single observation was classified incorrectly. However, to ensure consistency, the network has been retrained a total of 5 times and the results are displayed in Table 4.2. The values calculated include the overall classification rate, the test data classification rate and the validation data classification rate. In addition to this, the number of neurons in the hidden layer has been altered, to determine the optimal number to be used in the network, comparing it to the prediction of 32 identified earlier in the section. To clarify, the network has been retrained 5 times per variation in the number of hidden neurons.

No. of hidden neurons	Avg. Overall Classification	Avg. Validation Classification	Avg. Testing Classification
8	97.5%	92.8%	90.4%
16	98.4%	95.2%	94.4%
24	98.1%	94.4%	92.8%
32	98.7%	94.4%	94.4%
64	98.0%	94.4%	92.0%
128	98.1%	94.4%	92.8%

Table 4.2: Neural Network Classification Results

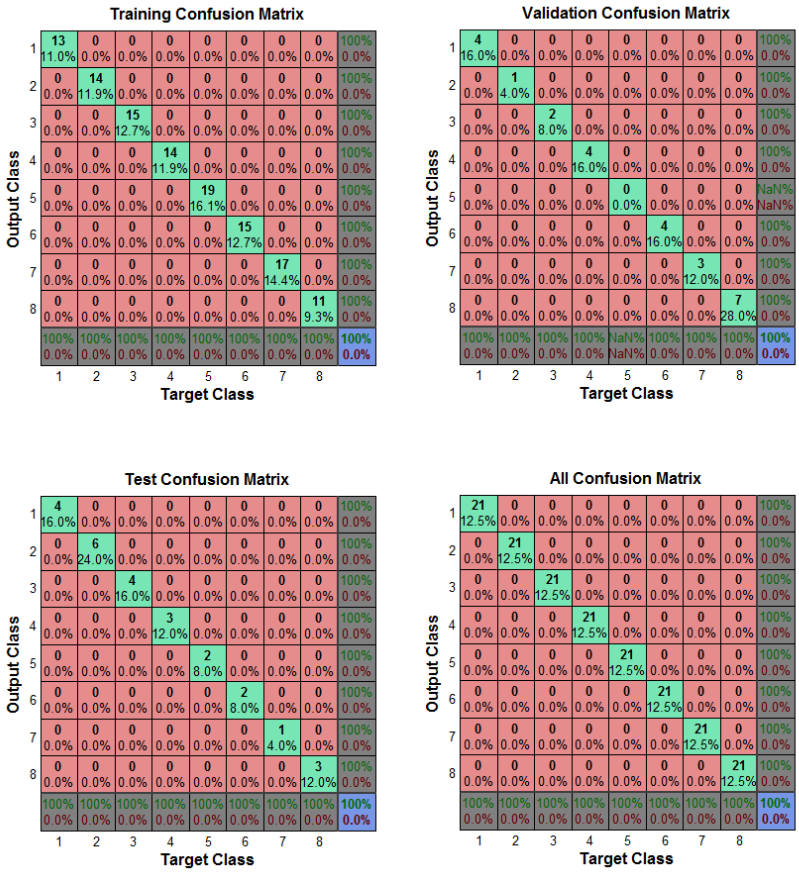


Figure 4.10: Confusion matrix displaying the number of correctly classified observations

As can be seen from the results, regardless of the number of neurons in the hidden layer of the network, the overall classification rate is consistently above the required rate of 90%, and both the validation and test data is classified at a rate of above 90% in all cases. It can therefore be stated that the network works successfully. It can also be noted that the network with 32 hidden neurons performed marginally better than the other networks, therefore validating the original prediction as a reasonable value to use.

4.5 Time-of-flight Measurement

Computing the time-of-flight for each observation was trivial. All that is required is a simple extension to the windowing algorithm. By setting a threshold for the input signal generated by the microcontroller, this can simply be compared to the start time of the reflected signal in order to calculate the difference. Since the sample rate is constant at 250kSa/s, the time-of-flight, in seconds, can be calculated by taking the difference between the sample numbers of the beginning of the reflected signal and the beginning of the input signal and dividing the result by 250,000. The updated algorithm can be found in the Appendix.

250,000 samples are taken every second, which means a sample is taken every 4 microseconds. Consequently, the maximum measurement resolution of the time-of-flight value is 4 microseconds. As the time-of-flight measurement is used to calculate the distance, it will at some stage be divided by two in order to calculate the time taken for the signal to travel in one direction. This means that the measurement resolution increases to 8 microseconds for this new value. Measurement resolution is an important measure of accuracy; the lower the

resolution, the more precision in each measurement.

4.6 Multiple Signals

In order to separate multiple signals from one observation, the windowing algorithm must be further improved. In its current state, the algorithm will recognise the first signal it finds in an observation before exiting once the signal has been windowed. This solves the problem to the extent that it can separate two signals, but only classify the first that is received. The program must be able to detect if another signal is present so that it may also be classified by the neural network. This is not straightforward and requires plenty of consideration. An artificial observation can be created by using the sum of an original observation and an already-windowed signal at some position in the original observation. An example observation containing multiple signals is displayed in Figure 4.11. The first issue is that it would be undesirable to have the algorithm identify reflected noise as signals. Reflected noise is strongest in observations where the reflective surface was positioned closest to the ultrasonic sensors; at 30cm it was strongest. In these observations, instead of looking for signals straight after the initially received signal, it would be preferable to skip forward past the reflected noise first. However, this is not the action that would ideally be taken with observations at further distances, as less reflected noise increases the chance of picking up an extra signal to identify. A solution to this issue is to use a dynamic threshold. Firstly, the observation can be identified as being at a close distance if the maximum value exceeds a certain threshold. This is trivial to check using the current algorithm as a starting point. If the maximum value is found to be high enough and therefore at a close distance, the number of values to skip can be altered accordingly. The dynamic aspect of this means that more of the available data can be used successfully.

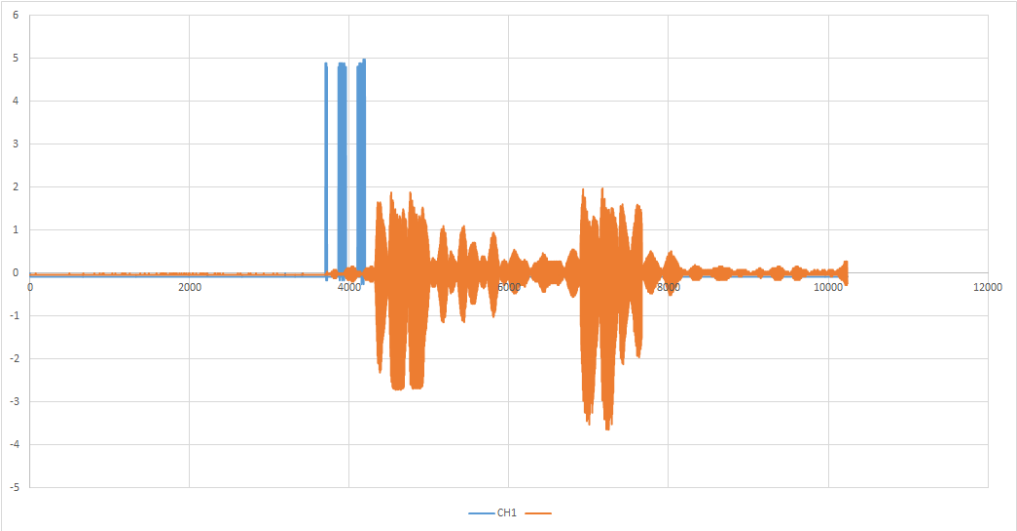


Figure 4.11: An example observation containing multiple signals

The windowing algorithm was updated to skip forward 2000 values if the identified signal was close, that is, the maxiumum value exceeded 1.4. This value was chosen by graphically observing many of the pre-existing observations and making an informed estimate as to an appropriate value. The boolean value indicating that the threshold for detecting signals is then reset and the algorithm continues searching for signals. If the maximum value does not exceed 1.4, then the signal is not reported to be close. The same boolean value is reset, but the algorithm only skips forward 1000 values, just enough in order to move past any remaining reflected noise from the original signal. This process is repeated until the end of the

observation is reached. The updated algorithm can be found in the appendix.

5 Testing and Evaluation

This chapter looks to test and evaluate the system against the requirements specified in the problem analysis.

5.1 System Testing

5.1.1 Input resolution reduction

The size inputs has been reduced in size by a varying scale factor as a means to test whether the performance of the system can be optimised further, as the faster an autonomous robot can process its sensor data, the faster it is capable of moving. As the dimensions have changed, the network has been retrained with the new, smaller dataset.

The method used to reduce the size of the inputs was to essentially resample the data. By selecting every other value, the size of the inputs is halved. By selecting every third value, the size of the inputs is reduced by a factor of 3 and this can be extended to any integer value. This is simple to achieve in MATLAB, requiring a single line of code. For each reduction in the input size, the network was retrained 5 times and the average classification errors were calculated for both validation and testing. This is to ensure the results are reliable as training the network just once will not necessarily give the most reliable results, as the network could be performing well on the training data anomalously and very poorly in subsequent trains. Table 5.1 shows the performance of the network after the reduction in the resolution has been applied to the inputs. The results indicate that the network can still perform reasonably well when the inputs are scaled down by a factor of 3, but any more and the classification performance drops below the 90% classification rate identified in the problem analysis. The conclusion that can be drawn from this is that the size of the inputs can be decreased and the network will still function, but as expected, the greater the drop in size, the bigger performance hit received.

Reduction Scale Factor	Values per Input	Avg. Validation Error	Avg. Testing Error
2	384	4 %	7.2%
3	256	11.2 %	8.8%
4	192	8%	14.4%
6	128	11.2%	18.1%
8	96	20.8%	33.5%
12	64	46.9%	54.2%

Table 5.1: Resolution Reduction Results

5.1.2 White Noise

In order to simulate electrical noise that can occur during the capture process, white noise has been artificially added to the recorded observations and fed back into the network that has already been trained. To illustrate the effects of adding white noise, the observation containing the reflected signal of code #6 has been plotted alongside the same observation with varying degrees of white noise. These graphs are displayed in the appendix.

A matrix of Gaussian white noise is generated of equivalent size to the inputs matrix. This matrix is then added to the inputs matrix and used as test data on the trained network. Table 5.2 displays the classification performance of the network in the presence of white noise of varying strength. From this table, it is clear that at -10dBW (0.1W) of white noise, the classification performance is unaffected. When the strength of the noise is increased to 2dBW, the classification performance drops below the 90% specified as a requirement in the problem analysis section. However, as can be seen in the graphs in the appendix, the very last plot is the maximum level of white noise in which the system performs to the requirement specifications and it is clear to see that this signal is very distorted compared to the original, so the conclusion can be drawn that the system can in fact successfully classify signals in the presence of white noise.

White noise power (dBW)	Classification Error (%)
-10	0
-5	0.6
-3	1.8
-2	4.8
-1	6.0
0	9.5
1	7.1
2	16.7
3	19.0

Table 5.2: Classification performance in the presence of white noise

5.1.3 Processing Multiple Signals

To test whether the system is able to separate multiple signals from one observation, artificial observations have been fed into the windowing algorithm. These artificial observations have been created by taking one of the original observations and adding different codes at various locations in the signal, in order to simulate a situation where multiple signals are recorded in one observation.

For the first test, code #4 at 40cm will be altered by adding code #6 at 40cm (windowed) at 5 separate locations in the signal. The algorithm will then be tested to see if it will identify both signals. The windowed signal has been added starting at the value 5000, 6000, 7000, 8000 and 9000 in the original observation, which contains 10240 values. The results are displayed in Table 5.3. A success is considered as a case where exactly two signals are output by the algorithm. If one signal is output, the algorithm did not successfully identify the additional signal. If more than two signals are output, the algorithm picked up noise and identified it as a signal. It should be noted that the algorithm is not expected to successfully window the signal added at the 5000th value, as the algorithm should have skipped forward past this point when windowing the first signal.

As can be seen from the results, the algorithm works successfully for these observations, and the prediction about the first observation was correct. In theory, this algorithm could be extended to a real-time system that is not just looking at a set of 10240 values, but continuously taking readings from an ADC.

Start Value	Success? (Y/N)	Reason for Failure
5000	N	1 signal detected
6000	Y	N/A
7000	Y	N/A
8000	Y	N/A
9000	Y	N/A

Table 5.3: Detection of multiple signals results

The same test was then run with code #7 at 60cm as the base observation, with code #5 added at the same regular intervals. As the base observation was recorded at a further distance than the previous test, there is likely to be interference between the two signals, which could cause issues in the detection. The results of this test are displayed in Table 5.4. As can be seen, this test was less successful. It can be expected that if the two signals are at a similar location in the observation, that the algorithm will detect the first, then skip past the second. However, with the signal placed at value 7000, the constructive interference between the two signals has created noise with a high enough power to be picked up as a signal itself, indicating that the algorithm has not been successful, and may need altering in order to correctly separate a greater number of examples than it currently is able to.

Start Value	Success? (Y/N)	Reason for Failure
5000	N	1 signal detected
6000	N	1 signal detected
7000	N	3 signals detected (noise picked up as signal)
8000	Y	N/A
9000	Y	N/A

Table 5.4: Detection of multiple signals results

5.2 Requirements Testing

Chapter 3 identified a set of tests to evaluate the success of the system. The first requirement was that the system must be able to classify 90% of observations. This requirement has been met, as evidenced in Chapter 4, Table 4.2.

The second requirement was that the system must be able to identify the time of flight of each observations. This requirement has also been met, as the windowing algorithm was run on all 168 observations and a time of flight value was returned. The time of flight for secondary signals in observations containing multiple signals cannot be calculated, but this is impossible, as the observation is created artificially and therefore the signal's origin is unknown.

The third requirement was that the system must correctly separate the reflected signal from the rest of the values within each observation. Again, the windowing algorithm was run on all 168 observations and the reflected signal was successfully windowed.

The fourth requirement was that the system must be able to correctly classify observations in the presence of white noise. The power of white noise that the system could withstand, while still maintaining a 90% classification rate was 1dBW as seen in Table 5.2. This requirement has

therefore deemed to have been met, as 1dBW leaves a very distorted signal, but the network can still successfully classify nearly all observations subject to this level of noise.

The fifth and final requirement was that the system must be able to classify observations where multiple signals are present. This was not successfully met, as the results of the tests displayed in Table 5.3 and 5.4 indicate. Much more work is needed to improve the algorithm to enable this feature to function correctly

6 Conclusions and Further Work

This chapter attempts to make conclusions about the project and explain whether it was successful. Afterwards, possible further work is outlined, suggesting ways in which the research started in this project could be continued.

6.1 Conclusions

The aims of this project were to develop a system capable of encoding ultrasonic signals with specific patterns, and subsequently decoding these patterns, in order to improve the reliability of the ultrasonic data received. Once the system was found to work, the next step was attempting to minimise the processing power required for the system to run, in order to maximise performance.

The generation of signal codes, performed by the Arduino microcontroller, worked without issue. A program was written to generate custom codes and 8 separate codes were generated successfully. Some of the parameters such as the spacing between bursts of pulses were determined through trial and error, leading to the conclusion that there may be space for optimisation. With more research and testing, the codes could be designed to be easier identify. In turn, this could improve the classification rate of the system.

The next issue was identifying a signal from an observation, so that it could then be classified. A windowing algorithm was written to separate the signals from each observation. This worked without issue for all 168 observations. Another key aim of the project was to identify the time-of-flight of each signal, a solution for which was integrated into the windowing algorithm successfully. The problem then with the algorithm was that it only identified the first signal it found, and any signals found after would be ignored. The algorithm was further updated to enable classification of multiple signals, which worked with limited success. Not every possible combination of signals summed together could be used to test the algorithm, but of the combinations used, the majority were identified successfully. The time-of-flight could be calculated for the original signal in the observation, but the system had no means to know the origin of the extra signals added artificially.

Once the signals had been identified, they were fed into the heart of the system: the neural network. The network correctly classified the majority of recorded observations. After reducing the resolution of each observation in order to improve performance, the network still managed to classify 90 % of observations when the size of each input was reduced by a factor of 3. When subject to white noise, the classification results were also impressive. At 1dBW of artificially added white noise, the network was still capable of classifying 92.9% of observations, implying that the system could perform well, even when subject to some levels electrical noise, although when the noise level was increased to 2 and 3dBW, the classification performance dropped below the 90% set out in the problem analysis. The classification rate at 3dBW was 81%, while below the desired rate, the network still classified a high proportion of signals, and if better trained, this rate could definitely be improved.

The main area where the project failed was the classification of observations containing multiple signals. Although just one part of the project, one of the key issues this project attempted to solve was the detrimental effects of crosstalk scenarios. The algorithm almost works without issue, but a few tweaks will be required for it to work as intended.

The network was trained with a limited amount of data, as only a certain amount of data could be recorded given the time constraints of the project. Consequently, the network does not have a good level of generalisation, and observations at other distances than the training data are unlikely to be classified correctly. To improve the classification rate, much more data is required. In conclusion, the project can be considered a partial success, and given more time, the project could be developed into a very good system, as many of the key features of the project worked almost perfectly.

6.2 Further Work

There are many ways in which the work started by this project can be taken further. The current system cannot actually be used with a robot as it is eventually designed to be. This section explores some of these ideas.

6.2.1 Robot Integration

This project is designed to offer a novel solution for reducing the affects of crosstalk in an autonomous robot system. Although this project does not implement these features in tandem with a physical robot, it is the ultimate goal for a robot to be built that implements a system akin to that which has been demonstrated here. The first step on the path to achieving this is to integrate both the transmitting and receiving transducers to a microcontroller. As outlined in the problem analysis section, the system is designed to work with 8 transmitters however with a small amount of editing to the code generation program, this could be scaled to any desired size. One or more receivers will also be required, but there is no need to use 8, as a functioning system should be able to identify the transmitter of a particular signal. Two receivers, one to cover the front of the robot and one to cover the rear, should suffice. An high performance ADC will also need to be connected to the microcontroller, with a sample rate of at the very least 80kSa/s, although ideally higher to ensure consistent performance.

The system should also have the processing power to run the neural network software. As observed in chapter 4, there are alternatives to the MATLAB Neural Network Toolbox, that are less resource-intensive, and alternative microcontrollers with much higher processing power possibly able to cope with the demands of a neural network implementation. If this were not feesible using current technology, an alternate solution may still run the network on a high performance computer, but the robot could nterface with the computer via a wireless link such as bluetooth, in order to send and receive data. The primary advantage to this would be that the processing power of a full computer would be available to the system, while remaining free to navigate and travel around a location. However, the obvious downside is that the computer would need to remain running continuously during the robot's operation, and the navigation range would be limited to the range of the wireless link. Finally, the system needs the means to move around, creating the need for wheels, motors and software to drive these components.

6.2.2 Neural Network

Although the neural network implemented in this project was shown to be reasonably successful, there is a lot more that can be done to improve its performance. The first major point is that not enough training data was provided to the system. Much more training data should be recorded, in a variety of different scenarios, in order to improve the generalisation, and therefore performance and robustness, of the system.

Another potential improvement on the current system would be to extend the neural network to intelligently learn commands to control the robot based on the sensor readings. This would

give the robot a true sense of autonomy, because once trained, the robot would not have to rely on a simple algorithm, but a more complex understanding of its surroundings.

Finally, the network could be altered to classify reflected signals previously ignored by the system. This would benefit the system as these reflected signals do contain useful data, and this data could be used to add more information about the origins of signals and consequently the distance to objects surrounding the robot.

7 Appendix

```
#define outputPin 7
#define waveFrequency 40000
#define pulseLength 30
#define noPulses 4
#define pulseGap 500
#define refreshRate 500

void setup() {
  Serial.begin(9600);
  pinMode(outputPin, OUTPUT);
}

int calculateDelay(int x) {
  if (x == 4) {
    return 29;
  }
  else if (x == 16) {
    return 165;
  }
}

void generateCode(int a, int b, int c) {

  int values[] = {a, b, c};

   for (int i=0; i < 3; i++) {
     tone(outputPin, waveFrequency);
     delayMicroseconds(calculateDelay(values[i]));
     noTone(outputPin);
     delayMicroseconds(pulseGap);
   }

}

void loop(){
  generateCode(16,16,16);
  delay(refreshRate);
}
```

Figure 7.1: Arduino program to generate custom signal codes (Code #8)

```
import csv
from Tkinter import Tk
from tkFileDialog import askopenfilename

fileTypeName = "Comma-separated Variable"
extension = ".csv"
threshold = 0.8
threshold_met = False
current_val = 0
endex = 10000
window_size = 768
window = []

Tk().withdraw()
# Let user browse for input file
filename = askopenfilename(filetypes=[(fileTypeName, extension), ("All", "*")])
# Set new file name
newfilename = filename[:-4] + "_windowed" + extension


with open(filename, 'rb') as csvfile:
    sample_reader = csv.reader(csvfile, quotechar='|')
    index = 0
    for row in sample_reader:
        # Start checking after headers
        if (index >= 2):
            # Set current value
            current_val = float(row[2])
        if not threshold_met:
            # Compare current value to threshold
            if (current_val >= threshold):
                threshold_met = True
                # Set index for end of window
                endex = index + window_size - 1
        if (threshold_met):
            # Add current value to window list
            window.append(row[2])
            print str(index) + ": " + row[2]
        # If end of window reached, end loop
        if (index == endex):
            break
        index += 1

with open(newfilename, 'wb') as csvfile2:
    sample_writer = csv.writer(csvfile2)
    # Iterate through window list and add write each value to new file
    for x in window:
        sample_writer.writerow([x])
```

Figure 7.2: Windowing algorithm written in Python

```
import csv
from Tkinter import Tk
from tkFileDialog import askopenfilename

fileTypeName = "Comma-separated Variable"
extension = ".csv"
input_threshold = 4
input_threshold_met = False
threshold = 0.8
threshold_met = False
current_val = 0
current_input = 0
endex = 10000
window_size = 768
window = []

Tk().withdraw()
# Let user browse for input file
filename = askopenfilename(filetypes=[(fileTypeName, extension), ("All", "*")])
# Set new file name
newfilename = filename[:-4] + "_windowed2" + extension


with open(filename, 'rb') as csvfile:
    sample_reader = csv.reader(csvfile, quotechar='|')
    index = 0
    for row in sample_reader:
        # Start checking after headers
        if (index >= 2):
            # Set current value
            current_val = float(row[2])
            current_input = float(row[1])
        # Check if input signal threshold met
        if not input_threshold_met:
            # If input signal detected, set input_start to current index
            if (current_input >= input_threshold):
                input_threshold_met = True
                input_start = index
        if not threshold_met:
            # Compare current value to threshold
            if (current_val >= threshold):
                threshold_met = True
                # Set index for end of window
                endex = index + window_size - 1
                # Calculate time-of-flight in seconds
                time_of_flight = float(index - input_start) / 250000
        if (threshold_met):
            # Add current value to window list
            window.append(row[2])
            print str(index) + ": " + row[2]
        # If end of window reached, end loop
        if (index == endex):
            print "Time of flight: " + str(time_of_flight) + " seconds"
            break
        index += 1

with open(newfilename, 'wb') as csvfile2:
    sample_writer = csv.writer(csvfile2)
    # Iterate through window list and add write each value to new file
    for x in window:
        sample_writer.writerow([x])
```

Figure 7.3: Updated windowing algorithm enabling time-of-flight calculation

```
import csv
from Tkinter import Tk
from tkFileDialog import askopenfilename

fileTypeName = "Comma-separated Variable"
extension = ".csv"
input_threshold = 4
input_start = 100000
input_threshold_met = False
threshold = 0.8
threshold_met = False
short_distance = False
current_val = 0
maximum_val = 0
current_input = 0
no_of_signals = 0
endex = 100000
window_size = 768
writers = []
writer_no = 0
window = []

Tk().withdraw()
# Let user browse for input file
filename = askopenfilename(filetypes=[(fileTypeName, extension), ("All", "*")])
# Set new file name
newfilename = filename[:-4] + "_windowed_0" + extension


def writeFile():
    current_csv_writer = open(newfilename, 'wb')
    sample_writer = csv.writer(current_csv_writer)
    # Iterate through window list and add write each value to new file
    for x in window:
        sample_writer.writerow([x])
    current_csv_writer.close
```

Figure 7.4: Updated windowing algorithm enabling multiple signal detection (lines 0-37)

```
with open(filename, 'rb') as csvfile:
    sample_reader = csv.reader(csvfile, quotechar='|')
    index = 0
    for row in sample_reader:
        # Start checking after headers
        if (index >= 2):
            # Set current value
            current_val = float(row[2])
            current_input = float(row[1])
        # Check if input signal threshold met
        if not input_threshold_met:
            # If input signal detected, set input_start to current index
            if (current_input >= input_threshold):
                input_threshold_met = True
                input_start = index
        if not threshold_met:
            # Compare current value to threshold
            if (current_val >= threshold):
                threshold_met = True
                # Set index for end of window
                endex = index + window_size - 1
                # Calculate time-of-flight in seconds
                time_of_flight = float(index - input_start) / 250000
                # Indicate new signal found and edit filename
                no_of_signals += 1
                newfilename = newfilename[:-6] + "_" + str(no_of_signals) + extension
        if (threshold_met):
            # Add current value to window list
            window.append(row[2])
            print str(index) + ": " + row[2]
        ''' If end of window reached, reset threshold_met, write file, print
        time-of-flight of signal, clear window and skip forward dynamically '''
        if (index == endex):
            writeFile()
            print "Time of flight: " + str(time_of_flight) + " seconds"
            threshold_met = False
            window = []
            if (short_distance):
                # Skip through 2000 values
                for i in range(2000):
                    sample_reader.next()
                index += 2000
            else:
                # Skip through 1000 values
                for i in range(1000):
                    sample_reader.next()
                index += 1000
        # Update maximum value
        if (current_val > maximum_val):
            max_val = current_val
        # If max value great enough, set short_distance to True
        if (maximum_val > 1.4):
            short_distance = True
        index += 1
```

Figure 7.5: Updated windowing algorithm enabling multiple signal detection (lines 38+)

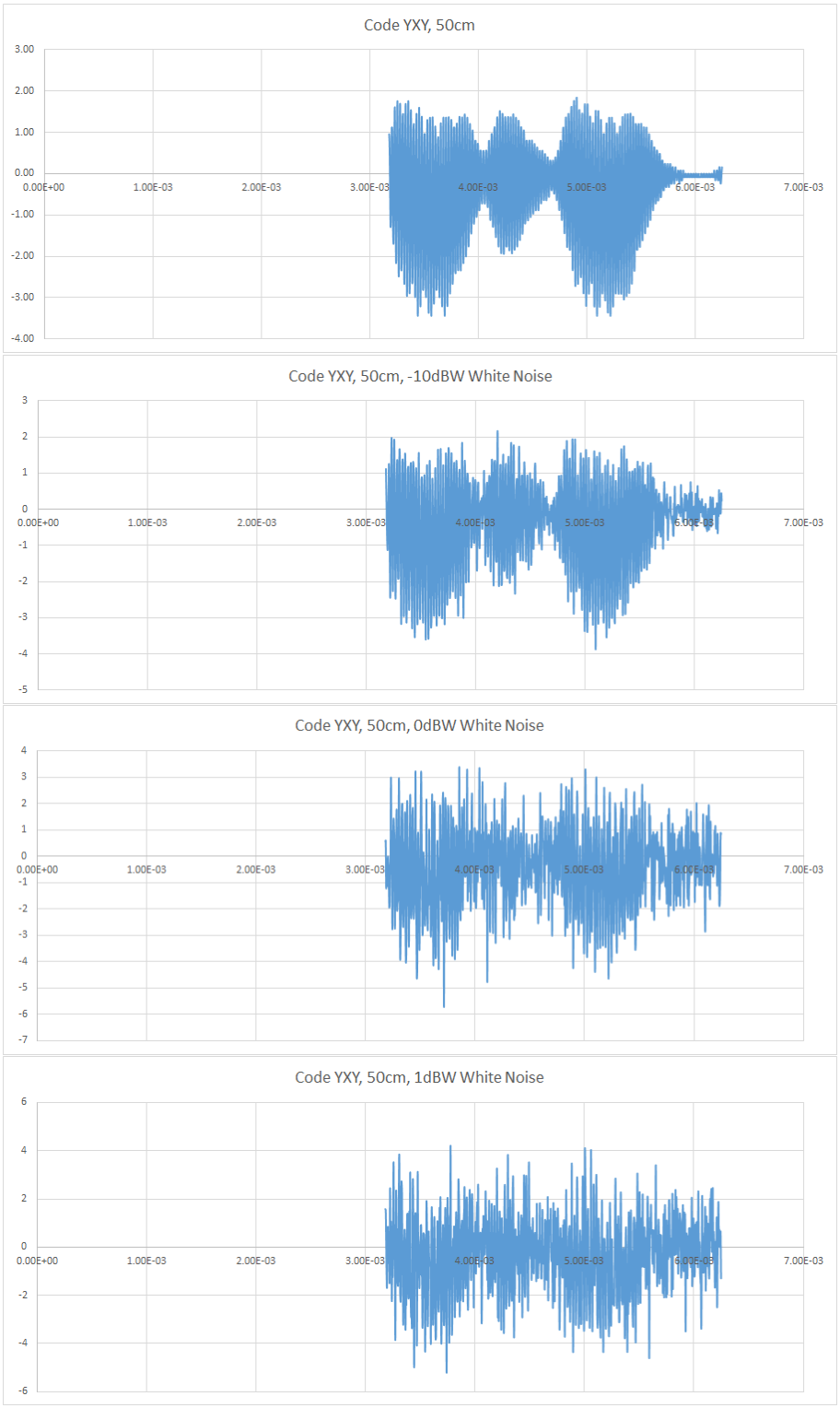


Figure 7.6: Code YXY at 50cm, with varying levels of white noise

Bibliography

[1] "Introduction," in *Understanding GPS: Principles and Applications*, 2nd ed., E. Kaplan and C. Hegarty, Eds. Boston: Artech House Publishers, 12 2005.

[2] L. Chrisman, "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach," *AAA*, pp. 183–188, 1992.

[3] E. Tjandranegara, "Distance estimation algorithm for stereo pair images," 2005.

[4] J. Mrovlje and D. Vrančić, "Distance measuring based on stereoscopic pictures," *9th International PhD Workshop on Systems and Control*, 2008.

[5] C. Facchinetti and H. Hügli, "Using and learning vision-based self-positioning for autonomous robot navigation," *ICARCV*, pp. 1694–1698, 1994.

[6] Sharp infrared ranger comparison. [Online]. Available: https://acroname.com/articles/sharp-infrared-ranger-comparison

[7] Sharp, "Optoelectronic device," *GP2YoA21YKoF*, 2006. [Online]. Available: http://www.sharpsma.com/webfm_send/1489

[8] G. Benet, F. Blanes, J. Simó, and P. Pérez, "Using infrared sensors for distance measurement in mobile robots," *Robotics and Autonomous Systems*, vol. 40, no. 4, pp. 255–266, 2002.

[9] T. Mohammad, "Using ultrasonic and infrared sensors for distance measurement," *World Academy of Science, Engineering and Technology*, vol. 51, pp. 293–299, 2009.

[10] A. P. Corrado, S. W. Decker, and P. K. Benbow, "Automotive occupant sensor system and method of operation by sensor fusion," Jan. 9 1996, uS Patent 5,482,314.

[11] M. Kam, X. Zhu, and P. Kalata, "Sensor fusion for mobile robot navigation," vol. 85, no. 1, pp. 108–119, 1997.

[12] J. Guivant, E. Nebot, and S. Baiker, "Autonomous navigation and map building using laser range sensors in outdoor applications," *Journal of robotic systems*, vol. 17, no. 10, pp. 565–583, 2000.

[13] J. Ring, "The laser in astronomy," *New Scientist*, vol. 18, p. 344, 1963.

[14] How lidar works. [Online]. Available: http://www.lidar-uk.com/how-lidar-works/

[15] "Mercedes benz parktronic system (pts)." [Online]. Available: http://www.micro-tronik.com/Support/Info/Benz-PTS.htm

[16] W. D. Rencken, "Concurrent localisation and map building for mobile robots using ultrasonic sensors," in *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*, vol. 3. IEEE, 1993, pp. 2192–2197.

[17] D. Pagac, E. M. Nebot, and H. Durrant-Whyte, "An evidential approach to map-building for autonomous vehicles," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 4, pp. 623–629, 1998.

[18] K. Yasuda, H. Mukai, S. Fujimoto, M. Nakajima, and K. Kawai, "The diagnosis of pancreatic cancer by endoscopic ultrasonography," *Gastrointestinal endoscopy*, vol. 34, no. 1, pp. 1–8, 1988.

[19] M. I. Haller and B. T. Khuri-Yakub, "A surface micromachined electrostatic ultrasonic air transducer," *Proceedings of IEEE Ultrasonics Symposium ULTSYM-94*, 1994.

[20] O. Oralkan, A. Ergun, J. Johnson, M. Karaman, U. Demirci, K. Kaviani, T. Lee, and B. Khuri-Yakub, "Capacitive micromachined ultrasonic transducers: next-generation arrays for acoustic imaging?" *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 49, no. 11, pp. 1596–1610, 2002.

[21] D. P. Massa, "Choosing an ultrasonic sensor for proximity or distance measurement," *Sensors*, vol. 16, no. 2, 1999.

[22] MaxBotix, "High resolution, precision, low voltage ultrasonic range finder," *HRLV-EZo*, 2014. [Online]. Available: http://www.maxbotix.com/documents/HRLV-MaxSonar-EZ_Datasheet.pdf

[23] Hc-sr04 ultrasonic range finder. [Online]. Available: http://www.micropik.com/PDF/HCSR04.pdf

[24] J. LeClare. (2003, Jun. 27) A simple ADC comparison matrix. [Online]. Available: http://pdfserv.maximintegrated.com/en/an/AN2094.pdf

[25] W. Kester, "Which adc architecture is right for your application?" [Online]. Available: http://www.analog.com/library/analogDialogue/archives/39-06/architecture.pdf

[26] H. Nyquist, "Certain topics in telegraph transmission theory," *Transactions of the American Institute of Electrical Engineers*, vol. 47, no. 2, pp. 617–644, 1928.

[27] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[28] J. Jimenez, M. Mazo, J. Urena, A. Hernandez, F. Alvarez, J. Garcia, and E. Santiso, "Using pca in time-of-flight vectors for reflector recognition and 3-d localization," *IEEE TRANSACTIONS ON ROBOTICS*, vol. 21, no. 5, pp. 909–924, 2005.

[29] B. Smagowska, "Ultrasonic noise sources in a work environment," *Archives of Acoustics*, vol. 38, 2013.

[30] G. L. Stüber, "Co-channel interference," in *Principles of Mobile Communication*, 3rd ed. Springer, 2012.

[31] H. Nyquist, "Thermal agitation of electric charge in conductors," *Physical Review*, vol. 32, no. 1, pp. 110–113, 1928.

[32] G. Ponuratinam, B. Patel, S. S. Rizvi, and K. M. Elleithy, "Improvement in the spread spectrum system in dsss, fhss, and cdma," 2009.

[33] M. Saad, C. Bleakley, and S. Dobson, "Robust high-accuracy ultrasonic range measurement system," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 10, pp. 3334–3341, 2011.

[34] R. Scholtz, "The origins of spread-spectrum communications," *IEEE Transactions on Communications*, vol. 30, no. 5, pp. 822–854, 1982.

[35] M. Hazas and A. Ward, "A novel broadband ultrasonic location system," *UbiComp 2002: Ubiquitous Computing*, pp. 264–280, 2002.

[36] D. J. MacKay, *Information theory, inference, and learning algorithms*, 1st ed. Cambridge, UK: Cambridge University Press, 09 2003.

[37] S. Gorn, R. W. Bemer, and J. Green, "American standard code for information interchange," *Communications of the ACM*, vol. 6, no. 8, pp. 422–426, 1963.

[38] S. Shoval and J. Borenstein, "Using coded signals to benefit from ultrasonic sensor crosstalk in mobile robot obstacle avoidance," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, 2001.

[39] S. Lawrence, C. Giles, A. C. Tsoi, and A. Back, "Face recognition: a convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[40] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[41] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[42] How do mlps compare with rbfs? [Online]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-14.html

[43] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[44] S. Kim and H. B. Kim, "High resolution mobile robot obstacle detection using low directivity ultrasonic sensor ring," *Lecture Notes in Computer Science*, pp. 426–433, 2010.

[45] "Arduino duemilanove." [Online]. Available: http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove

[46] J. Belk, "The first four partial sums of the fourier series for a square wave." [Online]. Available: http://commons.wikimedia.org/wiki/File:Fourier_Series.svg

[47] Arduino reference: Tone. [Online]. Available: http://www.arduino.cc/en/Reference/tone

[48] Dso1000a/b series portable oscilloscopes. [Online]. Available: http://literature.cdn.keysight.com/litweb/pdf/5989-9368EN.pdf

[49] *MATLAB Release 2014a*, The MathWorks, Massachusetts, United States, 2014. [Online]. Available: http://uk.mathworks.com/products/matlab/

[50] *Neural Network Toolbox*, The MathWorks, Massachusetts, United States, 2014. [Online]. Available: http://uk.mathworks.com/products/neural-network/

[51] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "PyBrain," *Journal of Machine Learning Research*, 2010.

[52] Raspberry pi 2 model b. [Online]. Available: http://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[53] Csv file reading and writing — python 2.7.9 documentation. [Online]. Available: https://docs.python.org/2/library/csv.html

[54] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. United States: Cambridge University Press, 2010.

[55] A. Choudhary, R. Rishi, S. Ahlawat, and V. S. Dhaka, "Performance analysis of feed forward mlp with various activation functions for handwritten numerals recognition," *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, 2010.