NavSense: A Navigation Tool for Visually Impaired

John Ryan*, Daniel Okazaki†, Michael Dallow‡, and Behnam Dezfouli§
Department of Computer Science and Engineering, Santa Clara University, Santa Clara, CA, USA
*jcryan@scu.edu, †dtokazaki@scu.edu, ‡mdallow@scu.edu, §bdezfouli@scu.edu

***Abstract*—The visually impaired rely heavily on hearing and touching (with their cane) to navigate through life. These senses cannot make up for the loss of vision when identifying objects in the user's path. In this paper, we propose NavSense, an assistive device that supplements existing technology to improve navigation and peace of mind in day to day life. NavSense relies on range detection, computer vision, and hardware acceleration mechanisms to provide real-time object identification and context to the user through auditory feedback. In particular, we use four hardware platforms – Raspberry Pi 3 B+, Coral Accelerator, Coral Development Board, and Intel Neural Computer Stick – to compare the efficiency of object detection in terms of time and energy during setup and inference phases. Based on these results, it is possible to tailor the design for specific energy-accuracy requirements. Also, we have implemented and used NavSense in real-world scenarios to show its effectiveness.**

***Index Terms*—Blindness, Accessibility, Computer Vision, Machine Learning, Object Recognition, Image Classification**

I. Introduction

Visual impairment is a disability that is estimated to affect 1.3 billion people worldwide, the majority of which are over the age of 50 [1]. According to a 2011 study on 300 legally blind or functionally blind individuals, over 50% of the participants said that they had had at least one head-level accident in the last year, while 12% said that they experience mishaps more than once a month. The same study found that over 50% of participants have tripped resulting in a fall, once a year or less with just over 30% occurring once a month or less [2]. In the United States, it is estimated that costs associated with visual impairments total upwards of $40 billion a year with medical costs accounting for $22 billion of that amount [3].

Navigating while blind or visually impaired can be extremely dangerous, with each step forward posing a significant risk for tripping and serious injury. In order to safely navigate, those affected by visual impairments are typically accompanied by another person or service animal [4]. However, these methods can limit independence and are often inconvenient and cumbersome. Current solutions for Electronic Traveling Aids (ETAs) come with a variety of problems ranging from the expensive cost of proprietary technology to their large overbearing sizes. Portable devices for the visually impaired are often ineffective for everyday use, sometimes even harming the user's experience by providing inaccurate or vague information [5].

While there are many existing ETA products, they present an abundance of issues. One of the more popular ETA products, the Sunu Band [6], is a wristband that uses sonar and haptic feedback (vibrations) to tell the user how far they are from an object. A big shortcoming of this device is that it must be pointed in front of the user, meaning that their arm must be pointed down and their elbow cannot be bent. Additionally, the user has no way of differentiating the kind of object that they're approaching. Another common shortcoming seen in ETA product design is the interference with a visually impaired person's ability to navigate by listening for audial cues [4]. They use sounds to navigate their surroundings, and any sounds produced by a device may interfere with their capability to navigate. This shortcoming has been observed in products such as the UltraCane, Smart Cane, Silicon Eyes, and more [5]. Furthermore, they have problems with usability and high cost.

The most expensive of the four products mentioned previously is the UltraCane, starting at $720 [7]. The price of this product is extremely high for consisting mainly of two ultrasonic sensor and vibration motors. The Sunu Band is cheaper, starting at $300, but is still expensive compared to the cost of its components. As mentioned above, the biggest problem with the Sunu Band is usability. Some products, such as the UltraCane and Sunu Band, use vibrations rather than sounds, and visually impaired people often find it difficult to learn the meanings of different vibrations. Our goal, therefore, is to improve usability by providing intuitive signals and warnings, while keeping the cost low.

In this paper, we propose NavSense, a device that increases independence and mobility for those who are affected by visual impairments. This solution uses a camera, computer vision, and range finding sensors to notify a user about obstacles in the environment around them. The technology offers context – such as object identification – to the user to assist in navigation. By identifying and classifying objects such as stairs, people, vehicles, and stationary obstacles, NavSense enables the user to safely progress along their intended path and lower the risk of injury. We rely on the user's side to side motion when moving the cane or the motion of a guide dog to frame the camera and get an accurate depiction of the current environment in front of the user. The small footprint of this device simplifies integration with white canes without burdening the user with heavy and cumbersome devices.

NavSense attaches to a user's preexisting guide dog or white cane, integrating with the user's existing routines to minimize the learning curve associated with obtaining a new assistive device. The proposed solution improves upon existing systems and offers a simple, yet informative user experience to improve the quality of life of the visually impaired. By adding on top of preexisting technology, users who do not fully trust the technology can still have peace of mind that the assistive devices they have become accustomed to are still available as

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.

a backup.

In conjunction with the Raspberry Pi 3B+ [8], Raspberry Pi Cam [9], an AI hardware accelerator, and a distance sensor, we developed a cost effective, low power system capable of offering real-time context and navigation assistance. In particular, we compare the efficiency of various hardware platforms in terms of accuracy, energy, and duration. The main software libraries we use are: the Neural Compute Stick API, the Google Edge TPU API, and the Python Imaging Library (PIL) to gather and process images, perform inference, and list the identified objects and their classes. It is worth mentioning that our results are complementary to those of [10], which does not study hardware-accelerated solutions.

The rest of this paper is organized as follows. Section II presents the design and implementation of NavSense. We evaluate the performance of NavSense in Section III. Section IV concludes the paper.

II. Design and Implementation

The development of NavSense required a large focus on user needs. Through surveys, we identified the issues that visually impaired users found with existing ETAs and were able to make adjustments to create a more accessible system. By creating a system that is used with existing technologies such as white canes and guide dogs, we do not force the user to abandon technologies that they are comfortable with already. Figure 1 shows the components of NavSense.

Our largest challenge was creating a user interface with no visual components. Our interface uses audio to communicate information and allows the user to physically feel any buttons they may need to press for operation. Functionally, NavSense, as shown in Figure 2 goes through several steps in the identification process to output information to the user. These steps are: image gathering, processing, feature collection, point of interest detection, point of interest processing, decision making, and user notification.

For hardware, we test and compare three different boards: the Raspberry Pi 3B+ [8], the Xilinx Ultra96 [11], and the Google Coral Development Board [12]. We also compare three USB hardware accelerators: the Intel Neural Compute Stick (NCS) [13], the Intel Neural Compute Stick 2, and the Google Coral Accelerator [14] for use in conjunction with the Raspberry Pi 3B+. There were two cameras that we tested for image capture: the Raspberry PiCam and the Logitech C920 Webcam [15]. In addition, we compare two distance sensors: the MB1040 LV-MaxSonar-EZ4 ultrasonic sensor [16] and the TFmini Infrared Time of Flight sensor [17].

For the main programming language, we use Python for its ease of use with machine learning, and the built in APIs included with both the Intel Neural Compute Stick and Google Coral Accelerator which are both written in Python.

A. Image Gathering and Processing

The image gathering process within Navsense uses the Python Imaging Library (PIL) to gather images from the PiCam. PIL not only manages when we gather our frames, but it also keeps track of how many frames per second our

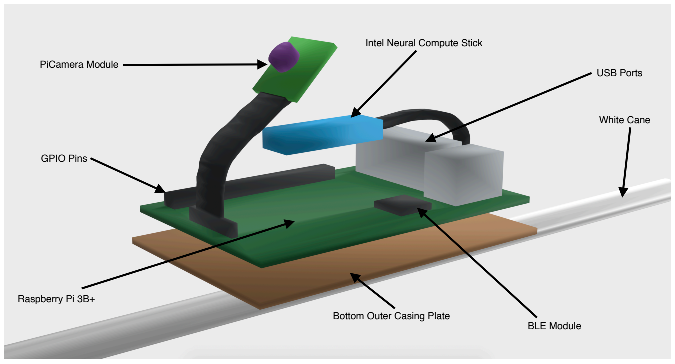

Fig. 1. The physical component layout of the NavSense system. The USB AI hardware accelerator, Raspberry Pi 3B+, and PiCam module form the main components of the system.

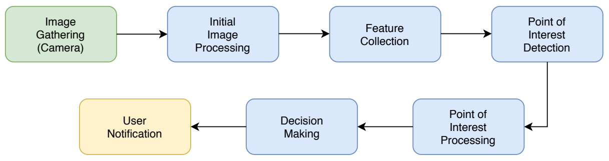

Fig. 2. The software system architecture of the NavSense system detailing the seven main sections of processing the device completes.

system is able to compute for the benchmarking tests. The images are taken from the camera and then sent on to Image Processing.

For the Intel Neural Compute Stick, image processing occurs on the Raspberry Pi using the OpenCV (CV2) library [18]. Simple processes such as image resizing and reformatting for float16 calculations are handled by this library in order to prepare the image to be sent to the Neural Compute Stick. We use the CV2 library due to its simplicity, widely available documentation, community support, and the fact that it is open source. CV2 is written in C and is compiled in multiple embedded platforms allowing for easy porting between devices and development [19]. The features found in the library offer precisely what we need to deliver an image in the proper format. For the Coral Accelerator and Coral Development Board, we use the built in machine learning engine included with installation of either device to run object detection.

The choice to use the Raspberry PiCam was made due to its ability to easily interface with the Raspberry Pi and the pre-processing features it offers on the board, which helps offloading processing away from the CPU [20]. The simple calling function for the capture of an image helps speed up the image gathering process. The customizability of the image capture process, with variables that are adjustable by the programmer such as the preview time and the format of the image speeds up image processing. The small form factor of the PiCam allows us to place the camera with the Raspberry Pi board together in a small package. With four mounting holes on the PCB, we are also able to create custom mounts and angle the camera in a way we see fit.

The other option for camera that we experimented with was

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.

the Logitech C920 Webcam. While this device offers higher resolution pictures, its large form factor, bulky USB connector, and incompatible mount led us to believe that it would not be a good fit for NavSense. Since the device must be mounted on a white cane or guide dog, any increase in size or weight decreases the comfort for the user.

B. Point of Interest Detection and Decision Making

For the Intel Neural Compute Stick (NCS), the work of image analysis and object recognition using machine learning models is handled by the Neural Compute Stick API. The chosen recognition model is processed using the Neural Compute Stick SDK to produce a graph file compatible with the device. This graph file is loaded into the NCS's memory and used to complete a forward pass over the image using its 12 SHAVE cores to identify points of interest and identify the objects associated with them using a MobileNetSSD graph file [21] and relay the locations and names of those objects back to the Raspberry Pi 3B+. The Coral Accelerator uses its own engine to process object detection and comes with its own EdgeTPU ASIC machine learning accelerator to process TensorFlow Lite models, and can be used through API calls similarly to the NCS.

We use the Single Shot Detection (SSD) models due to their general accuracy improvement over YOLO networks [22]. While speed is certainly important to our system, we believe that the most important aspect to the safety of the user is that the predictions made by the system and the information delivered to the user are as accurate as possible [4].

We use the MobileNet v2 Convolutional Neural Network due to the speed advantages offered over its previous generation. By using depthwise separations, Mobilenet is able to preform two layers of convolution – depthwise and pointwise – to gather and filter information from the image at a faster speed. Furthermore, Version 2 projects the layers further apart so as to reduce bottlenecking in the analysis of the features. By directly decompressing, filtering, analyzing, and then recompressing the data, Mobilenet V2 [23] is able to achieve the performance of larger models using fewer resources and at faster speeds.

C. Distance Measuring

In order to measure distance, we compare two different sensors: the MB1040 LV-MaxSonar-EZ4 (Ultrasonic) produced by MaxBotix, and the TFmini Infrared Time of Flight Sensor. Both of the sensors we evaluate can measure distances at least 20 feet away. NavSense needs to be able to accurately measure the distance of objects in front of the user. This allows the user to better visualize their surroundings. Since NavSense is attached to a white cane, which continuously moves side to side in a horizontal motion, we will have to ensure that the distances provided to the user are fast and accurate, relating to the same objects detected by the camera.

To set up either sensor, the VCC pin on the sensor is connected to the 5V output on the Raspberry Pi, ground on the sensor is connected to ground, and the TX pin on the sensor is connected to the RX pin on the Raspberry Pi. The Raspberry Pi provides the Python library *RPi.GPIO* [24], that allows the user to access and control the GPIO pins on the board. The value read from the RX pin on the Raspberry Pi is mapped to a certain distance. The reading of the RX pin is synchronized with the camera taking an image so that we know the distance read is corresponds to the object at the center of the image.

D. User Notification

As mentioned in the introduction, the goal of this project is to improve usability by making NavSense's responses intuitive. For this reason, we have decided that audial cues will be the clearest way to communicate what the system is observing. We determined that the simplest and clearest way to do so is using text-to-speech. For this project we are using *pyttsx* [25], a Python text-to-speech library that utilizes *espeak*. This functionality can be accessed through both Bluetooth and the built-in headphone port on the Raspberry Pi 3B+. Our research into current ETA products shows that audial cues can interfere with navigation when used incorrectly, so we are extra careful with how we present information to the user [5]. We allow the user to adjust both the volume and speaking speed of the device, in order to cater to their needs.

When the user has the device in audio mode, NavSense scans the environment about every 5 seconds, an image is taken, the distance sensor value is read, and inference is run on that picture. Once the object at the center of the image is found, the distance to the object is put into a string, and concatenated to the object label string. This string is then passed to the text-to-speech library and read to the user. The user will also be provided with a button, so that they can interrupt the device at any time and choose when the inference and distance measuring is performed in real-time. The reason why the object detection is done every 5 seconds is to ensure that the user is not bombarded with too much information, most of which would have already been presented to the user the last time an inference was taken. Continuous running of the device would also increase the power usage, decreasing the battery lifespan of NavSense.

E. Challenges Faced

Over the course of development we researched and developed on different possible platforms and technologies for use in NavSense. The development boards used were the: Xilinx Ultra96, Google Coral Development Board, and Raspberry Pi 3B+. We had initially planned development with the Intel Neural Compute Stick 2 in conjunction with the Raspberry Pi 3B+, however we soon learned that this device was not initially compatible with the Raspberry Pi. The OpenVino [26] toolkit that is used to interface with the Intel Neural Compute Stick 2 was not compatible with ARM processors such as the Raspberry Pi until very recently and the version that was compatible was a stripped down subset of the toolkit which did not contain the features that we need, such as object recognition support or a working installer. This led us to look into development with the original Intel Neural Compute Stick that was built specifically for the Raspberry Pi. The biggest difficulty with this device was finding object detection

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.

models that are compatible with it. There was a lack of pre-compiled graphs, which are necessary for inference detection with this device, so we had to find pre-trained models and convert them to graphs. It was difficult to find models in the correct format that could be converted to graphs. Most of the TensorFlow and Caffe models that we have found to be compatible were image classification models and are not able to detect multiple objects in a scene. The model we have chosen is a MobileNetSSD v2 model trained on the COCO dataset [27].

Working with the Ultra96 posed several problems in the development process. The first problem encountered was with installing libraries. TensorFlow [28] and OpenCV (CV2) both were difficult to install due to the lack of aarch64 architecture wheels available on their respective websites that would compile for this particular platform. The Xilinx Object Detection GitHub repository [29] will install CV2, however, this version of CV2 does not have the Deep Neural Networks (DNN) module included. We managed to install the most recent version of CV2 through a bash script, whereas TensorFlow was installed after being directed to a user GitHub repository containing the installation files that we needed [30].

The next problem was figuring out which models would run the fastest and most accurate on the Ultra96. Most of the object detection programs that we found ran extremely slow on this board due to the lack of hardware acceleration utilization of those algorithms. In order to fully take advantage of the FPGA, we needed to find a program that worked with hardware acceleration. We found three different programs: QNN [31], BNN [32], and FINN [33] built for the Ultra96. The first program QNN was only built for use with TinyYolo [34]. This model only provides 20 classes, and was insufficient for our purposes. The second program, BNN, was trained on CIFAR 10 [35] which only has 10 classes. We could retrain the overlay, however the maximum number of classes for the overlay is 64, which isn't enough to hold the 90 classes from the COCO [27] data set that we wanted to use. The third program, FINN, does not compile on our machine.

III. Evaluation

In this section, we evaluate the performance, accuracy, and power consumption of the Raspberry Pi 3B+ in combination with the Coral Accelerator and the Intel Neural Compute Stick. We also evaluate the performance and the accuracy of the Coral Development Board.

A. Performance Testing

For performance testing, we modified the NavSense program to receive an image as input instead of capturing an image from the camera. The program calculates the start-up and inference time of each device. We ran both of these programs on the Raspberry Pi 3B+ and the Coral Development Board. We also used the Intel Neural Compute Stick and the Coral Accelerator in conjunction with the Raspberry Pi 3B+. For the Intel Neural Compute Stick, we are using a MobileNetSSD V2 Caffe [36] model trained on the VOC data set that was converted to a graph file to use with the Intel Movidius SDK. The Coral Accelerator is using a MobileNetSSD V2 TensorFlow Lite [37] model trained on the COCO [27] data set and optimized to use with the Coral Accelerator. The Caffe model we used with the Intel Neural Compute stick is the same model as the TensorFlow model, but converted into a Caffe version. The Coral Accelerator is running in normal mode without the high performance setting turned on. The high performance mode causes the Coral Accelerator to become extremely hot to the touch according to the official Coral website. This problem in conjunction with the higher power draw of the device is why we did not pursue enabling this mode for NavSense. The Coral Development board runs the same program as the Coral Accelerator. The control group with the Raspberry Pi 3B+ CPU uses the same program as the Coral Accelerator, but with a regular TensorFlow Lite model instead of the Coral Accelerator optimized model.

For our testing method, we gathered 10 images, 5 images of streets, and 5 images of the interiors of houses, and tested each image 3 times for a total of 30 tests per device. We then averaged the times for the start-up time and inference time and compared the results. We did not perform any performance testing of the Ultra96 because the models that we have on the Ultra96 are too different than the models of both the Coral Accelerator and the Intel Neural Compute Stick. The end result of those tests would not provide us with accurate data to compare between each device.

The start-up time is measured as the time it takes for the device to start-up and is only measured once every initialization of NavSense. As seen in Figure 3, the start-up time of the Coral Accelerator is almost double the start-up time of the Intel Neural Compute Stick with an average set-up time of 3.28 seconds compared to the Intel Neural Compute Stick's 1.26 seconds. The start-up time for the control group and the Coral Accelerator was negligible, as expected, since there is no initialization of hardware required when the program starts. Since the start-up time is only taken into consideration once the device is turned on, it is not as important as the inference time, which will be run every time an image is taken to be processed.

We took the standard deviation of the data to see how consistently the start times were as seen in the error bars of Figure 3. The start-up time of all four devices were low, with all devices having a standard deviation less than 0.125 seconds. The most inconsistent result was using the Raspberry Pi 3B+ with the Coral Accelerator which had a standard deviation of 0.0759 seconds, showing that throughout all of the tests, the start-up time of each device is extremely consistent on all platforms. The standard deviation gives us a glimpse of how reliable NavSense will be, and through our tests, we have demonstrated that no matter the platform, our results will always be extremely consistent and predictable.

In Figure 4 we report that the Intel Neural Compute Stick has an average inference time of 0.196 seconds compared to 0.281 seconds for the Coral Accelerator. We can observe that the Intel Neural Compute Stick is slightly faster than the Coral Accelerator, however for our purposes, the difference

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.

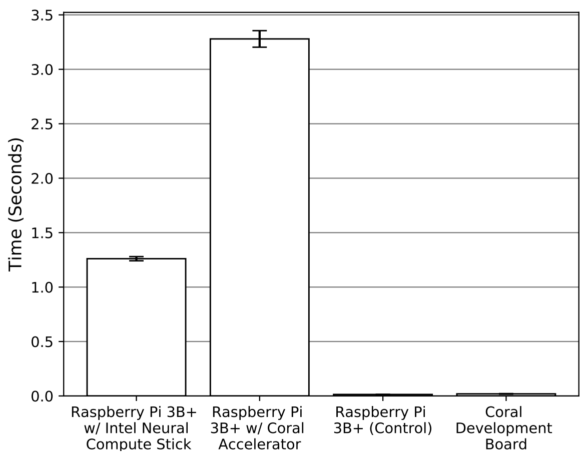

Fig. 3. Setup time: the average amount of time it takes for each device to load and initialize the neural network to accept input for prediction.

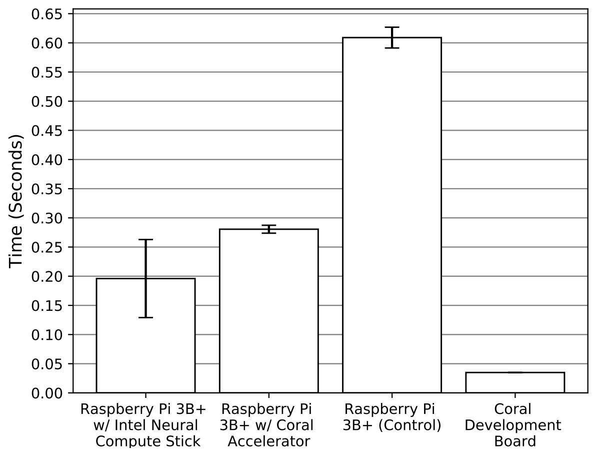

Fig. 4. Inference time: the average amount of time it takes the device to return its predictions once the inference process has been started.

in time for this test is trivial since we will be polling images from the camera about every 5 seconds, and both of these inference times are far below the 5 second mark. The inference time for the control group was twice as slow as the other hardware accelerated options as expected. The Coral Development Board had the fastest inference times with an average of 0.035 seconds per image. However, the greater power consumption, larger form factor, as well as higher heat output prevents us from using this device in our project. Once the program starts to run, inferences will be run every time the camera captures an image, so optimizing the time it takes between each inference is extremely important for both the user experience as well as improving power consumption.

The inference test standard deviation can be also be seen in Figure 4. All of our tests showed extremely consistent performance no matter the image being tested. We report the most inconsistent result being the Raspberry Pi 3B+ with the Intel Neural Compute Stick with a standard deviation

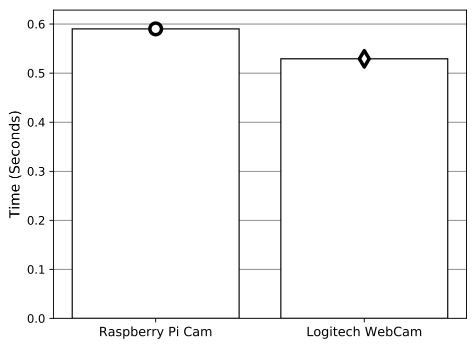

Fig. 5. Average time to take a picture. The Logitech WebCam has a faster capture rate than the Raspberry Pi Cam.

of 0.0669 seconds and the most consistent result being the Coral Development Board with a 0.00019 seconds standard deviation.

Depending on the situation, either the Intel Neural Compute Stick or the Google Coral Accelerator both provide extremely high performance with low power consumption. After the performance tests for object detection, we ran performance tests on both the Raspberry Pi Camera and the Logitech HD Pro Webcam C920 webcam. We measured the time it takes to turn the camera on, capture the image, save it as a file, and finally turn the camera off. We ran this test 150 times on both cameras (150 photos on each camera).

The results can be seen in Figure 5. The speed of both cameras are very similar, with the PiCam being about 100ms faster. Because we want to minimize the physical size of NavSense, the PiCam is the better option.

B. Accuracy Testing

The accuracy of each device was found using the output of the performance test, and then manually finding which objects were correctly identified in each image. The test program outputted the first 10 objects detected with a confidence level of 25% or above if applicable. The Intel Neural Compute Stick identified 45 objects throughout the 10 images and the Coral Accelerator identified 46 objects. However, Figure 6 illustrates how the Coral Accelerator was less accurate than the Intel Neural Compute Stick by about 10%. We hypothesize these results could be influenced by the small sample size of these tests, so we decided not to use this result in our analysis of the devices. The Coral Accelerator had difficulties misidentifying objects as people, while the Intel Neural Compute Stick had difficulties misidentifying bathtubs as boats when detecting at low confidence levels. The control program is the same as the Coral Accelerator program, so the errors are the same. Both programs misidentified objects with a dark background, therefore we cannot recommend the reliance of object detection in the night time without adequate lighting. The Coral Accelerator, the Coral Development Board, and the Raspberry Pi 3B+ CPU all use the program, so accuracy for the Coral Accelerator is the same for all of these devices. The results

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.

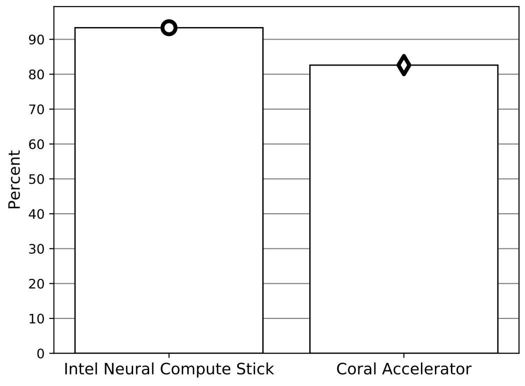

Fig. 6. Prediction accuracy. The Intel Neural Compute Stick predictions are nearly 11% more accurate compared to the predictions of the Coral Accelerator.

of this test reflects the accuracy of the model running on each device rather than the device itself, as different models will result in differing accuracy.

C. Power Testing

In this section, we describe the results gathered through our power testing of the multiple systems we have used in prototyping. To do this power testing we have used the EMPIOT power measurement tool [38]. This board allows us to plug in the Raspberry Pi 3B+ with the USB accelerators connected and accurately measure power usage through different phases of the system processes. In this section, we test two main phases of the decision making stage of our system: the setup, and inference. In these tests, we completed 30 iterations of the setup and inference power usage while averaging the information gathered. We completed these tests on the Intel Neural Compute Stick, the Google Coral Accelerator, and by running a TensorFlow Lite model on the Raspberry Pi 3B+ CPU as a control similar to the performance testing. We did not perform any power tests on the Ultra96 due to the power requirements of the Ultra96 being too high to test with the EMPIOT power measurement tool.

In our first test, we use GPIO interrupts to trigger power measurements at the beginning and end of the setup period for the inference engine on all devices. This gave us an accurate view of exactly how much energy was used by the different devices through the same phases of their processes. The results are presented in Figure 7. The standard deviations of our measurements for the different devices are also shown in Figure 7.

These figures show that the Google Coral Accelerator uses more energy while loading the TensorFlow Lite model during the setup phase. We also test this process on the Raspberry Pi 3 B+ CPU and see that it uses significantly less energy than either other device at the expense of speed. Finally, the Intel Neural Compute Stick offers a trade off between energy consumption and speed. In the next test we use the same procedures on the EMPIOT board to measure the energy

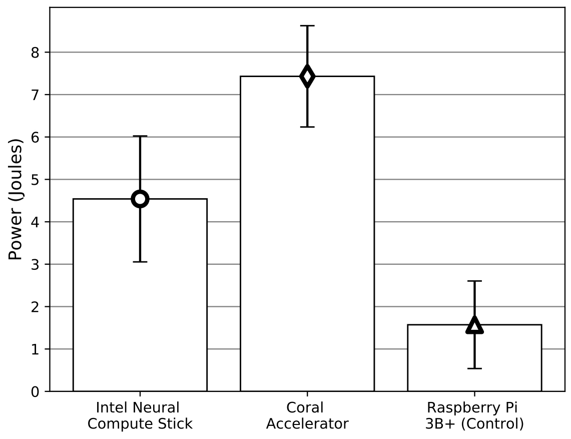

Fig. 7. Setup power usage: the average amount of power used by the device during the initialization process. The Google Coral Accelerator uses more power than the Intel Neural Compute Stick.

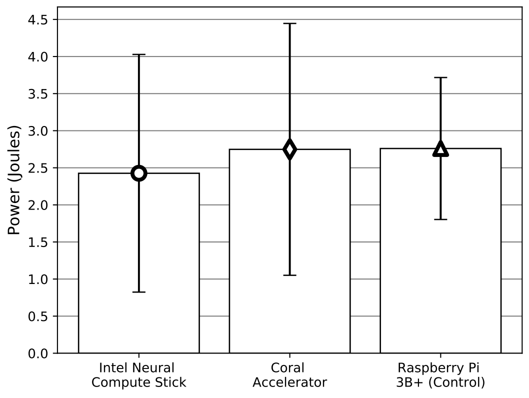

Fig. 8. Inference power usage: the average amount of power the device uses during inference. All three devices use a similar amount of power, but the Coral Accelerator uses slightly more than the others.

usage during an inference operation. These results are shown in Figure 8.

These results show that all three devices use a similar amount of energy to perform inference. However, the Intel Neural Compute Stick uses slightly less energy than the Coral Accelerator and the Raspberry Pi 3 B+ control test. The standard deviation of the measurements on the Coral Accelerator are also more significant than the Intel Neural Compute Stick. Our power testing has shown that a 4000 mAh battery can reach up to 5 hours of continuous battery life, while a 10,000 mAh battery can reach up to 12.6 hours of continuous battery life. Our system is completely flexible, and allows the user to choose the size of the battery that fits their usage and lifestyle.

D. Distance Measuring

As we mentioned in Section II, both sensors that we compared, the MB1040 LV-MaxSonar-EZ4 and the TFmini,

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.

are approximately the same physical size and consume similar power according the official product specification sheets. The MB1040 has a maximum range of approximately 21 feet and a minimum range of 6 inches. The benefit to using an ultrasonic sensor is that the beam is wider than an infrared pulse, making it more likely to find the object detected in the inference stage.

The TFmini infrared sensor is more accurate because it has a narrower and more direct beam. Furthermore, the TFmini is the more cost efficient option costing $10 less than the MB1040 at the current time. The MB1040 has a maximum range of approximately 39 feet and a minimum range of 1 foot. The problem with the TFMini sensor, is that the narrow beam makes it easier to miss the object detected during the inference stage. The sensor must line up exactly with the object, or the sensor will get in inaccurate or incorrect reading.

IV. Conclusion

In this paper we present NavSense, a device capable of providing important navigational information to visually impaired users on the edge. This paper has discussed the benefits of several different system components including hardware visual processing accelerators and edge micro-controllers. Using measured data for performance speeds and power usage, we identified the best components to be used in conjunction to make NavSense as fast and energy efficient as possible. For the hardware accelerators, the Intel Neural Compute Stick and Coral were unable to detect a number of different objects in each image, so a more comprehensive dataset should be used to train a new model in the future to further improve the amount of inferences in a scene.

The Intel Neural Compute Stick is far faster in terms of start-up speed compared to the Coral Accelerator. However since this duration is only taken into account once every time the program is started, we do not think that if we were to use the Coral it would negatively affect the user experience. The inference times for the Intel Neural Compute Stick are slightly faster than the Coral Accelerator, however for all intents and purposes, they are about the same. The control group shows significant slowdowns when processing inferences, and we believe that since the user can decide whenever they want to run an inference through the push of a button, that the inference should be as fast as possible to reduce confusion caused by a disassociation between the press of the button and a pause while the program is running. For this reason, we will not be using the Raspberry Pi 3B+ alone to do the processing. The Intel Neural Compute Stick model was also slightly more accurate than the Coral. For these reasons, we will be pursuing the Intel Neural Compute Stick or Google Coral Accelerator with the Raspberry 3B+ for the end system. However, we plan to continue testing with the newly released Raspberry Pi 4 and on the Raspberry Pi Zero W when official library support is released.

NavSense achieves a near real-time, affordable, open source, and portable object detection and recognition system. This system, designed to aid the blind and visually impaired, can revolutionize the freedoms of those with visual impairments in countries and environments typically less accessible. NavSense offers important information to the user completely isolated from network connectivity and fully on the edge. It can provide accurate object recognition predictions at a rate that allows for real world navigation. In conjunction with other navigation technologies such as a white cane or guide dog, NavSense can aid the visually impaired in improving their quality of life and the safety of their transit. By making use of new technologies in the fields of machine learning and edge computing, our cutting-edge device is unlike any others seen throughout our research, offering more context and more direct information to the user at any given time.

Our open source model allows us to share our work with the world so that people in all communities can benefit from our research and development to recreate similar systems. We hope that NavSense will not only benefit the computer vision and machine learning fields, but also those with visual impairments that may use the system to aid in everyday life.

Acknowledgment

We would like to acknowledge the Engineering Undergraduate Program Senior Design Project Funding at Santa Clara University for their financial support. We would also like to thank Anxech – our partners in Puebla, Mexico – and Xilinx for their financial and academic support throughout the project.

References

[1] "Vision impairment and blindness," *World Health Organization*, October 2018.
[2] R. Manduchi and S. Kurniawan, "Mobility-related accidents experienced by people with visual impairment," *Insight: Research and Practice in Visual Impairment and Blindness*, vol. 4, 01 2011.
[3] "Visual impairments," *Health Policy Institute — Georgetown University*.
[4] R. Mastachi Torres, "Interview with acrip association," 11 2018.
[5] K. Elleithy and W. Elmannai, "Sensor-based assistive devices for visually-impaired people: Current status, challenges, and future directions," *Sensors*, vol. 17, pp. 1–42, March 2017.
[6] S. Inc., "Sunu," 2017. https://www.sunu.io/en/index.html.
[7] "Buying an ultracane." https://www.ultracane.com/buy_an_ultracane.
[8] "Raspberry pi 3 model b." https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.
[9] "Camera module v2," 2018. https://www.raspberrypi.org/products/camera-module-v2/.
[10] S. A. Magid, F. Petrini, and B. Dezfouli, "Image classification on iot edge devices: Profiling and modeling," *arXiv preprint arXiv:1902.11119*, 2019.
[11] "Getting started with the ultra96," Mar 2019. https://www.96boards.org/documentation/consumer/ultra96/getting-started/.
[12] "Dev board — coral." https://coral.withgoogle.com/products/dev-board/.
[13] "Intel neural compute stick 2," Aug 2018. https://software.intel.com/en-us/neural-compute-stick.
[14] "Usb accelerator — coral." https://coral.withgoogle.com/products/accelerator/.
[15] "Logitech c920 pro stream hd webcam." https://www.logitech.com/en-hk/product/hd-pro-webcam-c920.
[16] "Mb1040 lv-maxsonar-ez4." https://www.maxbotix.com/Ultrasonic_Sensors/MB1040.htm.
[17] A. Industries, "Tfmini infrared time of flight distance sensor." https://www.adafruit.com/product/3978.
[18] "Opencv library," 2018. https://opencv.org/.
[19] B. Thorne, "Introduction to computer vision in python," *The Python Papers Monograph*, vol. 1, Jan 2009.
[20] I. Amirtharaj, T. Groot, and B. Dezfouli, "Profiling and improving the duty-cycling performance of linux-based iot devices," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–29, 2018.

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.

[21] chuanqi305, “Mobilenetv2-ssdlite,” May 2018. https://github.com/chuanqi305/MobileNetv2-SSDLite/tree/master/coco.

[22] “Yolo vs. ssd: Deep learning and precise object detection method,” Feb 2019. https://technostacks.com/blog/yolo-vs-ssd/.

[23] M. Hollemans, “Mobilenet version 2.” https://machinethink.net/blog/mobilenet-v2/.

[24] “Rpi.gpio.” https://pypi.org/project/RPi.GPIO/.

[25] RapidWareTech, “Rapidwaretech/pyttsx,” Oct 2016. https://github.com/RapidWareTech/pyttsx.

[26] “Intel distribution of openvino toolkit,” May 2018. https://software.intel.com/en-us/openvino-toolkit.

[27] “Common objects in context.” http://cocodataset.org/#home.

[28] “Tensorflow,” 2018. https://www.tensorflow.org/.

[29] Xilinx, “Xilinx/pynq-computer vision,” Mar 2019. https://github.com/Xilinx/PYNQ-ComputerVision.

[30] Itdaniher, “itdaniher/aarch64-tensorflow.” https://github.com/itdaniher/aarch64-tensorflow/releases/tag/v1.5.0.

[31] Xilinx, “Xilinx/qnn-mo-pynq,” Oct 2018. https://github.com/Xilinx/QNN-MO-PYNQ.

[32] Xilinx, “Xilinx/bnn-pynq,” Mar 2019. https://github.com/Xilinx/BNN-PYNQ.

[33] Xilinx, “Xilinx/finn,” Feb 2019. https://github.com/Xilinx/FINN.

[34] J. Redmon, “Yolo: Real-time object detection.” https://pjreddie.com/darknet/yolo/.

[35] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.

[36] “Caffe.” http://caffe.berkeleyvision.org/.

[37] “Tensorflow lite — tensorflow.” https://www.tensorflow.org/lite.

[38] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, “Empiot: An energy measurement platform for wireless iot devices,” *Journal of Network and Computer Applications*, vol. 121, p. 135148, 2018.

Authorized licensed use limited to: Santa Clara University. Downloaded on August 02,2020 at 05:23:23 UTC from IEEE Xplore. Restrictions apply.