An Embedded Machine Learning Approach to Assist Navigation for People with Visual Impairments

Seán Ó Fithcheallaigh[1], Chris Nugent[1], Jun Liu[1] and Ian Cleland[1]

Department of Computing, Ulster University, Belfast, N. Ireland,
o_fithcheallaigh-s@ulster.ac.uk

Abstract. An ever-increasing number of people are living with visual impairments. As the machine learning techniques evolve alongside improving hardware available on embedded devices, there exists the potential to develop a system which can detect and localise objects in an indoor setting. This system would aim to detect objects and localise them, thereby allowing the user to navigate around these obstacles in an unfamiliar environment. The work presented here show the initial investigations into the development of such a system. The information presented will cover the investigation of a number of machine learning techniques as well as the deployment of a model onto a constrained device.

Keywords: Machine Learning, Deep Learning, Embedded Systems, The Edge, Neural Networks, Arduino

1 Introduction

1.1 The lives of people dealing with sight loss

There are over two million people in the UK who live with sight loss; where sight loss includes: people who are registered blind or partially sighted; people whose vision is better than the levels that qualify for registration; people who are awaiting or having treatment such as injections, laser treatment or surgery that may improve their sight; Correctly prescribed glasses or contact lenses could improve the sight of people who have sight loss [1]. Experts also predict that the number of people suffering from sight loss will double to over four million by 2050 [2]. The main reason for sight loss includes age-related macular degeneration (AMD), uncorrected refractive errors and cataracts [1].

Individuals who have visual impairments tend to avoid unfamiliar environments, which can have adverse effects on their overall health and wellness. Obstacle detection and warning can improve the mobility and safety of visually impaired people, particularly in unfamiliar environments, which can help facilitate the journeys they need to do. However, transport systems are not built with the visually impaired in mind [2]. It is likely that this contributes to the

limitation of four out of every ten blind individuals who can only complete some of the journeys they require or desire. [1].

Visually impaired people face many issues when navigating their journey, such as understanding how to reach their destination. Typically, the route to a destination will stay the same - streets remain in the same place, and road crossings tend not to move. However, another issue they face is random obstacles placed in their path. These are things that a visually impaired person could have no way of knowing are there. Of course, aids such as the white cane can be used to assist with this kind of issue. However, not everyone may want to use a cane (potentially because of the signal that sends: I am different, I have an impairment). Another reason could be because, although a person may be registered as blind, they may still have some level of vision (as many do) and do not wish to give up a level of independence completely.

1.2 The Problem

A need exists to develop systems that can assist visually impaired people in navigating their surroundings. In any proposed navigation system for the visually impaired, obstacles must first be detected and localised. Then, navigation information must be communicated to the person, allowing them to avoid obstacles. Using various modalities such as voice, tactile feedback, and vibration could facilitate the achievement of this goal.

This research proposes using machine learning methods within a constrained device to develop a solution that can detect obstacles in the path of a visually impaired user navigating an indoor environment.

A constrained device is something that works at "the Edge". Devices that work at the Edge will typically do any data processing and analysis on the device itself. This allows for a reduction in the time it takes for a system to get a result. This reduction in time, or latency, is critical in the system this research aims to investigate. Power is another important constraint when dealing with Edge devices, and is obviously very important when developing a portable navigation system, as it will need to run under its own power. Memory is another important consideration when working at the Edge. This means that any proposed model must be able to be stored in the device's memory, and still be able to take in sensor data for processing.

Several tasks will need to be completed to determine the system's feasibility. The first step will be the investigation of various sensor modalities in order to determine the most appropriate sensor or combination of sensors. Then a dataset will be gathered covering a range of obstacle detection and avoidance scenarios. This dataset will then be used to train, test, and validate several Deep Learning and machine learning models to understand which is best at detecting and localising obstacles. Development work will then be done to allow this model to be implemented on a constrained device, and the performance of the final model will be assessed against critical parameters.

1.3 Paper Structure

The paper will present information on the technical aspects of machine learning at the Edge, as well as work on related research. There follows a discussion on data capture and the initial analysis that was carried out using classification algorithms. From there the discussion moves on to the investigation carried out on neural networks and the deployment of the system on a constrained device. The final sections will be an evaluation of the deployed model and a discussion on potential future work.

2 Technical Review

2.1 Introduction

As has been discussed, the goal is to design and build an object detection system which researchers can implement on a constrained device. This detection system will operate at what is called "the Edge". In recent years, the term "Edge" has been more common in discussions about the future directions of machine learning, and is often discussed in connection with the Internet of Things (IoT). When discussing machine learning at the Edge, one may hear terms such as "Edge AI" or "Edge ML".

What is the Edge and how does it relate to machine leaning? When discussing the internet, computers, or IT systems, most people will have an image of their PC at home or the computer they use at work. However, there are more devices connected to the internet than computers. As of 2021, research shows there were 12.2 billion active IoT connections [3]. These IoT devices cover almost any aspect of our lives that one cares to think about, everything from smartwatches; intelligent kitchen appliances; baby monitors connected to the internet, allowing parents to check in from anywhere in the world; shipping containers; and industrial sensors used to monitor the health of machinery. The list goes on and on.

How are all these billions of devices connecting to networks and communication? They are connected to servers, and these servers are often referred to as the "cloud". These devices are connected to a network; they take readings from their sensors and send that information to a location where it can be stored and processed. From this perspective, these devices sit at the 'Edge' of the network, hence the name.

Edge AI. For a long time, IoT devices have been seen as ways to collect data via onboard sensors. They would collect the data and transmit it back to a hub for processing. However, this approach is costly in several ways. First, it can cost a lot of money to transmit large amounts of data, due mainly to the connectivity and storage costs; there is also the issue that transmitting data is a highly power-hungry task for any battery-powered IoT device. Second, it is expensive in human time, too, because people will need to evaluate the data and process it, potentially making some decisions based on that data analysis.

Sending information back to a central location to be processed and then returning the result can take time, referred to as latency. The time required for this may only be a matter of seconds, which is acceptable for some applications, but would be too long for other applications where near-instant feedback is needed. An example of a time-critical system could be a system on an autonomous vehicle needing to react to traffic lights. The vehicle cannot wait a few seconds to determine if it is approaching a red light and needs to stop.

While a lot of high-end Edge AI devices have been developed, lower-end systems are still capable of carrying out AI applications. One example of a lower-end system is the Beaglebone AI [5]. This system is built on Linux, and based around the Texas Instruments (TI) AM5729, with a TI C66x floating point DSP, with TI embedded vision engines (EVE) [6]. Then there is the Arduino Nano 33 BLE Sense which is a small development board which is AI-enabled. The Arduino Nano 33 contains a number of onboard sensors, such as a 9-axis inertial sensor, humidity sensor, microphone and proximity sensor. The Arduino board also has GPIO pins which can allow extra sensors to be added if needed or required. The Arduino development kit allows for embedded machine learning applications to be run where models created using TensorFlow Lite can be uploaded to the board via the Arduino IDE [7].

Software and Frameworks. As previously discussed, when building Edge AI applications on an MCU, one must give the availability of memory due consideration. Unfortunately, available memory is problematic when implementing any machine learning algorithm or neural network on an MCU because the weights and biases required for a model to function and any data points can take up large amounts of memory. Software libraries and frameworks have been developed to help reduce things like memory requirements.

The main tools or frameworks used in deep learning would be TensorFlow and its associated Keras API and PyTorch. TensorFlow, developed by Google Brain, is a powerful open-source library aimed at deep neural networks. These tools are typically optimised to run on GPUs and other specialised hardware. The reason for this is to accelerate the training processes. Once a model is trained, using that model to make inferences is less computationally expensive, which is essential for dealing with Edge devices. However, it is important to note that why the inference is less computationally expensive, the MCU does not have a free ride - a reasonable amount of memory and processing power is still required to carry out the inference. To try and reduce the requirements on the front end, frameworks such as TensorFlow and PyTorch have implemented model compression techniques with little to no effect on computational accuracy [6].

While TensorFlow (with TensorFlow Lite) and PyTorch have been mentioned here, TensorFlow has been the main focus, because while PyTorch has PyTorch Mobile, it is aimed at mobile devices which run on either Android or iOS, whereas TensorFlow Lite can be used on embedded system [6] [8].

TensorFlow Lite (TLF) is a set of tools that have been designed and built to allow on-board machine learning. It has been designed to help developers run models on embedded systems, as well as on mobile devices [8].

Some of the key features of TensorFlow Lite (TFL) include:

- Optimisation
- Multi-platform
- Multi-language options
- High performance

Process optimisation is extremely important when developing embedded machine learning systems. TFL deals with optimisation by dealing with a number of key constraints which are common to embedded systems: latency, privacy, connectivity, size, and power consumption [4]. The reduced latency comes from the fact that there is no information going to the server to be processed, and then back to the device. TFL removes a lot of privacy issues which may traditionally have been present in cloud-based machine learning tasks because no data will actually leave the device. Connectivity issues are side-stepped because the device does not need to connect to a network. Some systems may send information back to a cloud-based server, but this would become a non-critical task, in terms of timing. TFL can also reduce the size of any associated files, meaning they are more suitable for embedded applications. Power consumption is reduced through efficiencies in the interface, as well as the lack of network connection, since, typically, sending information tends to be a large drain on power.

Optimisation. Various optimisation techniques can be applied to models to allow efficient operation with the limited computational power and limited memory available on a constrained device. As previously discussed, some optimisation techniques allow for the use of accelerated inference [9].

Models should be optimised for a number of reasons. One of the main reasons would be size reduction. There is no single method used for reducing the model size, but no matter the process, there are a number of advantages in doing this:

- **Smaller storage size**: A smaller size model will take up less memory on the end device. If we take the Arduino Nano as our target device, this MCU does not come with a lot of onboard memory, so reducing the model size is critical for deployment.
- **Reduced memory usage**: Using a smaller model will result in less RAM being used when the model is run. This will allow other more memory to be available for other elements of the application to use. This will typically result in better performance overall.
- **Reduced download size**: A smaller model will need less time to download onto the end device. his is a benefit when a system might have a small window of time to receive a newly trained model.

Another reason is **latency reduction**. In the context of embedded machine learning, latency can be defined as the amount of time taken for the system to

process a piece of data and make a decision based on that data. Optimisation can reduce the amount of computation needed to make this decision (or, to put it another way, reduce latency). Lower latency will also have the benefit of reducing power consumption.

Then we have **accelerators**. One consequence of model optimisation is that it can impact the model's accuracy, and this should be kept in mind during development. TFL offers tools such as Edge TPU, which can run inference extremely fast when using models which have been correctly optimised. Edge TPU is offered by Google, and is a purpose-built ASIC which has been designed specifically to run AI on Edge devices, allows for high-quality machine learning inferencing at the Edge and is based around the Carol development environment discussed above [10].

As discussed, optimisation has the potential to modify the accuracy of the model, and any developer must keep this in mind during the development process. The potential inaccuracies cannot be predicted, so a method of dealing with them cannot be developed and included as part of the TFL suite of tools.

Depending on what works best for a given application, TFL offers a number of optimisation paths: quantisation, pruning, and clustering. The quantisation approach works by reducing the precision of the numbers used to represent the model parameters [9]. By default, these are set to 32-bit floating point numbers. Reducing the precision of the numbers or parameters of the model means that less memory is required, this in turn means an overall smaller model, which will result in faster computation.

From [8], we get the quantisation techniques which are, with their benefits are:

- **Post training float 16 quantisation**: This will offer a size reduction of up to 50 per cent with insignificant accuracy loss. This technique is supported on CPUs and GPUs
- **Post training dynamic range quantisation**: Will offer up to 75 per cent size reduction with the smallest accuracy loss. This technique will work on CPUs and GPUs (Andriod)
- **Post training integer quantisation**: This technique will require an unlabelled representative sample and will produce up to 75 per cent size reduction with a small accuracy loss. This technique will work on CPUs and GPUs (Android), EdgeTPU and Hexagon DSPs
- **Quantisation-aware training**: This technique requires labelled training data, producing a size reduction of up to 75 per cent. It offers the smallest accuract loss and can work on CPUs, GPUs (Android), EdgeTPU, Hexagon DSP

Next is pruning. Pruning is an approach which removes parameters within the model which will have only a minor impact on the prediction. A pruned model will take up the same size of memory and will have the same latency, but it can be compressed more efficiently.

Then finally there is clustering. Clustering works by grouping the weights of each layer into a predefined number of clusters, which allows for the centroid

values for the weights belonging to each cluster. This means the number of unique weights is reduced, which also reduces the complexity.

3 Literature Review

Many articles have been written on the topic of obstacle detection, some of these, such as [11], use mobile devices such as smartphones and tablets with in-built cameras, which run algorithms to detect the presence of obstacles. While smartphones and tablets are not the proposed constrained device used in this research, the paper offers interesting insights into the use of a small portable device which is not intrusive. The model presented in this work uses a number of sensors for detecting obstacles. The sensors listed include the onboard camera, proximity sensor, accelerometer, and gyroscope. The researchers in this work use the Canny Edge Detection algorithm [12]. This algorithm extracts useful information from vision objects and reduces the amount of data this is processed. As such, it is a common algorithm found in the area of computer vision as well as Edge detection.

Other research, such as [13] offers insights into how machine learning can be used to carry out Real-Time Ranging and Localisation (RTRL) and discusses a few different approaches to labelling data which are interesting and offer a useful background to the proposed work. They use two different labelling methods – one is a simple multi-class labelling system, and the other is a grid labelling system. This work also discussed the performance of several classification models which was very informative – the authors showed that tree-based models along with Stochastic Gradient Boost performed better than linear models. The paper also discussed which of the labelling methods they investigated showed the best performance. As such, this paper contains a lot of useful information related to the proposed work. The proposed work's goal is to access the feasibility of a deep learning/machine learning model running on what is a constrained device.

The work presented in [14] offers a discussion on topics like the future of Machine Learning at the Edge, as well as a discussion on Machine Learning/Deep Learning algorithms, it also presents a discussion on how Machine Learning can be brought to the Edge, discussing architectures and hardware, and wireless standards for Artificial Intelligence-enabled devices. As such, the work in [14] provides a wide-ranging and extremely useful background to the topic of Machine Learning and Deep Learning on constrained devices.

The work presented in [15] used a thermal imaging camera and provides some useful insights into data collection as well as a discussion on the system architecture. The work then goes on to discuss the use of CNNs and highlighted some of the problems encountered when training on the AlexNet network – specifically lighting, which may be an issue for the work proposed here, depending on sensor type, and as such, is something to keep in mind.

Finally, a slightly older paper is presented in [16]. While this paper is older, it provides a very useful entry point into machine learning at the Edge. The work discusses some of the early stages of embedded machine learning development,

looking at work done using smartphones and how non-CPU processors, such as DSPs can play an important role in reducing deep networks to allow them to be used on a constrained device. The paper also discusses methods for overcoming the constraints of an embedded system, which includes a compression model which allows deep neural networks to fit and operate on embedded systems. Some work has been done to try and develop systems to assist people with visual impairments, however, these systems do not all use machine learning or deep learning techniques, the work is interesting as it gives some insight into the type and range of sensors used to assist visually impaired people.

In [17] the author discusses using a traditional cane which some visually impaired people use and upgrading this stick by connecting it with an application which runs on a mobile phone. The authors proposed system includes the use of an ultrasonic sensor for distance measurement, a water sensor to detect rainfall or water level and a NodeMCU microcontroller. When near an obstacle or some water, the proposed system will send an alert to the user.

The authors of [18] propose a system called LineChaser, which guides the user to the end of a line while continuously reporting the distance and direction of the last person in the line, so they can be followed by the LineChaser user. The equipment used in this system is a smartphone camera, which is used to detect people nearby with the built-in infrared sensor which is used to estimate distance. As such, this work requires the user to operate and hold a smartphone while using the LineChaser system.

Other work has proposed the use of drones, such as [19] which proposes a drone-assisted navigation system to support people with visual impairments to carry out "hand-object" tasks through the use of fine-grain haptic feedback. The study shows that the drone-based approach produced greater accuracy for the user when compared to audio-based hand guidance system.

The authors of [20] propose an object detection system for visually impaired people using machine learning. The proposed system uses computer vision and a machine learning multi-label classification approaches to recognise objects which are close to the user of the system. The system architecture uses TensorFlow and Open CV, the YOLO (Tiny and V3) algorithm and RetinaNet to detect objects in indoor and outdoor settings. Using various techniques, the authors were able to show good results in object detection.

The authors of [21] discuss a portable camera-based method to assist people with visual impairments to understand and recognise indoor objects. The authors proposed using a coarse description technique which allows the classification of multiple objects. This coarse approach has the benefit of keeping processing time low, which it does by sacrificing some information detail. The proposed system uses two techniques: Euclidean distance measurements, and semantic similarity measures which use Gaussian process estimation.

4 System Development

4.1 Data Collection

A list of potential sensors was examined against a list of criteria, such as cost, accuracy, size, power consumption and so on. Each potential sensor (from LiDAR to ultrasonic sensors to cameras) was scored. This scoring showed that the best sensor, given the constraints of the research, was the ultrasonic sensor. This was teamed with the Arduino Nano 33 BLE sense development kit which connects to a PC. The Arduino development kit offers a small, affordable system with its own IDE. The ultrasonic sensor's output can be read using the Arduino's general-purpose input/output (GPIO) pins. It was decided to use two ultrasonic sensors, to allow data collection on two channels. The image in Fig. 1 shows the hallway where the data was collected, with the large bin obstacle in place and Fig. 2 show the Arduino based data collection system:

Fig. 1. Data Collection Location with Large Bin

Data were collected for a number of obstacles, namely a closed door, a display stand, a storage box, and a large bin, with each obstacle placed in the centre of a hallway. These obstacles were chosen because they are reasonably large, and as such, should return strong distance measurements.

In front of the obstacle was an imagined 3-foot by 3-foot grid. This grid was broken into 1-foot by 1-foot sub-grids. This gives 9 grids in total, labelled 1 to 9. The data collection system was placed in each of those 9 grids, pointing directly forward. An example of this grid is shown Fig. 3.

When the measurements start, the data will be sent to a serial port monitor, specifically Tera Term. Tera Term allows the user to set up a file in which the transmitted data will be saved in comma-separate values (CSV) files. As well as

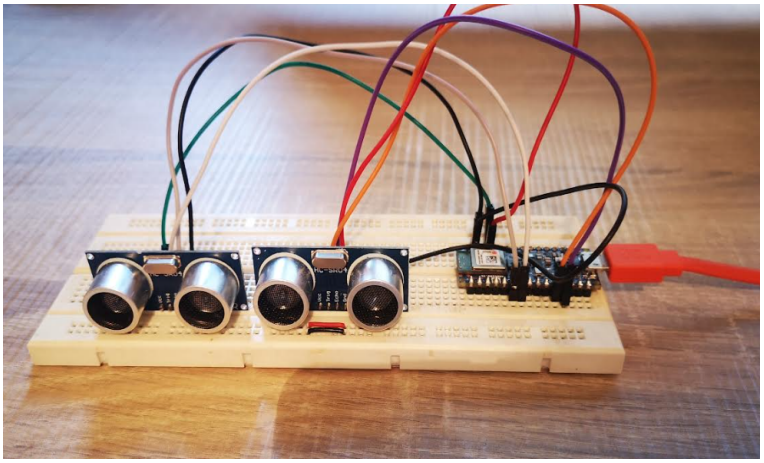

Fig. 2. Arduino Nano 33 BLE Sensor Development Kit

the obstacles, data will be collected from the hallway when there is no object in place. As well as the gathered distance measurements information was gathered on the grid location and if an object was present or not. This information was output via a program written for the Arduino. An example of the dataset headers can be seen in Table 1:

Channel1	Channel2	Object	Grid
2243	1392	Yes	1
2399	1476	Yes	1
2313	1245	Yes	1
2257	1388	Yes	1

Table 1. Example of Ultrasonic Dataset

4.2 Initial Analysis

Introduction. An initial analysis was carried out with a focus on what is referred to as "binary analysis" and "grid analysis". The purpose of the binary analysis is to understand how well the machine learning techniques are at determining if an object is present or not in the data. Grid analysis is trying to determine how well machine learning or deep learning techniques can be used to determine how accurate the model is at determining which grid the data was gathered from. This information would be useful for navigation (i.e. should the person move to the left or right to avoid an object).

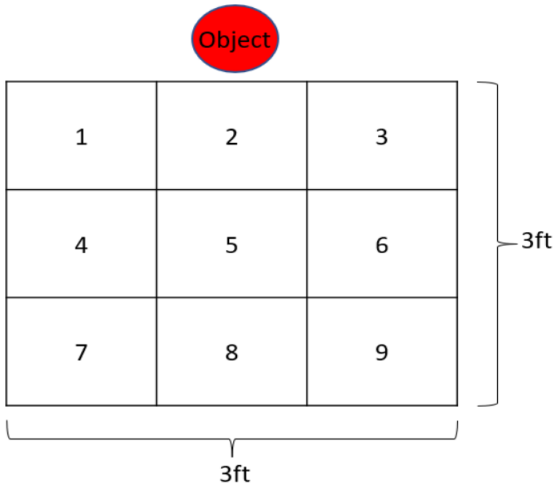

Fig. 3. Image Depicting the Data Collection 3x3 Grid

Binary Analysis. To understand how well a system may be able to detect an object or not, an initial test was carried out on the datasets. This initial examination would be carried out by using two datasets at a time: one dataset would be a dataset for an object, and the other dataset would be data gathered for no object.

The data was processed through a number of classification models, specifically: Logistic Regression, Decision Tree, K-nearest neighbour, Linear discriminant analysis, Gaussian Naive Bayes.

The order of the datasets tested is shown in Table 2:

Dataset 1	Dataset 2
Display Stand	Clear Hallway
Closed Door	Clear Hallway
Storage Box	Clear Hallway
Large Bin	Clear Hallway

Table 2. Datasets test for initial analysis

Using Python the two datasets were brought together into a single `DataFrame`. The data was then into training and testing datasets using `train_test_split` with the default settings. The results of this analysis can be seen in Fig. 4

It can be seen from the results in Fig 4 that the `Decision Tree Classifier` and `K-NN Classifier` are producing the best accuracy scores, with 100% accuracy, for all the test data. The lowest accuracy score was generated by the `LDA`

Model Name	Display Stand		Closed Door		Storage Box		Large Bin	
	Train	Test	Train	Test	Train	Test	Train	Test
Logistic Regression Classifier	73	73	100	100	81	81	80	80
Decision Tree Classifier	100	100	100	100	100	100	100	100
K-NN Classifier	100	100	100	100	100	100	100	100
Linear Discriminant Analysis	72	72	94	94	83	89	82	82
Gaussian Naïve Bayes	69	69	93	93	80	79	81	81

Fig. 4. Results, Shown as a Percentage, for the Initial Binary Search

algorithm, producing an accuracy score of 72%. It can be seen that typically the accuracy generated for the "train" data and that generated for the "test" data are the same. One reason for this could be that the data does not have enough variation to generate a test dataset that is sufficiently different. However, these results did provide confidence that a system could be designed to detect objects.

Grid Analysis. The next step in the analysis process was to understand how well we can determine an object from a grid perspective. This is important as having the ability to understand where the object is relative to the person is key for a navigation system. To complete this analysis, the datasets had to be modified. As has been discussed, there are datasets for four objects and a dataset for no object. Each dataset contained information for nine grids. The data for no object needs to be merged with the datasets for objects. This required the addition of an extra grip position, specifically grid 0. This grid 0 would represent no object. The important thing to get right here is that the datasets are balanced (i.e. the same number of samples for each grid).

With the balanced dataset in place, the grid analysis could be carried out. This time the response was the grid number, unlike the binary search analysis, where the presence of an object or not was the response. The data was processed through the same models listed above and the results of this analysis can be seen in Fig. 5.

Model Name	Display Stand		Closed Door		Storage Box		Large Bin	
	Train	Test	Train	Test	Train	Test	Train	Test
Logistic Regression Classifier	80	80	99	99	97	97	96	96
Decision Tree Classifier	100	100	100	100	100	100	100	100
K-NN Classifier	100	100	100	100	100	100	100	100
Linear Discriminant Analysis	66	65	97	97	96	96	95	95
Gaussian Naïve Bayes	91	91	100	100	99	99	99	100

Fig. 5. Results, shown as a Percentage Value for the Grid Search

We can see from this analysis that once again, the Decision Tree Classifier and the K-Nearest Neighbour algorithms produce 100% accuracy. The LDA algorithm produces some of the lowest accuracy figures. However, the majority of the results are over 90%. As with the binary analysis, it can be seen that the results for the train and test datasets are the same for each object. This will require further investigation.

4.3 Using Unseen Data

One possible explanation for the identical accuracy of the training and testing data could be that the data used for both sets are very similar. This is because the measurement system and the object being detected are static throughout the data collection process.

To test this idea, and to look for more representative results, the algorithms will be trained on datasets from three objects, and tested on data from one object, which it has not "seen". This analysis was done for both the binary analysis and the grid analysis. The results can be seen in Fig 6.

Grid Analysis
Trained: Closed Door, Display Stand, Large Bin
Tested: Storage Box

Model Name	Train	Test
Logistic Regression Classifier	36	10
Decision Tree Classifier	100	25
K-NN Classifier	100	48
Linear Discriminant Analysis	33	20
Gaussian Naïve Bayes	48	40

Grid Analysis
Trained: Closed Door, Storage Box, Large Bin
Tested: Display Stand

Model Name	Train	Test
Logistic Regression Classifier	67	20
Decision Tree Classifier	100	29
K-NN Classifier	100	29
Linear Discriminant Analysis	61	20
Gaussian Naïve Bayes	73	20

Grid Analysis
Trained: Closed Door, Display Stand, Storage Box
Tested: Large Bin

Model Name	Train	Test
Logistic Regression Classifier	36	28
Decision Tree Classifier	100	32
K-NN Classifier	100	30
Linear Discriminant Analysis	30	30
Gaussian Naïve Bayes	50	29

Grid Analysis
Trained: Storage Box, Display Stand, Large Bin
Tested: Closed Door

Model Name	Train	Test
Logistic Regression Classifier	39	30
Decision Tree Classifier	100	27
K-NN Classifier	100	28
Linear Discriminant Analysis	35	19
Gaussian Naïve Bayes	57	28

Fig. 6. Results for the Initial Binary Search

Fig 6 shows that the training accuracy results for the decision tree classifier and the K-nearest neighbours algorithms are still at 100 per cent, however, there is a drop in the accuracy figures when the algorithms are presented with "new" data (i.e. data it has not been trained on).

This analysis indicates that the standard classification models, using default settings, will likely struggle to make accurate classification predictions when presented with data gathered from unseen objects. This indicates two possible paths forward. One would be to gathered data for more objects, which can be used to train the models with. This would be quite a labour-intensive task, and one can never cover all objects. Another path is to investigate the ability of neural networks to train a model.

4.4 Neural Networks

Using neural networks offers a number of advantages. Neural networks can be used to learn more complex, they are very well suited to classification tasks, and since they are good at learning the underlying relationships in the data, they are good with data that is not cleaned and prepared. A number of frameworks are available when working with neural networks. This work will use the Keras framework.

Working with neural networks requires more work due to the need to decide on the number of layers, the number of neurons, the activation function and so on. During the neural network investigation, a number of parameters were investigated, such as using the Model and Sequential class as well as the rmsprop, Adam and SGD optimisers. The loss parameter investigated was the Sparse Categorical Cross-entropy, as well as the mean squared error loss. The number of epochs was varied as well at the batch size. Some investigation of the learning rate was carried out to understand the impact that can have on the accuracy.

This investigation found that, for the grid analysis, the following parameters shown in Fig 7, produced the best results

Layer			Optmiser	Compiler			Fit			
Dense Layers	Unit	Activation	Learning Rate	Optimiser	Loss	metrics	Epochs	batch_size	Acc	Loss
4	2,20,20,10	R-R-R-S	0.01	SGD	SCC	Accuracy	200	128	0.95	0.162

Fig. 7. Optimal Setting for Neural Network

Now that the network parameters have been set, the model can be converted to a header file that can be deployed on the constrained device. The model can be converted within the TensorFlow system.

4.5 System Algorithm

One of the constraints of an embedded device, as previously mentioned, is power, another is latency. When designing a potential navigation system for people with sight difficulties, these constraints need to be kept in mind. Continually processing data at the Edge will be power intensive. For this reason, it is proposed that

the navigation system is split into two sections. The first would be continually processing data to determine *if* an object is present. this processing would continue as long as the answer is no. When an object is detected, the system would switch to the second stage of processing, where it will determine *where* the object is, thus allowing the user to navigate around it. This very simple system is shown in the flowchart in Fig. 8:

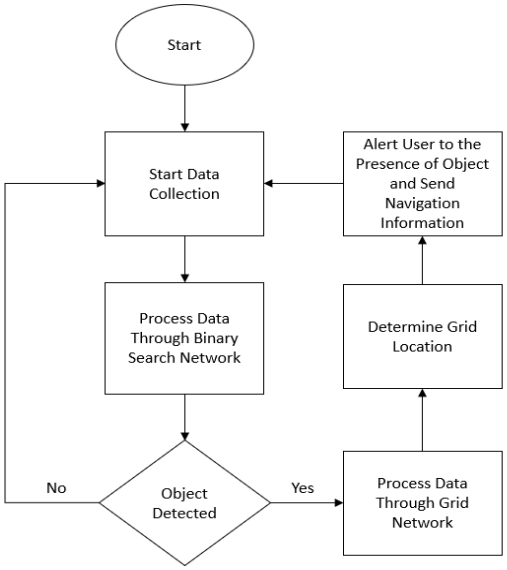

Fig. 8. Embedded Algorithm Flowchart

The part of the flowchart which asks if an object is detected or not could be carried out by a threshold value, or potentially a series of thresholds. For example, if x and y are a series of decreasing values (say, 5000 and 3000), a system could be designed to ask has the first threshold has been met, if the answer is yes, it then asks has the second threshold has been met. If the answer is yes again, the system processes the last gathered data through the grid analysis model to detect *where* the object is.

This process could save on power as the system would not always be processing data through the grid model.

5 Evaluating the Deployed System

When the model was initially deployed onto the Arduino, the system would not carry out any inference. Work was done to try and understand the issues, and this investigation lead to the potential problem being the shape of the data. To

try and make the system as simple as possible, a stripped-back version of the code was developed which read provided training data from header files. With this in place, and the data reshaped for the input and outputs, the system was tested again, and it started to do inferences. The result from this inference was always Grid 6, even when the input data was varied to give data from grids other than Grid 6.

6 Discussion

The research set out to understand if a system could be developed which could detect obstacles in the path of a visually impaired person, in an indoor environment, allowing the user to navigate that setting. To work towards this goal, a number of tasks needed to be completed, such as the development of a data collection system and the subsequent collection of usable data from a number of obstacles. This data would then be used to train and test a number of machine learning and deep learning models to understand which is best for detecting and localising obstacles. After that, the chosen model would be implemented on a constrained device and the performance evaluated.

Appropriate data was collected from a number of large obstacles using an Arduino development kit and two ultrasonic sensors. With the datasets in place, an initial analysis was carried out using machine learning algorithms to understand how well the data could be classified. This initial analysis indicated that these algorithms could detect both the presence of an object with a high degree of accuracy and could localise the grid location for that data. The accuracy results went down when unseen data was presented to the algorithms during the testing phase.

This reduction in model accuracy promoted an evaluation using deep learning techniques. Various hyperparameters were evaluated during this research to understand which combination would produce the best accuracy and loss scores.

When the neural network investigation was completed, the model was converted to a header file which was deployed on a constrained device. When tested, the model did not perform as expected. The system would carry out an inference, however, the result this inference would return did not change.

This work shows that it should be possible to implement an obstacle detection and navigation system on a constrained device due to the fact that a model was developed, which produced high accuracy scores, and deployed onto a constrained device. The issues with the incorrect inference were found in the later stages of the research and had more time been available, to allow an investigation of the issue, it is felt a solution could have been found.

7 Future Work

The incorrect inference from the constrained device would be the initial focus of any future research on this project, with an aim to root causing the issue and put a fix in place. As well as this, the system algorithm which would run

on the constrained device could be developed further to ensure it would run in as efficient a manner as possible, to allow the best use of the limited resources. Investigation of other constrained devices could produce promising results too.

While some work was done on quantisation, showing a reduction in size of the model from 5280 bytes to 5044 bytes using Float16 quantisation, more work in this area should yield better size reduction.

References

1. RNIB, "Key stats about sight loss 2021," 2021. [Online]. Available: https://media.rnib.org.uk/documents/Key_stats_about_sight_loss_2021.pdf. [Accessed 3 2 2023].
2. L. Pezzullo, J. Streatfeild, P. Simkiss and D. Shickle, "The economic impact of sight loss and blindness in the UK adult population," BMC Health Services Research, vol. 18, 2018.
3. Business Wire, "Global Microcontroller Market Size, Share & Trends Analysis Report 2021-2028 - ResearchAndMarkets.com," 2021. [Online]. Available: https://tinyurl.com/bdcpkuybm. [Accessed 2 3 2023].
4. NVIDIA, "Jetson nano Development Kit User Guide," [Online]. Available: https://developer.nvidia.com/embedded/dlc/Jetson_Nano_Developer_Kit_User_Guide. [Accessed 12 4 2023].
5. beaglebone.org, "Beaglebone AI 64," [Online]. Available: https://beagleboard.org/AI. [Accessed 12 4 23].
6. T. Sipola, J. Alatalo, T. Kokkonen and M. Rantonen, "Artificial Intelligence in the IoT Era: A Review of Edge AI Hardware and Software," 2022 31st Conference of Open Innovations Association (FRUCT), Helsinki, Finland, 2022, pp. 320-331, doi: 10.23919/FRUCT54823.2022.9770931.
7. Arduino, "Arduino Nano BLE 33 Sense" [Online]. Available: https://store-usa.arduino.cc/products/arduino-nano-33-ble-sense. [Accessed 23 4 23].
8. TensorFlow Lite Org., "TensorFlow Lite" [Online]. Available: https://www.tensorflow.org/lite/guide. [Accessed 14 4 2023].
9. TensorFlow Lite Org., "TensorFlow Lite" [Online]. Available: https://www.tensorflow.org/model_optimization. [Accessed 14 4 2023].
10. TensorFlow Lite Org., "Internet of Things - Edge TPU" [Online]. Available: https://cloud.google.com/edge-tpu/. [Accessed 16 4 2023].
11. K. A. B. Za'aba and L. B. Theng, "Edge Based Obstacle Detection Model Focused on Indoor Floor-Based Obstacles," 2019 IEEE 9th Symposium on Computer Applications & Industrial Electronics (ISCAIE), Malaysia, 2019, pp. 202-207, doi: 10.1109/ISCAIE.2019.8743866.
12. J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.
13. R. Sattiraju, J. Kochems and H. D. Schotten, "Machine learning based obstacle detection for Automatic Train Pairing," 2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS), Trondheim, Norway, 2017, pp. 1-4, doi: 10.1109/WFCS.2017.7991962.
14. M. Merenda, C. Porcaro, and D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," Sensors, vol. 20, no. 9, p. 2533, Apr. 2020, doi: 10.3390/s20092533

15. S. Quinn et al., "A Thermal Imaging Solution for Early Detection of Pre-ulcerative Diabetic Hotspots," 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Berlin, Germany, 2019, pp. 1737-1740, doi: 10.1109/EMBC.2019.8856900.
16. N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi and F. Kawsar, "Squeezing Deep Learning into Mobile and Embedded Devices," in IEEE Pervasive Computing, vol. 16, no. 3, pp. 82-88, 2017, doi: 10.1109/MPRV.2017.2940968.
17. Amira. A. Elsonbat. (2021). Smart Blind Stick Design and Implementation. International Journal of Engineering and Advanced Technology (IJEAT), 10(5), 17–20. https://doi.org/10.35940/ijeat.D2535.0610521
18. Masaki Kuribayashi, Seita Kayukawa, Hironobu Takagi, Chieko Asakawa, and Shigeo Morishima. 2021. LineChaser: M. Kuribayashi, K. Seita, T. Hironobu, A. Chieko and M. Shigeo, "LineChaser: A Smartphone-Based Navigation System for Blind People to Stand in Lines," Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, p. 13, 2021.
19. F. Huppert, G. Hoelzl and M. Kranz, "GuideCopter - A Precise Drone-Based Haptic Guidance Interface for Blind or Visually Impaired People," Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, p. 14, 2021.
20. V. N. Mandhala, D. Bhattacharyya, B. Vamsi and N. Thirupathi Rao, "Object Detection Using Machine Learning for Visually Impaired People," International Journal of Current Research and Review, vol. 12, no. 20, pp. 157-167, 2020.
21. M. L. Mekhalfi, F. Melgani, Y. Bazi and N. Alajlan, "A Compressive Sensing Approach to Describe Indoor Scenes for Blind People," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 7, pp. 1246-1257, July 2015, doi: 10.1109/TCSVT.2014.2372371.