# PLACE-BASED NAVIGATION FOR AUTONOMOUS VEHICLES WITH DEEP LEARNING NEURAL NETWORKS

Magee, Ashleigh

Monterey, CA; Naval Postgraduate School

**NAVAL POSTGRADUATE SCHOOL**

**MONTEREY, CALIFORNIA**

**THESIS**

# PLACE-BASED NAVIGATION FOR AUTONOMOUS VEHICLES WITH DEEP LEARNING NEURAL NETWORKS

by

Ashleigh Magee

December 2019

| | |
|---|---|
| Thesis Advisor: | Xiaoping Yun |
| Co-Advisor: | James Calusdian |

| **REPORT DOCUMENTATION PAGE** | *Form Approved OMB No. 0704-0188* |
|---|---|

| | | |
|---|---|---|
| **1. AGENCY USE ONLY** ***(Leave blank)*** | **2. REPORT DATE** December 2019 | **3. REPORT TYPE AND DATES COVERED** Master's thesis |

| | |
|---|---|
| **4. TITLE AND SUBTITLE** PLACE-BASED NAVIGATION FOR AUTONOMOUS VEHICLES WITH DEEP LEARNING NEURAL NETWORKS | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Ashleigh Magee | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)** N/A | **10. SPONSORING / MONITORING AGENCY REPORT NUMBER** |

**11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| | |
|---|---|
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release. Distribution is unlimited. | **12b. DISTRIBUTION CODE** A |

**13. ABSTRACT (maximum 200 words)**

Accurate navigation is crucial to successfully deploying autonomous vehicles but is often limited by GPS reliance. The purpose of this thesis was to develop an image-based navigation solution that does not require GPS. This work builds on an ongoing research project, which is to develop a mobile robot that can navigate anywhere on the Naval Postgraduate School campus, inside and outside of buildings. This work focused on indoor navigation using image classification, and images were classified using a convolutional neural network (CNN). Transfer learning was used to reduce CNN training time and increase learning efficiency. The trained CNN was integrated into a waypoint loop algorithm that ran autonomously on the robot, stopping when it had correctly identified its desired location. Tests of the CNN classification showed a high success rate, but that it was also susceptible to variations in light and similar types of images. Increasing the data set will improve the classification results and allow for use in a variety of applications. This system will also benefit from improved indoor localization techniques.

| | |
|---|---|
| **14. SUBJECT TERMS** place-based navigation, neural networks, deep learning, AlexNet, autonomous, GPS, GoogLeNet, transfer learning | **15. NUMBER OF PAGES** 87 |
| | **16. PRICE CODE** |

| | | | |
|---|---|---|---|
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UU |

THIS PAGE INTENTIONALLY LEFT BLANK

# PLACE-BASED NAVIGATION FOR AUTONOMOUS VEHICLES WITH DEEP LEARNING NEURAL NETWORKS

Ashleigh Magee
Lieutenant, United States Navy
BS, U.S. Naval Academy, 2013

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**
**December 2019**

Approved by: Xiaoping Yun
Advisor

James Calusdian
Co-Advisor

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

## ABSTRACT

Accurate navigation is crucial to successfully deploying autonomous vehicles but is often limited by GPS reliance. The purpose of this thesis was to develop an image-based navigation solution that does not require GPS. This work builds on an ongoing research project, which is to develop a mobile robot that can navigate anywhere on the Naval Postgraduate School campus, inside and outside of buildings. This work focused on indoor navigation using image classification, and images were classified using a convolutional neural network (CNN). Transfer learning was used to reduce CNN training time and increase learning efficiency. The trained CNN was integrated into a waypoint loop algorithm that ran autonomously on the robot, stopping when it had correctly identified its desired location. Tests of the CNN classification showed a high success rate, but that it was also susceptible to variations in light and similar types of images. Increasing the data set will improve the classification results and allow for use in a variety of applications. This system will also benefit from improved indoor localization techniques.

THIS PAGE INTENTIONALLY LEFT BLANK

## TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| AI | artificial intelligence |
| AGV | autonomous ground vehicle |
| APF | artificial potential fields |
| CNN | convolutional neural network |
| EKF | extended Kalman filter |
| GPS | global positioning system |
| LIDAR | light detection and ranging |
| MLP | multi-layer perception |
| NPS | Naval Postgraduate School |
| P3-DX | pioneer 3-deluxe |
| ROS | robot operating system |
| SLAM | simultaneous localization and mapping |

THIS PAGE INTENTIONALLY LEFT BLANK

# ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Xiaoping Yun and Dr. James Calusdian. Their support and guidance were instrumental in the success of this research. Thank you both for helping me develop new approaches when I got stuck and for being a sounding board for my ideas. Dr. Yun, thank you for continually challenging me while supporting me through this process. Dr. Calusdian, thank you for your invaluable assistance throughout the research and experimentation phases of this thesis. You truly helped me maintain my sanity. I would also like to thank professor Brian Bingham for teaching me everything I know about ROS. I could not have done this without you.

Lastly, I would like to thank my husband, Ryan. You always made me laugh when I was frustrated, especially since the problem was never with the GUI.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

## A. BACKGROUND

The improvement of artificial intelligence (AI) and machine learning has resulted in their use across a variety of applications. Using neural networks and decision trees, programmers can feed these AI systems data and use the resulting trained network for probabilistic predictions and classifications. These systems are being used to improve commute times, determine consumer trends for more effective marketing, and auto-fill email responses to help us work more efficiently [1]. AI is also being used to improve autonomous decision-making, to include autonomous navigation.

There is a significant focus on autonomous navigation in both military and civilian applications. A critical part of the Navy "Design for Maritime Superiority" involves "high velocity outcomes" that include awarding contracts for autonomous surface vessels and unmanned aircraft [2]. Increased joint service operations will also require more autonomous ground vessels. These assets must be able to deploy in a variety of environments around the world, which can complicate the navigation and localization solutions required. Many of the current solutions rely on GPS data for localization and navigation. While GPS is extremely useful, it is limited by requiring a line of sight between the satellites and navigation system. This requirement excludes navigation systems that are required to operate indoors, underground, or in other areas of limited GPS connectivity.

## B. PURPOSE AND GOAL OF THIS THESIS

The purpose of this thesis is to develop an autonomous navigation solution based on image data and classification. This will alleviate the dependence on GPS for navigation and provide a solution for indoor and limited-connectivity environments. Indoor environments provide a unique challenge because there is little variance in the visual features. Most buildings consist of rooms, doors, and hallways, which limits the ability of the AI to distinguish unique characteristics. Training a robust system to identify indoor locations will ensure the program can expand to other environments with more characteristics.

Another use for this research is improving GPS-based navigation systems. GPS-based localization has an error radius of approximately three to ten meters, depending on the quality of the GPS receiver [3]. If a robot is required to drive autonomously in a space that has a smaller radius than the GPS error tolerance, it can potentially get stuck or confused with inaccurate localization. Adding an image classification component that updates the navigation solution will allow increased localization certainty and decrease the error radius. Also, if the environment has ambiguities in the map, an image component could allow ambiguity resolution. Consequently, navigation solutions will become more efficient and useful.

The first goal of this research is to develop a method for image classification that allows a robot to navigate based on imagery in a variety of environments. The method must be platform-agnostic, so it can ultimately be applied to other projects and applications. The primary approach to image classification currently used is training neural networks to identify different objects. This research leverages that approach using a pre-trained neural network and a method known as transfer learning to limit the processing and data required to create a useful tool.

The second goal is to program a robot to successfully navigate autonomously to a goal location and identify the location on imagery alone. This approach combines the image classification method from the first goal with object avoidance and an autonomous search method. The navigation solution allows a robot to autonomously search an area, stopping when it identifies its predetermined goal location based on vision.

## C. PREVIOUS WORK

This thesis research is part of an ongoing research project in the Control Systems and Robotics Lab in the Department of Electrical and Computer Engineering at the Naval Postgraduate School (NPS). The goal of this program is to develop a robot that can drive autonomously both outdoors and indoors across the NPS campus. This program started by investigating solutions for dynamic object avoidance. In [4], Calvin Hargadine used an artificial potential field approach in his object avoidance algorithm that is used in this research. Essentially, the robot is artificially repelled from objects it encounters, and drawn towards its goal in the form of waypoints [4]. Calvin Hargadine's thesis work also integrated the hardware and software for the robotic platform used in this research. Matthew Audette followed this work by creating a route-planning algorithm that accounts for known objects and relies on satellite imagery [5]. One of the primary limitations of the route-planning algorithm was the ability to identify traversable terrain. In [6], Caliph Lebrun developed an algorithm that used vision and machine learning to distinguish navigable terrain from non-traversable. Also, Alexander Miyakawa used multiple LIDAR scanners to build a three-dimensional map that classifies low-level obstacles by terrain gradient [7]. The image classification approach used in this research is designed to be integrated into the route-mapping planner that was created by Audette. Adding a terrain avoidance component to this image classification research could create a robust autonomous navigation solution, providing there is sufficient available computing power. Integrating the previous work with this research as well as research to follow will allow an autonomous ground vehicle (AGV) to successfully navigate the NPS campus.

A widely used solution in AI is the convolutional neural network (CNN). Using a pre-trained CNN for image and object classification has been used in an array of applications. Many research projects at NPS have used CNNs for still images, such as satellite imagery or logos to test the ability of pre-trained networks to decrease processing time and data set size [8], [9]. This research leverages the lessons learned to use a pre-trained neural network on live images and assess the results.

## D. THESIS ORGANIZATION

In Chapter II, we present the background and methods used in creating the image classification algorithm for this research. The methods investigated and utilized to generate autonomous movement for the robot are discussed in Chapter III. In Chapter IV, we provide a discussion of the results of experiments carried out in support of this research. Finally, a summary of research and a brief discussion of possible future work are presented in Chapter V.

THIS PAGE INTENTIONALLY LEFT BLANK

# II. IMAGE CLASSIFICATION PROCESS

A primary goal of this research is to develop a method for image classification that allows a robot to navigate based on imagery acquired from an installed video camera. In this chapter an overview of machine learning and its related concepts is presented, which is useful for understanding the research work described later in this thesis.

A variety of approaches are used to classify images. The basics of image processing, computer vision, and machine learning are nicely outlined in “A Guide to Convolutional Neural Networks for Computer Vision” by Khan et al. in [10]. They briefly discuss older methods of traditional image processing in which the goal is to isolate basic image characteristics such as edges or color. Today this is considered a way of pre-processing an image before it can be classified by more robust and sophisticated methods. Of greater interest to the authors is classifying objects in both static images and live video, which is considered a more complex form of image processing and is most commonly performed using machine learning. Machine learning allows an algorithm to learn from data without being programmed at each individual level. One of the machine learning approaches available is neural networks, of which there are different types. A discussion on neural networks, as well as the methodology for using these neural networks in the image classification problem will be provided in this chapter.

## A. NEURAL NETWORKS

One of the simplest architectures for neural networks discussed by Khan et al. is the Multi-Layer Perception (MLP) network [10]. MLP networks are essentially a “black box” or something that receives inputs and then generates a series of outputs. The input layer receives inputs, then a series of hidden layers conduct the processing required. Lastly, the data is fed to an output layer that makes predictions. Each hidden layer has different weights, which prioritizes the process for the desired output. In Figure 1, an illustration of a small or “shallow” neural network [10] is shown. As indicated in the book, the difference between shallow neural networks and deep neural networks is the number of layers within the network. Neural networks with more layers create structures that allow greater simplicity, scalability, and domain transfer or transferability of applications.

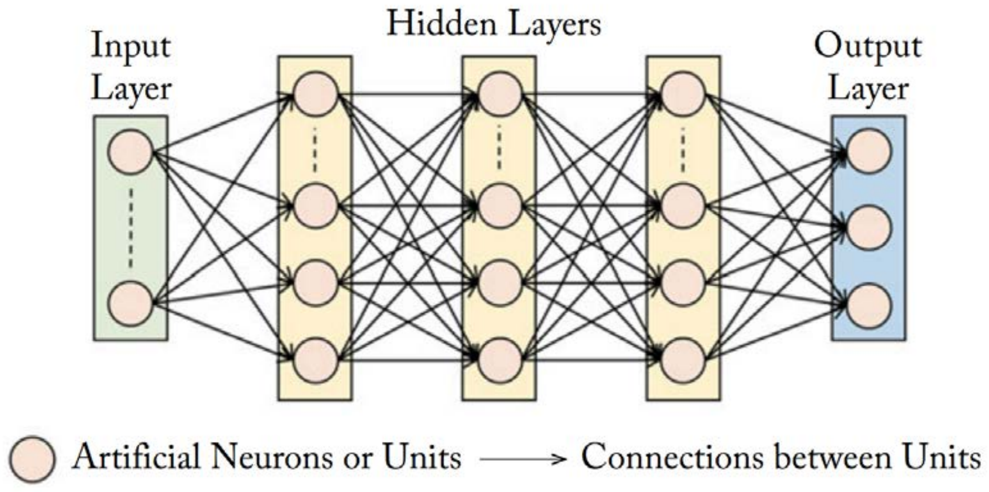


Figure 1. Illustration of a Small Neural Network. Source: [10].

When a neural network undergoes the training process, it uses an input of training data and updates the layer weights through multiple iterations. As explained by Khan et al., the weights within each layer are adjusted during each iteration of the training process until a desired level of performance is achieved, which is determined by minimizing a loss function [10]. The neural network is then tested or validated by submitting new data that was not included in the training data. Lastly, the output is the prediction based on the weights developed during the training process. One important consideration the authors stress is the step size used in the training process, especially for deep neural networks. The step size is the increment used to adjust the weights during each iteration of the training process. Consequently, it affects how long the training process takes to converge as well as how effectively the weights are updated. If the step size is too small, the training process will take an extremely long time, and there will not be significant gain in training effectiveness. However, if the step size is too big, the training process will not be able to effectively update all the weights in the layers, and the output will not be as successful.

## B. CONVOLUTIONAL NEURAL NETWORKS

The basic components of neural networks discussed in the previous section also apply to convolutional neural networks. Unlike neural networks, convolutional neural networks (CNN) have a convolutional filter that combines inputs with learned weights within each layer to generate an output layer [10]. A convolutional neural network was used for this research because it is capable of processing complex images and videos. Basic neural networks cannot process images and videos with as much success. Khan et al. discuss each of the CNN layers which typically consist of four components: convolutional filters or “kernels,” nonlinear activation functions, pooling layers, and fully-connected layers [10]. There is also a fifth component only used in the training process, the loss function.

### 1. Convolution Filter

As explained in [10], the convolution filter is a grid of discrete numbers that represent the weights learned during the network training process. This grid is multiplied by a section of the input and summed to generate a single output number. This filter slides and repeats the process until all the input has been processed, and an output grid is generated. This concept is demonstrated in Figure 2.

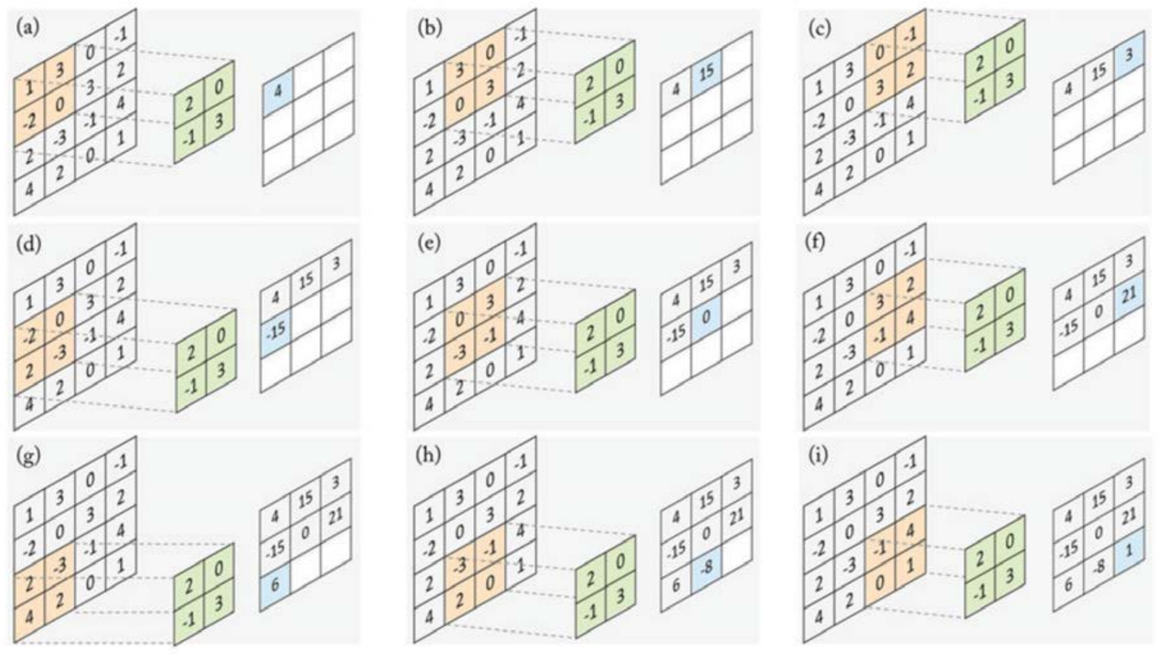


Figure 2. Convolution Process for 2x2 Filter. Source: [10].

Continuing the authors' discussion, the convolution process is essential because it reduces dimensions, called "sub-sampling," and increases invariance for changes in scale and pose of objects in the images [10]. The convolution filter tends to be significantly smaller than the input. This reduces the number of learnable parameters and ensures unique patterns stay related to their local area. There can be multiple convolution filters in each layer.

### 2. Nonlinear Activation Function

The activation function or "detection layer," as Goodfellow describes in [11], follows the weighted convolution layers and decreases the input to a small range of numbers. Khan et al. explains the function for this layer must be nonlinear because it allows the neural network to separate and learn from nonlinear data, as well as allow for more complex data analysis [10]. In a comparison to a biological neural network, the activation function is equivalently a "switch" for the neuron to fire or not.

### 3. Pooling and Fully Connected Layers

Following the nonlinear activation function is the pooling layer. This layer calculates the average or maximum value of the input, depending on its settings, which further down-samples the input [10]. This process creates further invariance to small translations in the data such as rotation or aspect ratio [11]. The pattern of convolution filters, activation function, and pooling layer repeats throughout the CNN architecture. In simple layer terminology, shown on the right in Figure 3, the convolution layer is described as a single layer followed by the detector layer. Complex layer terminology considers the combination of the pooling stage, detector stage, and convolution stage the convolution layer, as shown on the left in Figure 3. This research uses the simple layer terminology when describing CNNs.

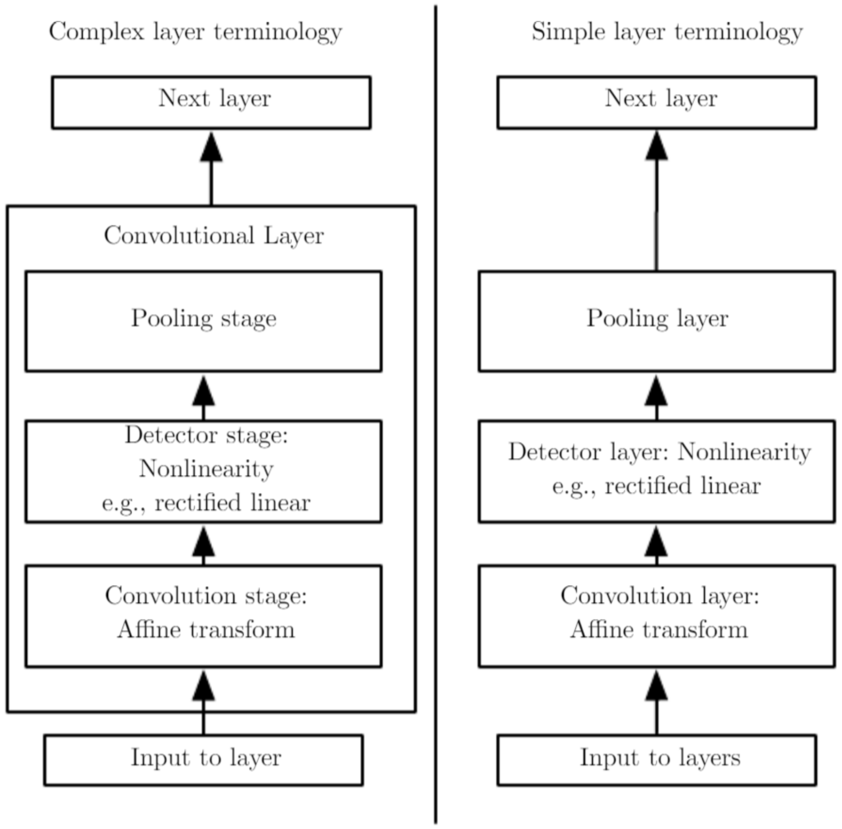

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Figure 3. Typical Convolutional Neural Network Layer Components. Source: [11].

#### 4. Loss Function

A variety of loss functions are used depending on the purpose of the CNN. The loss function layer is used only during the training process and estimates how well the CNN is making predictions based on the classification output [10]. The authors explain that the loss function determines the difference between the model prediction and the true output. The function used for multi-class classification is the soft-max loss function, often simply referred to as soft-max. Because the focus of this research is image classification, the CNNs discussed will have a soft-max loss function.

## C. DETERMINING WHICH CONVOLUTIONAL NEURAL NETWORK TO USE

Programmers and computer vision experts began increasing the widespread use of CNNs after the success of AlexNet in 2012 [12]. Consequently, a variety of open-source CNN architectures are available for research and experimentation. For this research, possible candidates were restricted by availability and ease of integration. AlexNet and GoogLeNet were chosen for testing, partially due to their easy integration into tools in MATLAB. Both are convolutional neural networks but have different architectures. In this section the differences between the networks and why they are optimal testing candidates for this thesis work are discussed.

#### 1. AlexNet Background

AlexNet was created to perform in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012, as detailed in [13]. Significantly, it was the first time a CNN performed better than traditional neural networks in image classification [12]. For the competition, AlexNet classified 1.2 million images into 1000 classes during training, validated against 50,000 images, and tested on 150,000 images [13]. The top-5 error rate, which is the percentage of classifications that do not have the correct label in the top-5 predictions, was 15.3% for the competition. The architecture has five convolutional layers and three fully-connected layers as its basis [13]. The unique features in its architecture are its nonlinearity function and overlapping pooling. The nonlinearity function used was the

Rectified Linear or ReLU function, which is favored by CNN creators due to its quick computation that increases training efficiency and is shown in Figure 4.

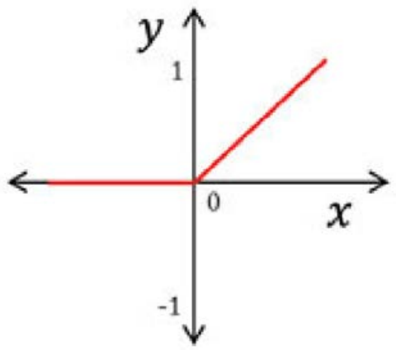


Figure 4. Graphical Representation of ReLU Nonlinear Function. Source: [10].

AlexNet's use of ReLU allows for faster training times due to decreased computation [13]. The max pooling layers are overlapped because doing so reduces error and makes it slightly more difficult to overfit the data. The full architecture is shown in Figure 5. Of note, the image is cropped in the original academic paper, and a full image is unavailable.

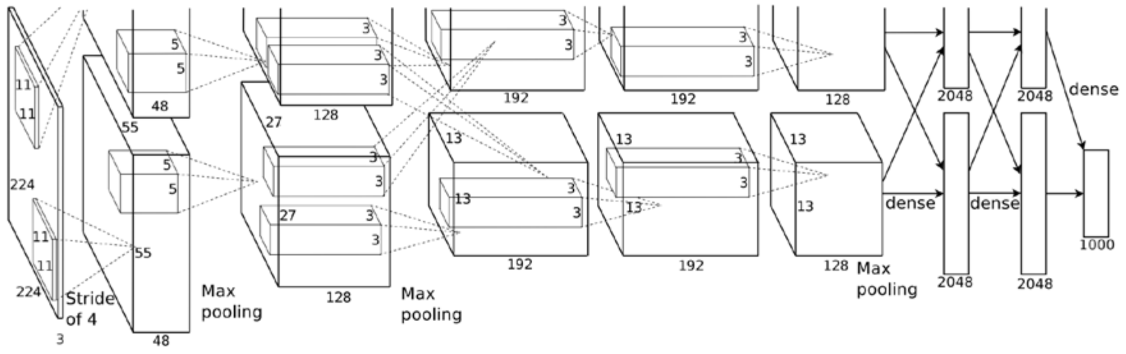
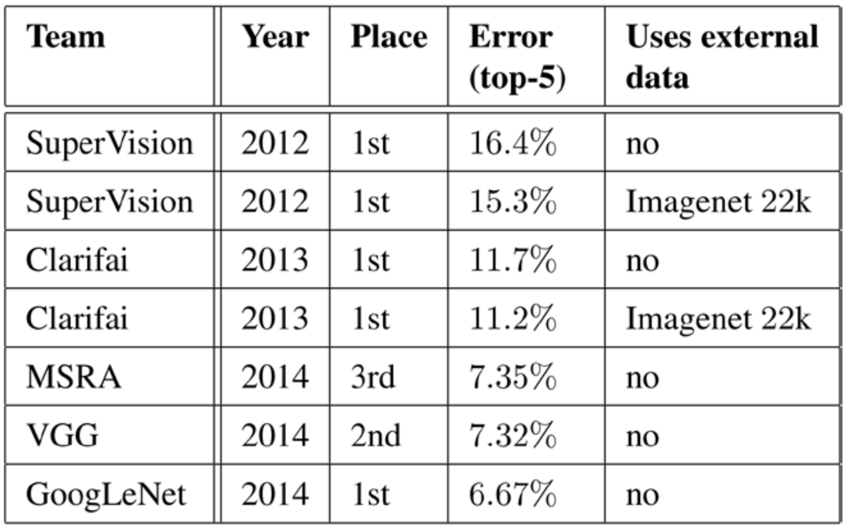


Figure 5. AlexNet Architecture. Source: [13].

### 2. GoogLeNet Background

GoogLeNet was created for ILSVRC 2014, focusing on a high-accuracy, low-computation solution that could be used in the real world, rather than be restricted to research and academia [14]. The contest parameters were like those in the 2012 challenge, changing only the number of test images to 100,000. GoogLeNet significantly outperformed the competition and previous winners with a top-5 error rate of 6.67%. A comparison of GoogLeNet to the previous top-performers is shown in Figure 6. Of note, AlexNet's performance is the second SuperVision entry shown in Figure 6.



| Team | Year | Place | Error (top-5) | Uses external data |
|---|---|---|---|---|
| SuperVision | 2012 | 1st | 16.4% | no |
| SuperVision | 2012 | 1st | 15.3% | Imagenet 22k |
| Clarifai | 2013 | 1st | 11.7% | no |
| Clarifai | 2013 | 1st | 11.2% | Imagenet 22k |
| MSRA | 2014 | 3rd | 7.35% | no |
| VGG | 2014 | 2nd | 7.32% | no |
| GoogLeNet | 2014 | 1st | 6.67% | no |

Figure 6. Classification Comparison, 2012–2014. Source: [14].

GoogLeNet architecture is significantly different from most CNNs because of its inclusion of the "Inception Module" [14]. Instead of guessing what size and type of convolutions best fit the training, the inception module iterates through all options and determines the optimal configuration. Two versions of this module are shown in Figure 7. The top module is the simple version, and the bottom module contains dimensionality reduction.

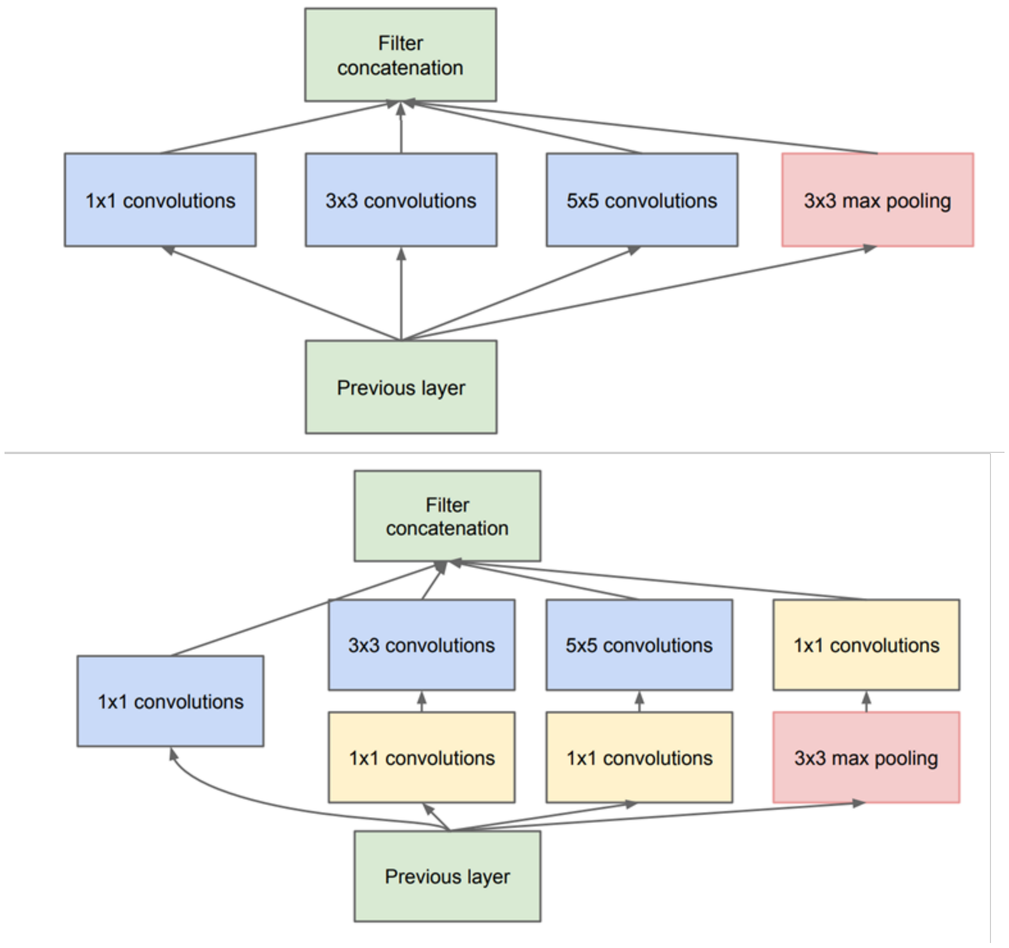


Figure 7. GoogLeNet Inception Module Illustration. Adapted from [14].

The inception module combines with the traditional CNN layers of convolution, max pooling, average pooling, and fully connected. GoogLeNet also used ReLU for the nonlinearity function, and a soft-max layer is used during the training process [14]. The resulting CNN uses 12 times fewer parameters than AlexNet but significantly increases accuracy [15].

### 3. Transfer Learning

One of the benefits to both AlexNet and GoogLeNet is the ability to use transfer learning. Transfer learning is a machine learning approach where a pre-trained neural network is trained with new data for specific categories but maintains its general classification abilities. The benefit of this approach is that it eliminates the requirement to train a network from scratch to classify for one specific problem. The resulting neural networks are more robust and useful for a variety of tasks. Furthermore, if the parameters of the problem or the data changes, the network can be easily retrained to assimilate the updated data and generate classifications accordingly.

To conduct the transfer learning, MATLAB has built-in commands that enable the user to load the net and rewrite the desired layers for the transfer process. This process involves loading the training data using *imageDatastore*, which stores the images and labels them based on their subfolder names. The labels are also used later in the classification process. The loaded pre-trained neural net is then converted to a *layerGraph*, allowing the individual net layers to be viewed and manipulated. Using the command *findLayersToReplace*, the MATLAB tool analyzes the net layers and determines which ones can be retrained. The training layers then have the weights replaced so they can learn from the new data. After the new layers have been updated, the weights are frozen to avoid overfitting the data. Once the layers are ready, the MATLAB tool *trainNetwork* is implemented to train the adjusted layers with the new data set. After the network is trained, a selection of validation images is used to check the accuracy of the classification training. Because MATLAB has these tools available for both AlexNet and GoogLeNet, they are both excellent candidates for testing.

## D. TESTING CONSTRAINTS

AlexNet and GoogLeNet were chosen for initial testing because they were open-source and easily integrated into MATLAB tools. This allowed for quick, small-scale proof-of-concept testing because there was not a large programming barrier for use. They also had proved to be highly adapted to the image classification problem. However, multiple research constraints needed to be addressed prior to their full implementation.

First, the scope of the research needed to be assessed compared to CNN capability. Both AlexNet and GoogLeNet are capable of classifying thousands of images; however, this research only required a few different classification categories. While a simpler CNN could have been used, one of the goals for this research was to produce a solution that can be integrated into other projects. For this reason, maintaining the ability to classify many images and categories was desirable.

The CNN also needed to be capable of running on live images. Most CNNs are trained and tested on a database of static images. This research had to determine if AlexNet and GoogLeNet were computationally fast enough to classify images from a live video stream and output the result. Because the platform is a moving robot, the results needed to be available and processed at a speed that could be used to inform real-time navigation.

The final constraint was processing power limitations. This research was conducted on a SlimPRO SP675P Mini PC. The computer ran an Ubuntu 14.04 Long Term Support operating system. From the vendor's website, the computer is small (14.6 x 25.4 x 4.2 cm), so it is widely used in robotic applications [16]. The specifications also include a processor that is a 3rd generation Intel Core Central Processing Unit (CPU). Given the age of the processor, testing needed to determine if the SlimPRO could handle the CNN processing as well as other navigation algorithms and sensor load.

## E. SUMMARY

In this chapter we presented a brief overview of CNNs, which are a popular type of machine learning method. AlexNet and GoogLeNet—two well-known variations of CNNs for image classification—were introduced in this chapter. These two neural networks were selected for this research because of their good performance and are easily integrated into MATLAB. This research work also uses a machine learning method known as transfer learning. It allows for the customization of a general neural network, such as AlexNet or GoogLeNet for example, into a specialized neural network application with a minimum amount of training required. This allows for ease of testing, which makes it the most efficient approach to the image classification problem.

THIS PAGE INTENTIONALLY LEFT BLANK

# III. ROBOT AUTONOMOUS NAVIGATION

A variety of methods are available for autonomous navigation. The greatest considerations for choosing a method are platform, ease of use, and desired goal. Many of the available methods require a specific start and end point, calculating a path between the two. For the purpose of this research, only methods that could be used in a search capacity were explored.

## A. ROBOTIC PLATFORM

The robot used was an Omron Adept MobileRobots Pioneer 3-DX (P3-DX) platform, which is different than the Pioneer 3-AT (P3-AT) used in previous thesis work. This platform varies from those used in previous thesis work because it is suited only for indoor terrain. The primary difference in platforms is the wheels. The P3-DX has two side wheels and a third caster wheel in the front whereas the all-terrain P3-AT has four wheels total, and a more rugged tire tread. The P3-DX is driven by a differential steering system with one encoder per wheel, resulting in 92 counts/degree accuracy [17]. The P3-DX has an extensive sensor suite and programming capabilities. The robot bumpers and sonar arrays were used in this research, as well as an additional LIDAR sensor and web camera that were connected and mounted separately. The primary limitation of this platform is that it is no longer manufactured, so software support is limited. In Figure 8 the base version of the P3-DX is shown, and in Figure 9 the P3-DX used for this research is shown.  This extensively modified robot has an onboard computer with Linux Ubuntu 18.04 in addition to the LIDAR and webcam mentioned previously.

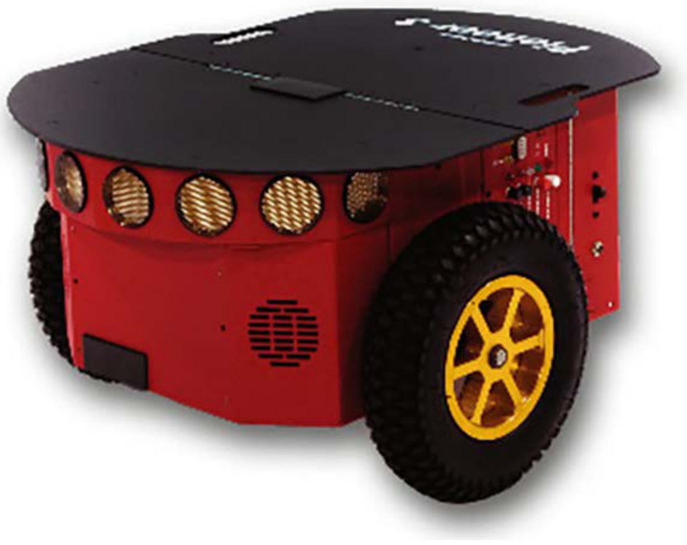


Figure 8. Base Pioneer 3-DX. Source: [18].

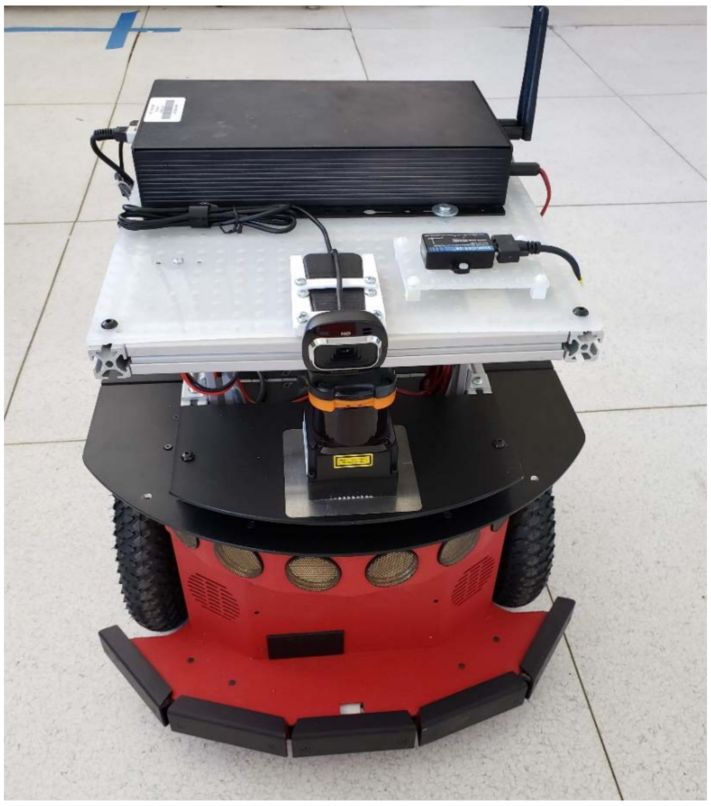


Figure 9. Pioneer 3-DX Used in Research

## B. ARTIFICIAL POTENTIAL FIELDS

Artificial Potential Fields (APF) are used in a variety of autonomous navigation applications because of their obstacle avoidance capabilities. Prior thesis work done by Hargadine used this method for dynamic obstacle avoidance in an unknown environment [4]. The simplest way to model APF is by picturing a neutral plane. A goal is added to the plane as a negative charge. Obstacles are represented by positive charges. Lastly, the robot is represented by a positive charge. The negative charge of the goal creates an attractive force on the robot whereas the positive charges of the obstacles create a repulsive force. The vector sum of the attractive and repulsive forces gives the robot trajectory. An example of this effect is shown in Figure 10. Jean-Claude Latombe describes the implementation of this method in *Robot Motion Planning* [19].

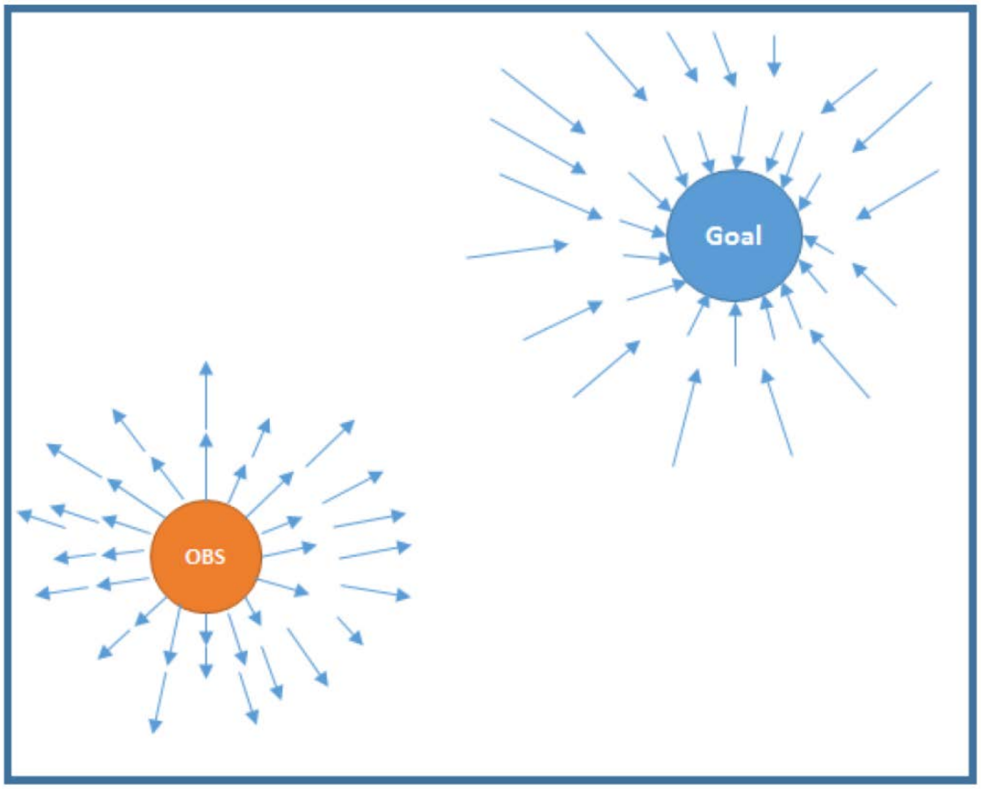


Figure 10. Attractive and Repulsive Potential Field. Source: [20].

APF are used in two applications for this research. The first is in object avoidance, using the same algorithm developed by Hargadine in [4]. The second is a wandering algorithm. The wandering algorithm creates an autonomous navigation solution that is applicable for searching for a target location. Using the methods described by Latombe in [19], the MATLAB algorithm calculates a repulsive force based on feedback from the P3-DX sonar returns. A forward force is also generated to move the robot forward. The repulsive force combines with the forward force for a total force. This total force is filtered to smooth the results, and the smoothed force combines with a translational gain for forward velocity and rotational gain for rotational velocity. These methods are both easily programmed and tested, making them ideal candidates for this research.

## C. ROBOT OPERATING SYSTEM (ROS)

ROS is a Linux-based open-source software that enables programmers and engineers to create code for controlling robots. Its flexible framework includes many libraries, tutorials, and programs for use across a variety of robotic platforms. It is considered the primary method of programming robots within the robotics discipline.

### 1. Navigation

Within the ROS navigation resources are two primary packages of interest: *move_base* and *frontier_exploration*. *Move_base* allows a user to enter  a goal position, and the robot determines a path to reach that goal [21]. It can create *costmaps* as well to avoid both static and dynamic obstacles. *Costmaps* are grids that indicate to the robot what is free space, and what is occupied by an obstacle. In order to do this, the package must be used with a LIDAR sensor. While the *move_base* package alone is a point-to-point navigation solution, when paired with the *frontier_exploration* package, the robot can search an area. *Frontier_exploration* allows a user to enter an exploration goal as a polygon within an area [22]. It can be used with or without a pre-defined map. Upon receiving the goal, it sends movement commands back to *move_base* in order to move the robot. A snapshot from a demonstration of *frontier_exploration* is shown in Figure 11. The virtual environment shown is through the ROS Visualization (RVIZ) utility, which can create model simulations, as well as display real-time robot sensor data. In this image, the robot was not given a pre-determined map, so it is creating a map based on its sensors. The outermost teal polygon is the given exploration area. As the robot explores, it creates a *costmap* from obstacles, shown in red, blue, purple, and teal. Open area is grey, and the small red dots are smaller obstacles. Through the course of its exploration, it will fill in all the map space within the polygon, unless it determines it is trapped within a room and can explore no further.

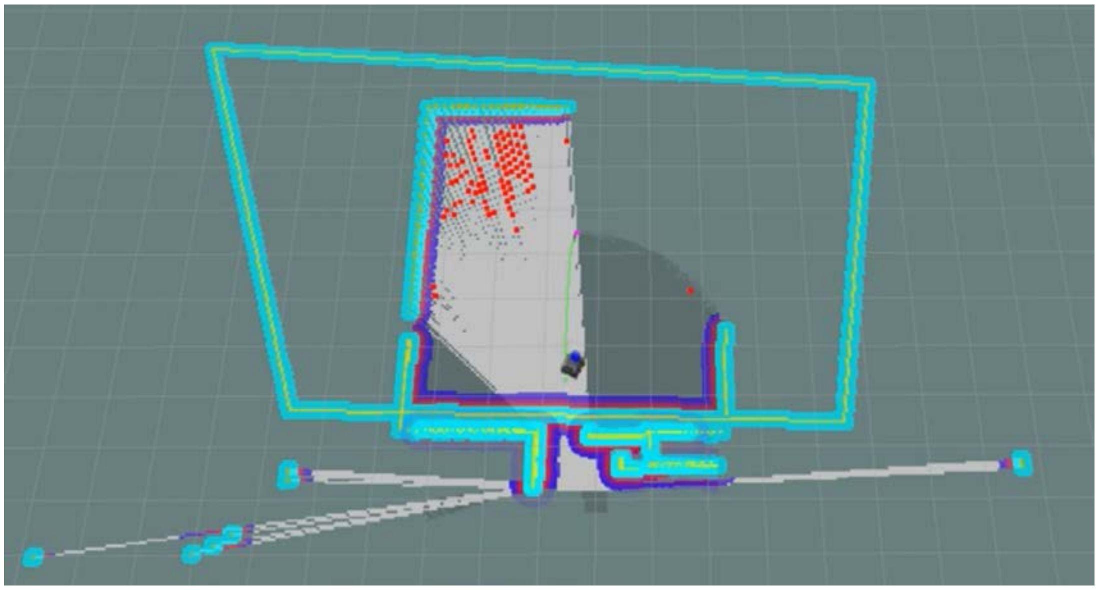


Figure 11. Snapshot of *frontier_exploration* Demonstration. Adapted from [22].

### 2. Constraints

While the ROS packages are robust and suited for the goals of this research, they are constrained by the hardware available. Most ROS software is written for a handful of robotic platforms. However, because the P3-DX is no longer supported, ROS integration is not readily available. While the P3-DX does run a ROSAria node, developed from previous thesis work, further navigation implementation is beyond the time and experience scope for this research.

## D. SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

SLAM is a process used in robotics to both determine the robot location and map the surrounding area based on sensor data. As explained by Riisgaard and Blas in their tutorial "SLAM for Dummies," the process is driven by the Extended Kalman Filter (EKF), which updates where the robot thinks it is based on surrounding features [23]. This process is shown in Figure 12. Essentially, the robot reports its current odometry data and feeds it as an input into the EKF. Simultaneously, the robot uses a LIDAR sensor to scan the area. The LIDAR data is also fed as an input into the EKF. The EKF re-observation block evaluates the data from known landmarks or objects and uses changes in their position to update the robot position. The EKF new observation block inputs new landmarks not previously seen to be used for later re-observation. This allows the EKF to reconcile the differences between the odometry and LIDAR data and create a new prediction for current location and update the error associated with that prediction.

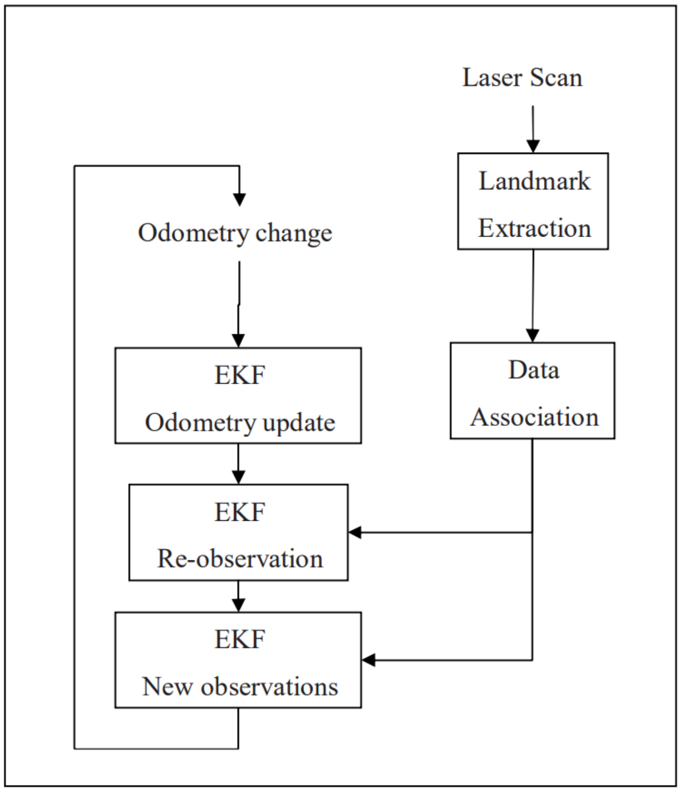


Figure 12. Overview of the SLAM Process. Source: [23].

### 1. Uses for SLAM

SLAM is useful for this research because it relies on localization from LIDAR data instead of GPS connectivity. It also has many applications. While SLAM is often used to improve path planning algorithms, its primary use is creating maps of unknown areas. The maps are generated from overlaying odometry and LIDAR data over time. The robot pose data can then be extracted from the generated map and used for waypoints in preprogramming search paths.

### 2. MATLAB SLAM Tool

Included in the MATLAB Robotics Toolbox, the MATLAB *SLAM Map Builder* tool generates a SLAM map from loaded LIDAR data. From the Mathworks website, the SLAM tool uses an algorithm to build the map, accounting for inputs such as max LIDAR range and desired map resolution [24]. This algorithm includes an essential component called loop closure. Loop closure recognizes previously visited areas while mapping and corrects pose data to account for drift. Two critical parameters impacting loop closure is *loop closure threshold* and *loop closure search radius*. *Loop closure threshold* helps filter false positives from the system. In an area with similar or repeated features, *loop closure threshold* should be set to a high number to reject the increased number of false positives [24]. Also described on the Mathworks website, *loop closure search radius* determines the range of the map search for loop closures. This tool not only generates a map but also outputs optimized poses. These poses are corrected for odometry drift, making them ideal to use in generating waypoints. An example SLAM map from this tool is shown in Figure 13, where the pink lines are obstacles and the blue lines are the optimized pose data.

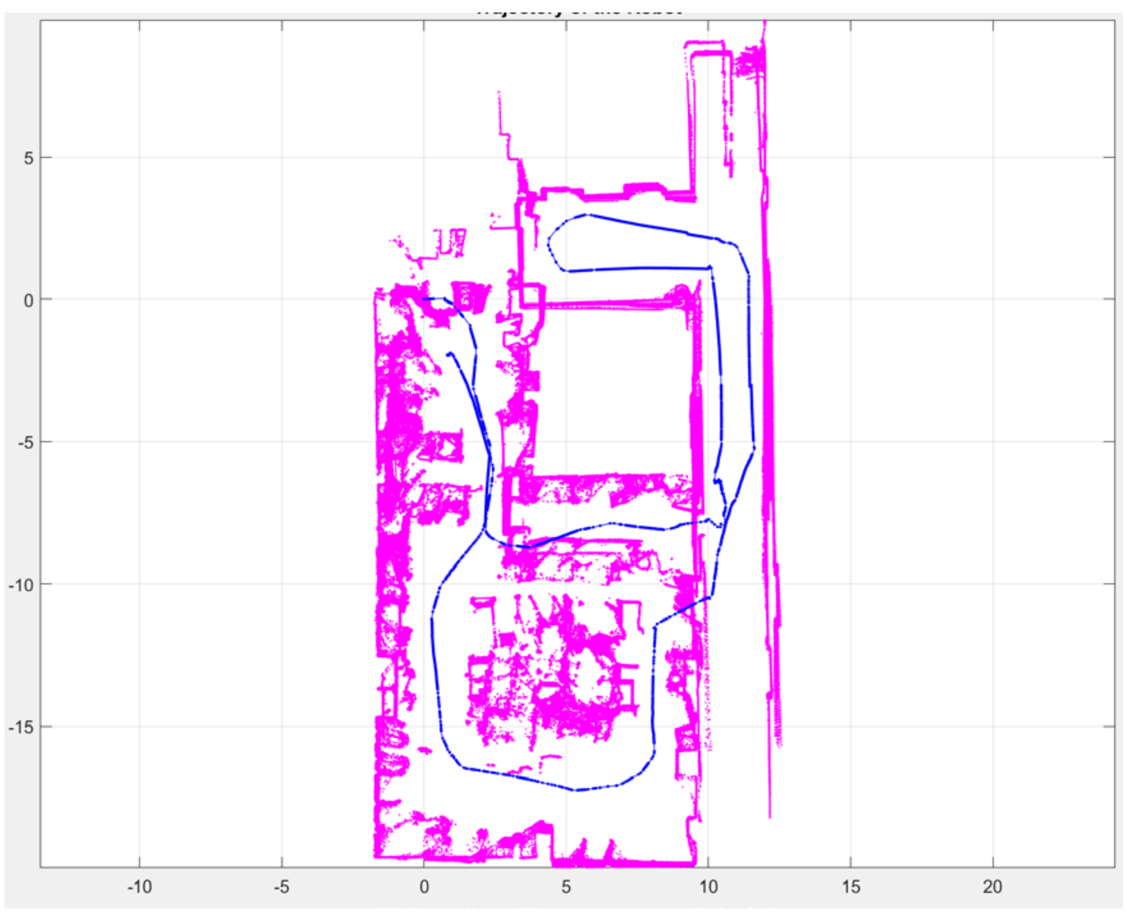


Figure 13. MATLAB-Generated SLAM Map

## E. WAYPOINT SEARCH PATHS

Path planning and waypoints are generally used in point-to-point navigation. Much of the existing research is focused outdoors and uses GPS for localization and waypoint following. Audette used this method for path planning in [5]. However, paths are also used in searching. If the area required to search can be defined by a simple path, path planning algorithms are useful in implementing a successful search. Because this research occurred indoors, there are limited methods for determining waypoints. The first is to manually measure waypoints. While this is the simplest method, it does not account for odometry error that can result in drift from the original calculations. A second method is using the MATLAB SLAM tool discussed above. Because the tool generates optimized poses that are corrected for odometry drift, the pose data is more accurate and suitable for use as waypoints.

Once the search path is determined, an algorithm is developed to generate the waypoint-following behavior. In this research, the MATLAB tool *pure pursuit* was used. *Pure pursuit* takes an input of waypoints and current pose. It then computes linear and angular velocities to reach the next waypoint. This tool is ideal for path-following that does not require the robot to stop at each waypoint. From the Mathworks website, the most important parameter in *pure pursuit* is the look ahead distance [25]. The look ahead distance is considered the local goal and tells the robot how far to look ahead on its path. This parameter is important because a small look ahead distance will cause the robot to oscillate on the path, and a large look ahead distance may miss the waypoint entirely. This concept is illustrated in Figure 14.

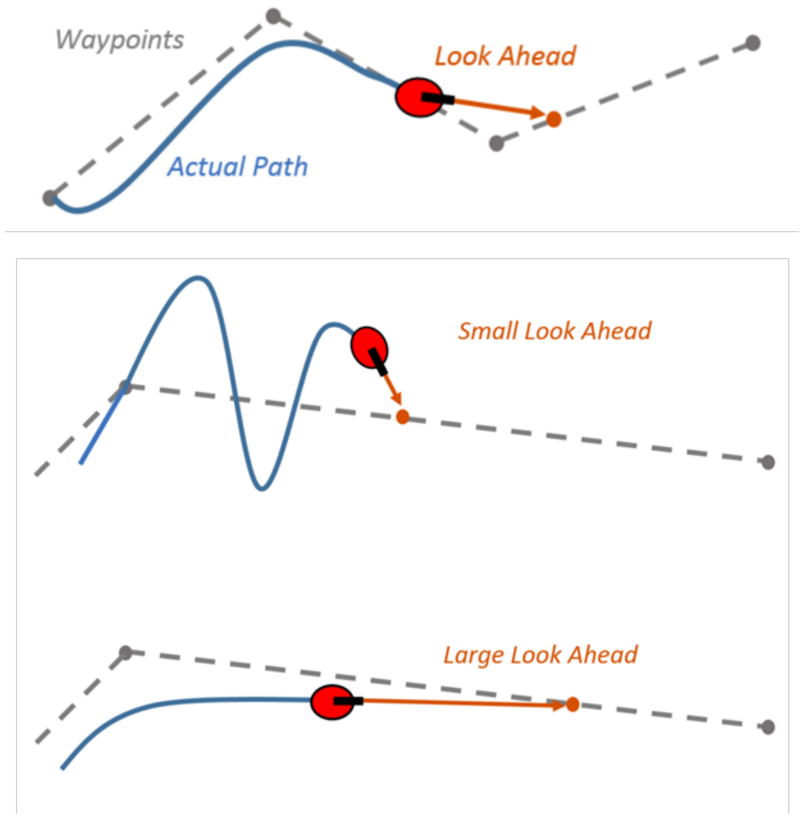


Figure 14. Look Ahead Distance. Adapted from [25].

Using the *pure pursuit* tool, the robot can follow a pre-programmed path autonomously while simultaneously conducting a search of the area. Due to the ease of implementation, this tool is ideal for this research.

## F. SUMMARY

This was an introduction to the robotic platform used as well as various methods for autonomous navigation. While the ROS navigation packages initially seemed ideal, their lack of support for the P3-DX platform proves them too time costly. APF provides a simple method for object avoidance and wandering behavior. This research work also uses multiple MATLAB tools. The ease of integration using the MATLAB *SLAM map building* and *pure pursuit* tools create a feasible approach for generating and implementing a search path.

# IV. RESULTS

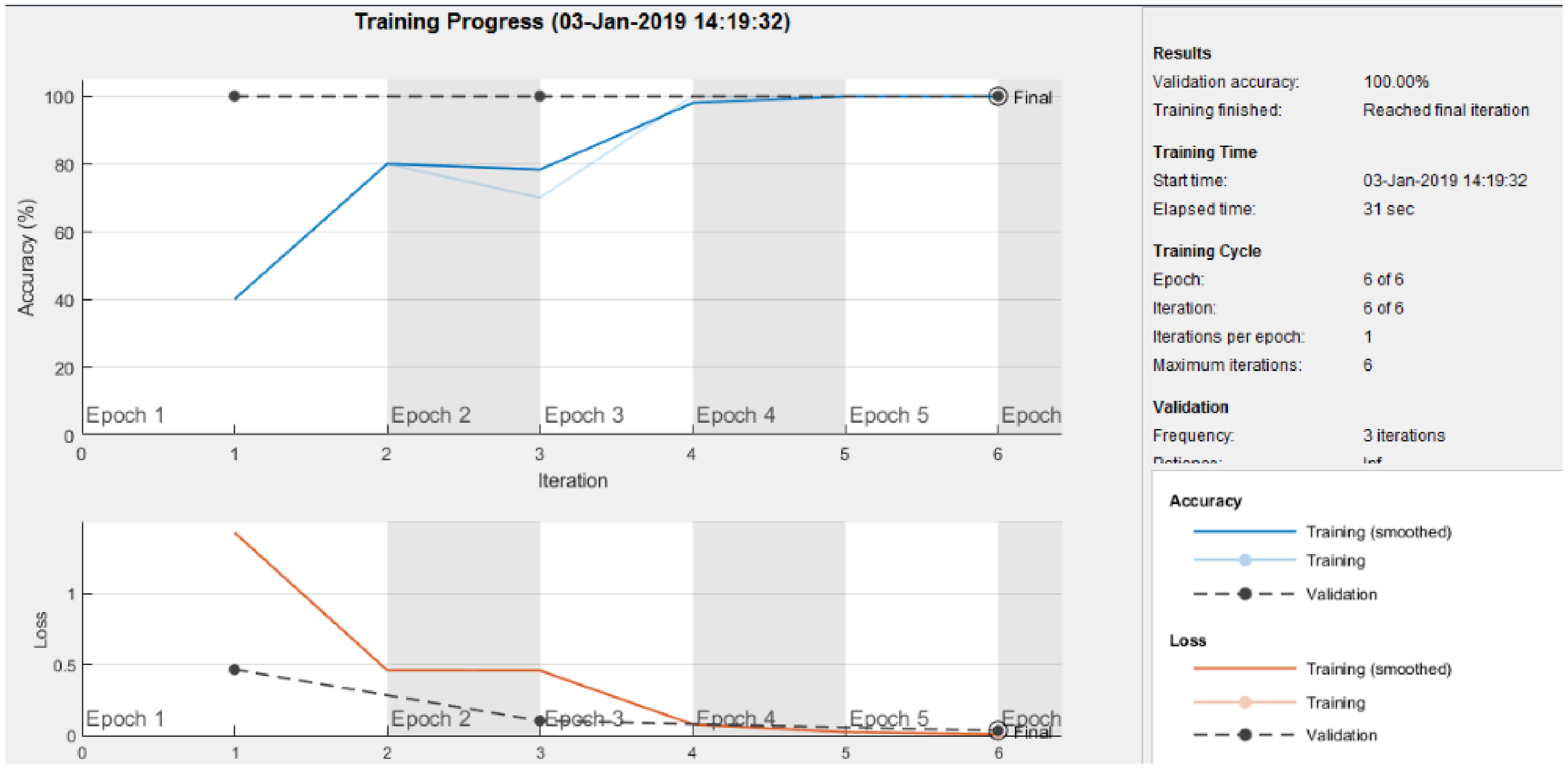
The general approach to achieving the goals of this research consisted of three parts: create a system for image classification, determine and implement an autonomous navigation solution, and integrate the software and hardware components into the P3-DX mobile robot platform.

## A. TRANSFER LEARNING AND CLASSIFICATION

In developing an indoor navigation solution, the priority was the image classification problem. Both AlexNet and GoogLeNet were tested for their success rates. MathWorks had examples available for the transfer learning of both nets. The basic strategy was to load the pretrained neural net and replace the last three layers that dealt with specific features. By keeping most of the neural network intact, the ability to classify images based on shape, color, and other general fundamental image features was maintained. However, by replacing the latter, more specific features layers, the neural network lost the classification capability for 1,000 pre-trained general categories and learned how to classify only the new training data. This is a result of the transfer learning process and is a compromise one must consider in exchange for reduced computational time required to train a neural network.

### 1. Preliminary Neural Network with Static Images

A preliminary neural network was developed to become familiar with the tools available within the MATLAB Neural Network toolbox and the transfer learning process. Random images from in and around the lab were collected to ensure the training data was different from other images the neural network was previously trained to identify. This determined if the neural network was versatile enough for place-based navigation applications. The new training data was input as a folder of photographs, separated into categories in subfolders within the main folder. This data was further separated into training and validation data. The loaded training images were fed into an augmented image datastore, which resized them to the network-required size of 227×227×3 and randomly flipped and translated them to avoid overfitting. There were several training options available, however, the primary ones of note were the *max epochs* and *mini batch size*. The *max epochs* dictated the number of full passes of the training algorithm over the entire training set. The *mini batch size* was a subset of the training set that evaluated the gradient of the loss function and updated weights accordingly. To increase accuracy and training success, these parameters were changed and evaluated until satisfactory results were achieved. The training process also displayed a graph that documented the success rate of the training data. In the graph the accuracy of the validation process over each epoch of the training process was shown on the top, and the data loss over each epoch was shown on the bottom. This graph was also useful because total training time and final validation accuracy were displayed when the process was complete. As a rule, more classification categories and training epochs caused training time to increase. An example from early trials using AlexNet with three classification categories is shown in Figure 15.



**Results**

| | |
|---|---|
| Validation accuracy: | 100.00% |
| Training finished: | Reached final iteration |

**Training Time**

| | |
|---|---|
| Start time: | 03-Jan-2019 14:19:32 |
| Elapsed time: | 31 sec |

**Training Cycle**

| | |
|---|---|
| Epoch: | 6 of 6 |
| Iteration: | 6 of 6 |
| Iterations per epoch: | 1 |
| Maximum iterations: | 6 |

**Validation**

| | |
|---|---|
| Frequency: | 3 iterations |

Figure 15. AlexNet Training Progress

Once the training completed, the network assessed the validation images and produced a label and probability score based on its classification.

The initial proof of concept trained the network with two categories: the control torsion plant and the pioneer robot. The training data included nine photos of the torsion plant and ten photos of the pioneer robot, taken from varying angles and distances. Because there were only two categories included in the training data, the network had no ability to identify items that were not in the two categories. When this trained network had to classify something new, it was limited to the torsion plant and robot categories. This demonstrated that in order to use this classification approach in navigation, the network required another category that served as a catch-all. The next test added a third category called "distractions," and the loaded images were random photos of animals, food, and landmarks. This test showed the CNN classified unique images with a high degree of accuracy and was suitable for further testing. An example of the classified images from the trained network is shown in Figure 16.

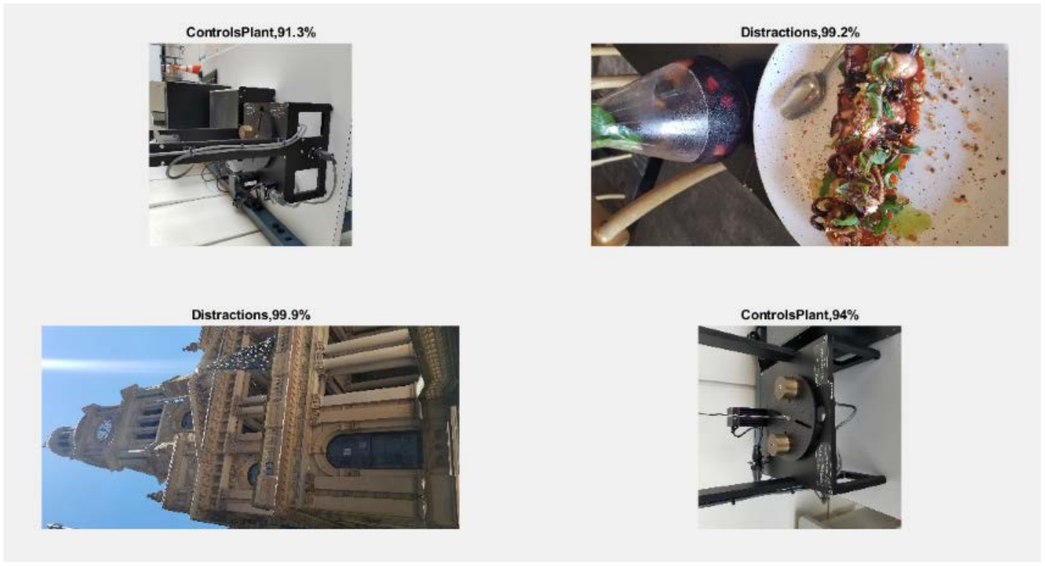


Figure 16. Classification Output for Trained AlexNet

### 2. Neural Network for Live Classification

After the initial testing with still images was completed, a new network was trained for live classification. An orange traffic cone and an orange bucket were used in the first iteration. The reason for choosing two similar objects was to determine how much weight the network placed on color. A third "Not Goal" category was also added for any item that was not the bucket or cone. Once the net was trained, it was implemented in a live stream from a webcam. It easily identified items that were "Not Goal" but struggled to correctly discriminate the cone from the bucket, as in Figure 17, which shows the orange cone being incorrectly identified as the orange bucket. This demonstrated that the training placed significant weight on color for classification and less weight on characteristics such as shape.

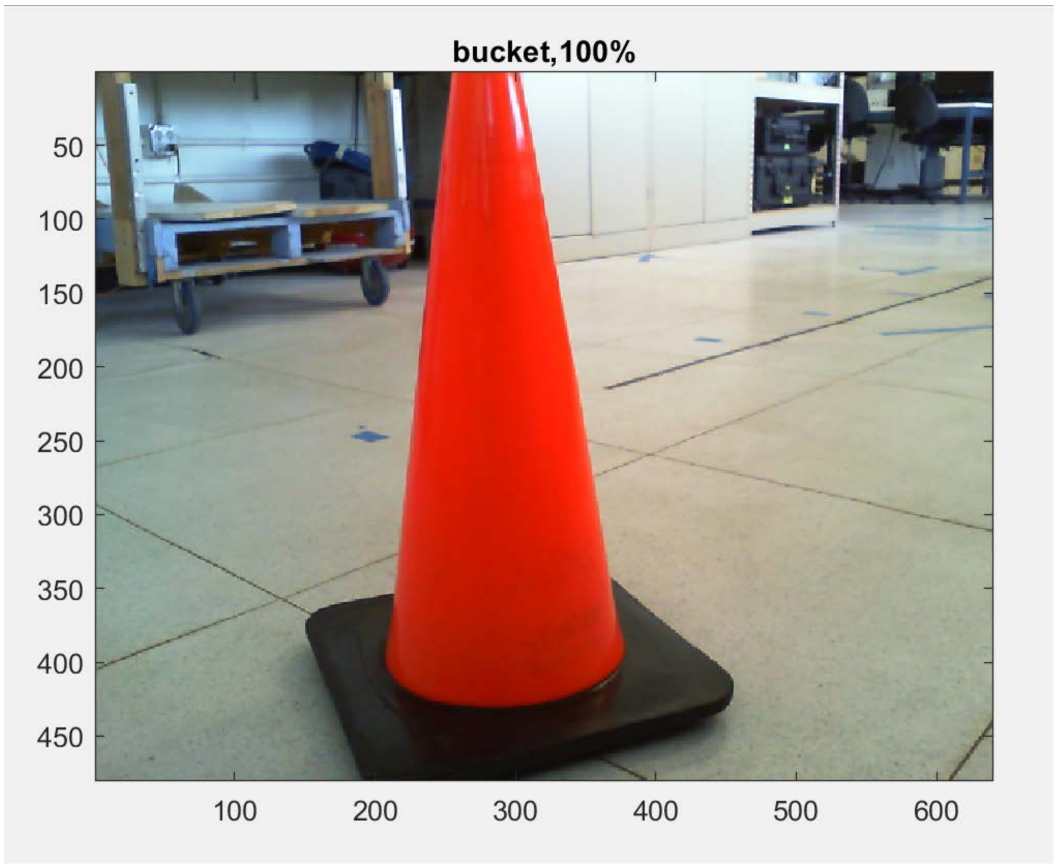


Figure 17. Classification Output for Cone Confused with Bucket

For the next step of testing, the cone pictures were removed from the training set. To simplify the training, two categories were created: “Goal” and “Not Goal,” in which the bucket photos were in the Goal folder and random lab photos in the Not Goal folder. The trained neural network was integrated into an existing MATLAB script that ran a wandering algorithm on the robot, as described in Chapter III. Inside of the main control loop, the neural network generated classification labels and associated probabilities for each image snapshot. Once the goal was identified with a mean probability of 0.8 or greater, taken over three loop iterations, a Boolean flag was set to break the loop and end the wandering program. The mean was included in the algorithm to limit the possibility of false positives breaking the loop. The image classification worked successfully live and ended the program upon identifying the bucket, as shown in Figure 18.

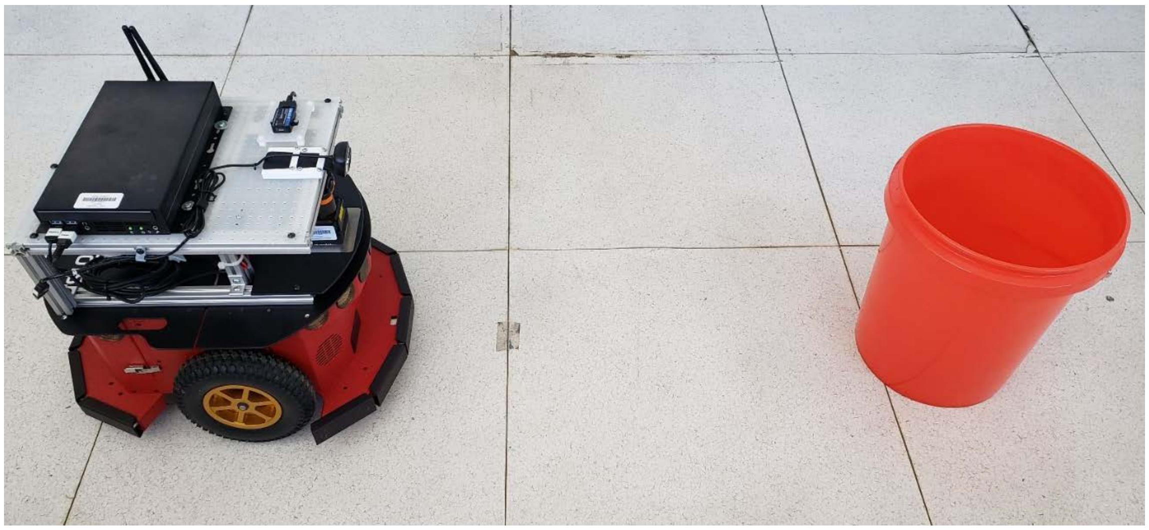


Figure 18. Robot Classifying Bucket

### 3. Neural Network for Place-Based Navigation

After determining the bucket testing was a success, the next step was to train a new neural network with the desired goal location. For the purpose of this research, the elevator doors located at the north end of the fifth floor in Spanagel Hall were selected for the goal location. This goal location was chosen because while the elevator doors did have distinguishing characteristics, they also had similarities to other doors and locations on the lab floor. The elevator doors also served as an example of a typical goal for a larger navigation problem under development. Using this goal location tests the robustness of the neural network in distinguishing between similar features. The classification was heavily color-dependent, which was one of the few differences between the elevators and other doors. To gather data, the robot was manually driven while taking photos and saving them to a file. This ensured the training photos had the same image source as the testing phase, and the photos were from the perspective of the robot. The output of the training process for the elevator data is shown in Figure 19. The validation portion of the training process showed the neural network was successfully able to distinguish between the elevator doors and other areas including lab doors.

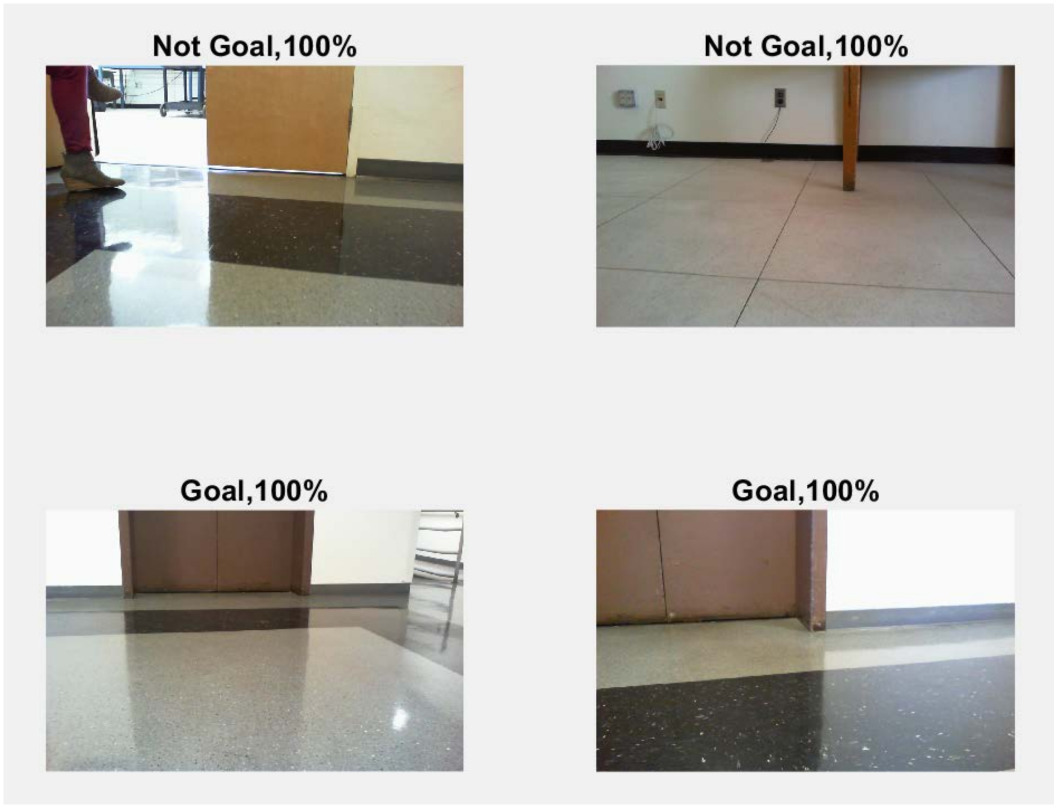


Figure 19. Classification Output for Trained GoogLeNet on Elevators

The trained network used still images when validating the training. However, when the new network was implemented live, the resulting "Goal" probabilities were low, ranging from 0.001 to 1%, when the robot was positioned directly in front of the elevator. The network training was examined to determine possible causes. One possible cause is that AlexNet is not as robust as GoogLeNet due to its differences in architecture, as discussed in Chapter III. Furthermore, the first iteration of training photos consisted of 30% elevator photos ("Goal") and 70% lab photos ("Not Goal"). This ratio created an imbalance in the training, consequently affecting the classification ability of the network. To correct this issue, more photos were taken of the elevator until the ratio was approximately 50/50. AlexNet was retrained, and the probabilities did not improve. Consequently, testing was switched to GoogLeNet. The GoogLeNet network was trained, and probabilities increased to approximately 20%. The algorithm, which integrated the image classification and navigation programming, was evaluated for possible time lapses or inconsistencies that could account for the error. After streamlining the algorithm by eliminating a *pause* command, success probabilities improved to approximately 50% at first detection. The detection probabilities gradually increased to 100% as the robot neared the elevator. The probability scores from GoogLeNet were a marked improvement over the scores of less than 1% that were being provided by AlexNet.

## B. INTEGRATION WITH ADVANCED NAVIGATION METHODS

Finding a navigation solution presented a unique problem because most autonomous navigation movement plans require the desired end point pre-programmed in the path planning. To meet the goals of this research, the robot needed to search an area efficiently, but only stop when it had identified its desired goal by vision.

### 1. Wandering Algorithm

The first phase of testing used a simple wandering algorithm. This algorithm was chosen as a starting point for testing because it was simple to write and implement. As described in Chapter III, The MATLAB algorithm calculated a repulsive force based on feedback from the robot sonar returns and combined with the forward force for a total force. This algorithm resulted in robot navigation that was random but had no memory of where it had previously been, which caused it to often drive in a circular path resulting in inefficient search patterns. The pose output from one of the wandering testing runs is shown in Figure 20, demonstrating the inefficient movement of the robot.

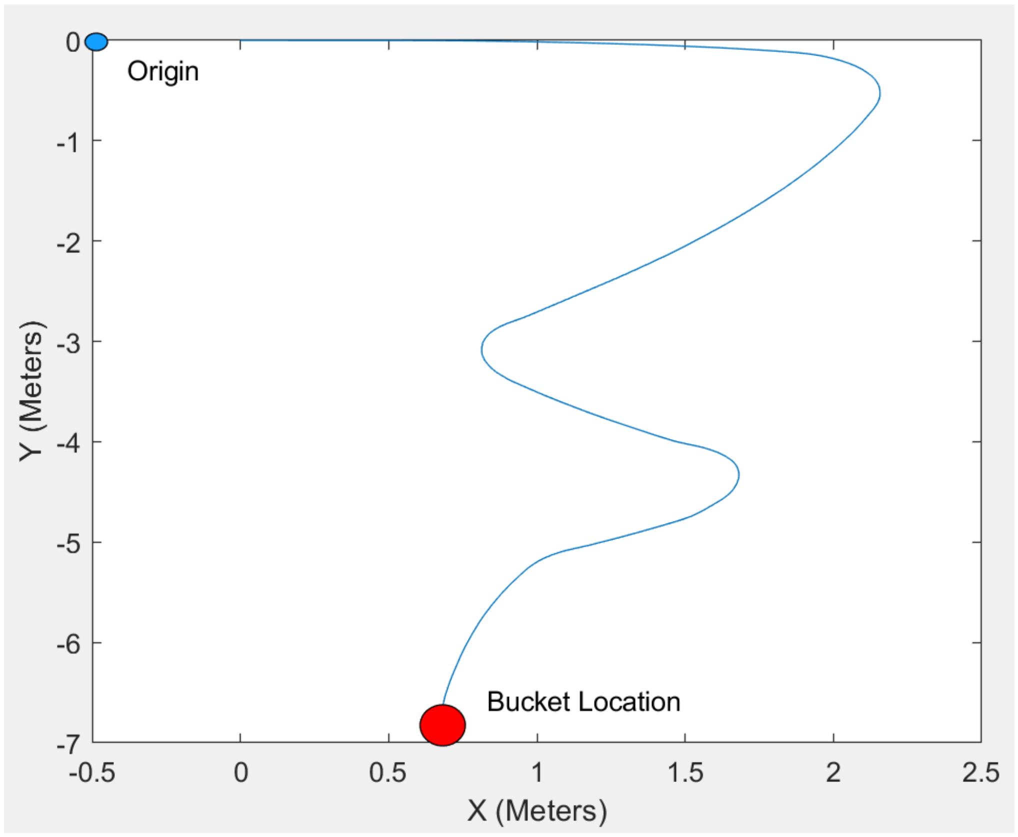


Figure 20. Wandering Algorithm Pose Graph

While this approach allowed for successful testing of the image classification program, a more efficient form of navigation was desirable.

### 2. ROS-Integrated Navigation

The next navigation approach investigated was the *frontier_exploration* package available in ROS. This package is an add-on to the *move_base* package used for autonomous navigation and allowed for the creation of an exploration area instead of requiring waypoints for the robot to navigate between. While this package had been successfully integrated and used with the TurtleBots mobile robot platform, use with the

P3-DX robot proved to be more complex than anticipated. Most of the TurtleBot files were pre-written and open source. Because the P3-DX is a discontinued platform, the support did not exist to implement the *frontier_exploration* package and writing ten layers of programming or more in Python was not feasible considering time constraints and skill required. For these reasons, the *frontier_exploration* approach for robot navigation was not viable.

### 3. Waypoint Following

The final navigation solution involved creating a pre-programmed search path. This method had multiple advantages. First, it allowed the robot to navigate efficiently while searching for its target location. Second, the pre-programmed search path involved waypoint navigation, but did not tell the robot where the goal endpoint was. This allowed the implementation of a simple algorithm for waypoint navigation that created more availability in computing power for the image processing and classification.

#### *a. Programming Manual Waypoints*

The first approach to creating the search path was to manually measure and program waypoints. Because the P3 commands and outputs are metric, the waypoint distances were measured in meters. Waypoints were measured to create a loop around the lab, and then programmed into MATLAB as a vector. To drive the robot, the MATLAB *pure pursuit* object was used. *Pure pursuit* uses an input of waypoints and the robot current pose and outputs linear and angular velocity commands to drive the robot towards the next waypoint. Initial testing quickly demonstrated that generating manual waypoints were not only inefficient, but also generated significant error due to localization inaccuracy and encoder drift. The desired waypoints and the actual path the robot took are shown in Figure 21.

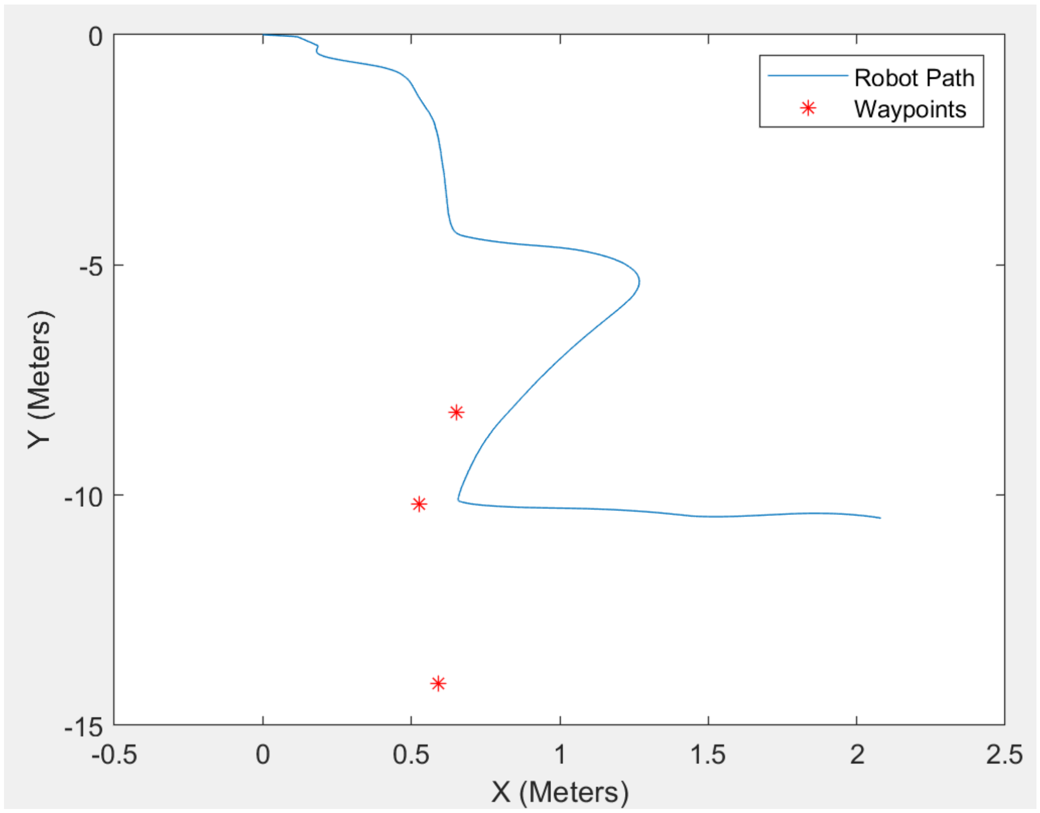


Figure 21. Manual Waypoints with Robot Path

Because the manual waypoints approach was not performing well, other approaches were investigated to create the waypoints.

### b. *Auto-Generated Waypoints from SLAM Algorithm*

The second approach attempted was to create a search path from MATLAB-generated waypoints using a SLAM algorithm. To create the search path, the LIDAR data first had to be collected. The robot was manually driven by keyboard while collecting and saving LIDAR data. The path started in the lab, followed the hallway past the elevators, looped around the elevator bay, and returned to the lab. Once the data was collected, it was imported into MATLAB’s Simultaneous Localization and Mapping (SLAM) algorithm, as described in Chapter III. This algorithm had two components: 1) it built a map based on LIDAR scans and 2) also returned the optimized poses for the robot’s path based on the LIDAR data. This approach was simple and quickly implemented but was susceptible to error due to the lack of real-time pose data from the robot. The resulting occupancy grid with robot poses is shown in Figure 22. In the occupancy grid, the map of the lab space in Spanagel Hall room 521 and the adjacent corridor including the elevator bay at the north end of the building is shown. Of note, the robot started and ended in the same location, so the figure also indicates a position error of approximately 2.5 meters.

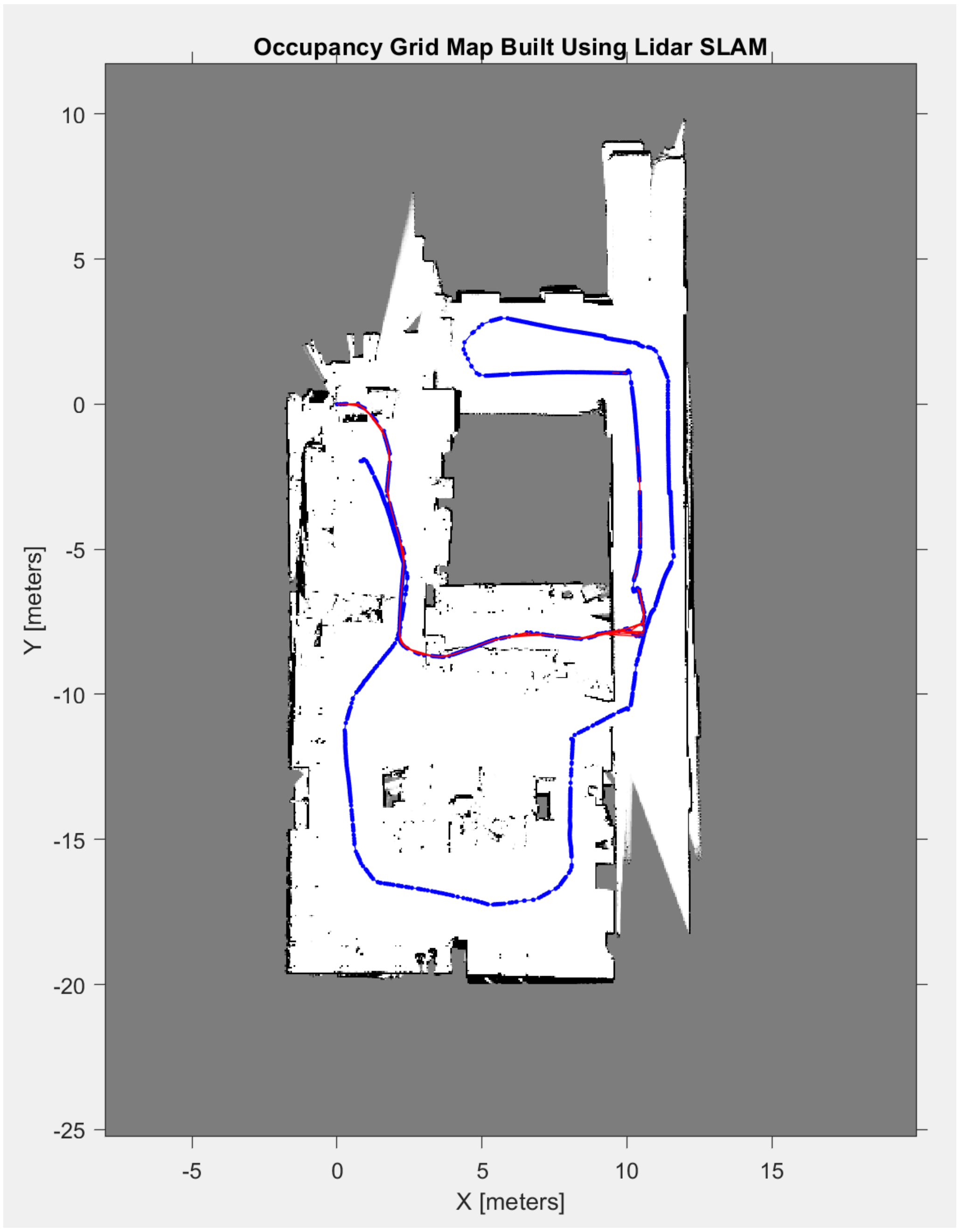


Figure 22. Occupancy Grid with Optimized Robot Poses

The next step was to drive the robot based on the optimized waypoints that resulted from the map that was developed of the lab space. The SLAM algorithm returned 2906 waypoints for the loop, which exceeded the requirement for an efficient solution. The waypoints were filtered for every 20th waypoint, resulting in 145 waypoints. As in the previous tests, the MATLAB *pure pursuit* object was used to drive the robot. Initial testing attempted to solely use this algorithm to give the robot drive commands, however, object avoidance was deemed a necessary addition in further testing. Using code snippets from previous thesis work, the robot used LIDAR to create a repellant force. The resulting linear and angular velocities from the repellant force were multiplied by respective gains and added to the pure pursuit velocity commands. This approach created a smooth path-following motion that also successfully navigated around obstacles. Due to the error introduced in the SLAM process, as well as encoder error and wheel slippage in the robot, localization suffered the longer the robot traveled, and planning a reliable path became more difficult. Consequently, the path was shortened to a single loop from the lab, to the elevator, and back into the lab through the doors near the elevator. The updated loop is shown in Figure 23. The waypoint metrics are in meters, however, the error introduced into the system skews the map of the path as time increases. For example, the origin and end point appear to be approximately eight meters away from each other but were only one to two meters apart.

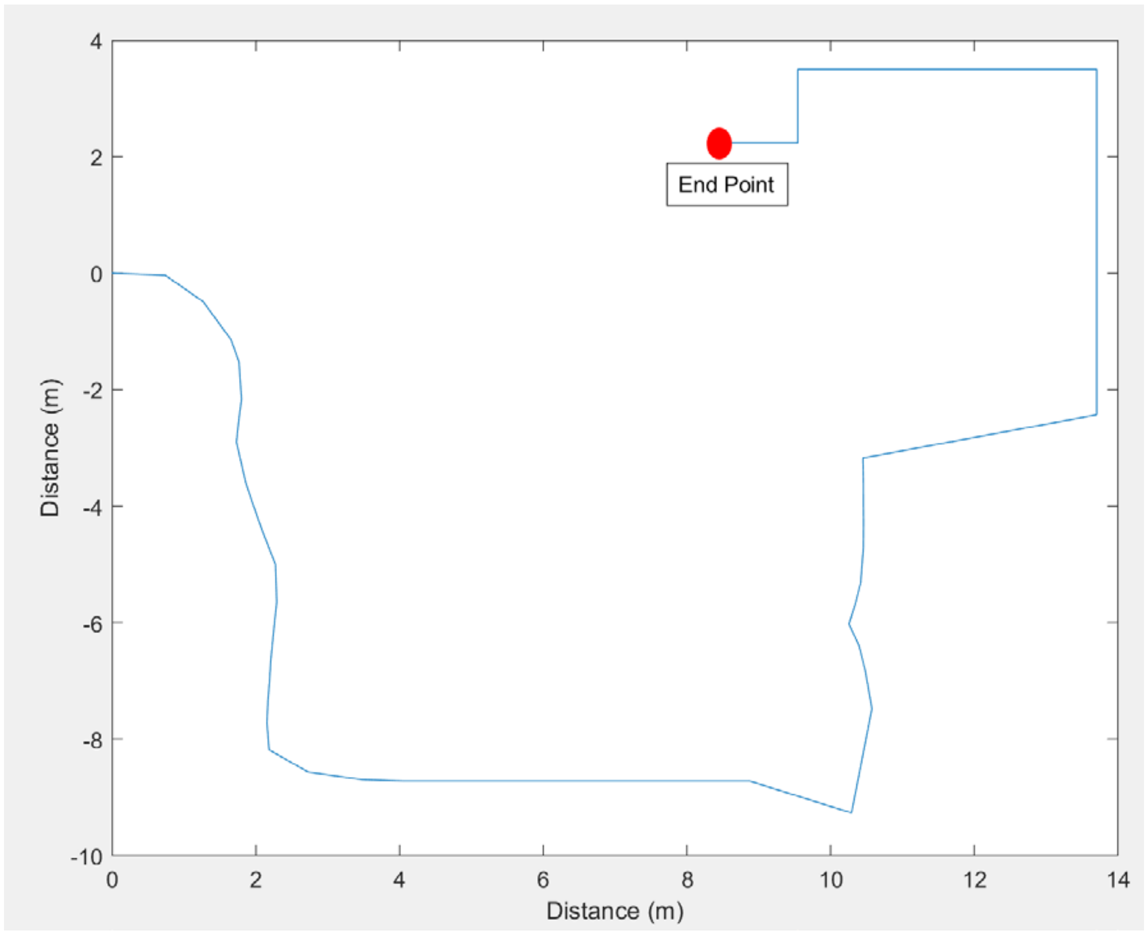


Figure 23. Robot Waypoint Loop

### *c. Integration of SLAM Waypoints and Image Classification*

After the robot ran the full loop successfully, the image classification code was added to the algorithm. Initially, the robot was programmed to stop at the first identification of the elevators above a 20% threshold. The robot stopped with a 46.8% probability as soon as the elevators entered the camera field of view, at approximately 12 feet from the elevator. The output image with probability is shown in Figure 24, and the real-time navigation overlaid with the given waypoints is shown in Figure 25. While the waypoints were a vector of goal destinations input into the *pure pursuit* tool, the error discussed earlier skewed the real-time navigation path. When the robot traveled along the hallway, the skew made the waypoints appear inside the wall. The LIDAR obstacle detection system allowed the robot to avoid the walls and other obstacles, which accounts for the diagonal path away from the given waypoints. The *pure pursuit* object allowed the robot to stray from the waypoints because it was programmed with a two-meter look ahead distance. The real-time navigation also stops before reaching the final waypoint because the place-based image classification identified the elevator prior to the loop end.

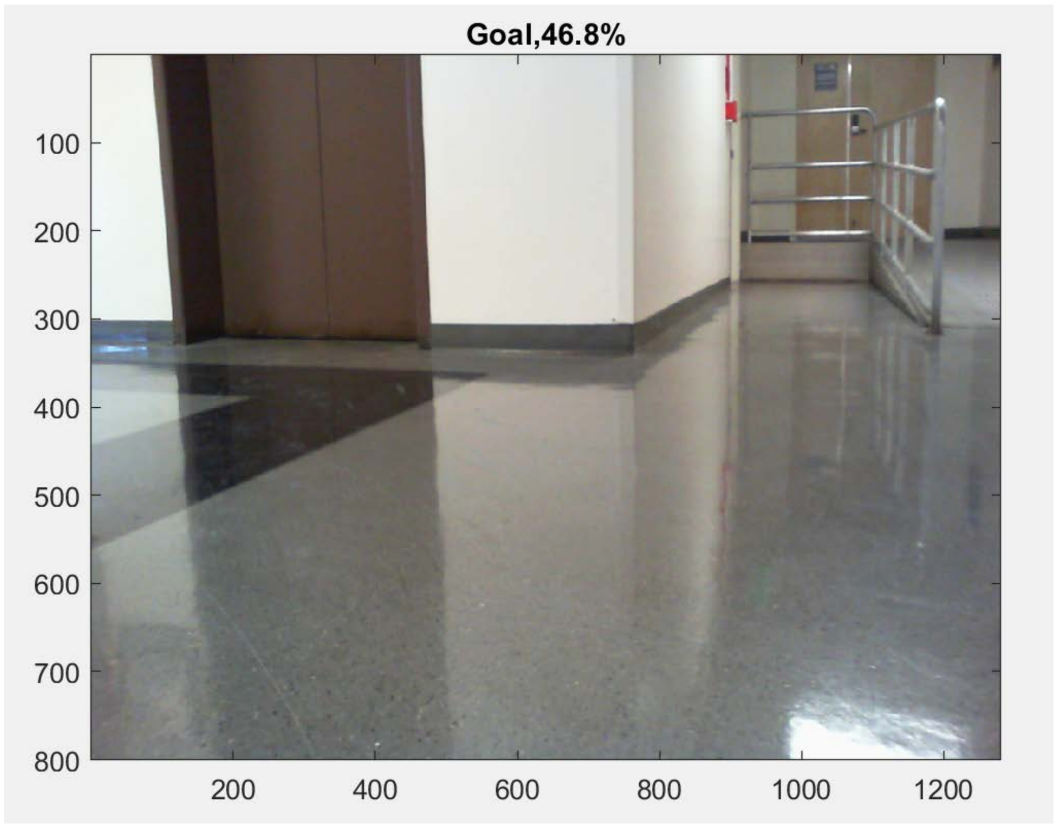


Figure 24. Classification Image for Elevator

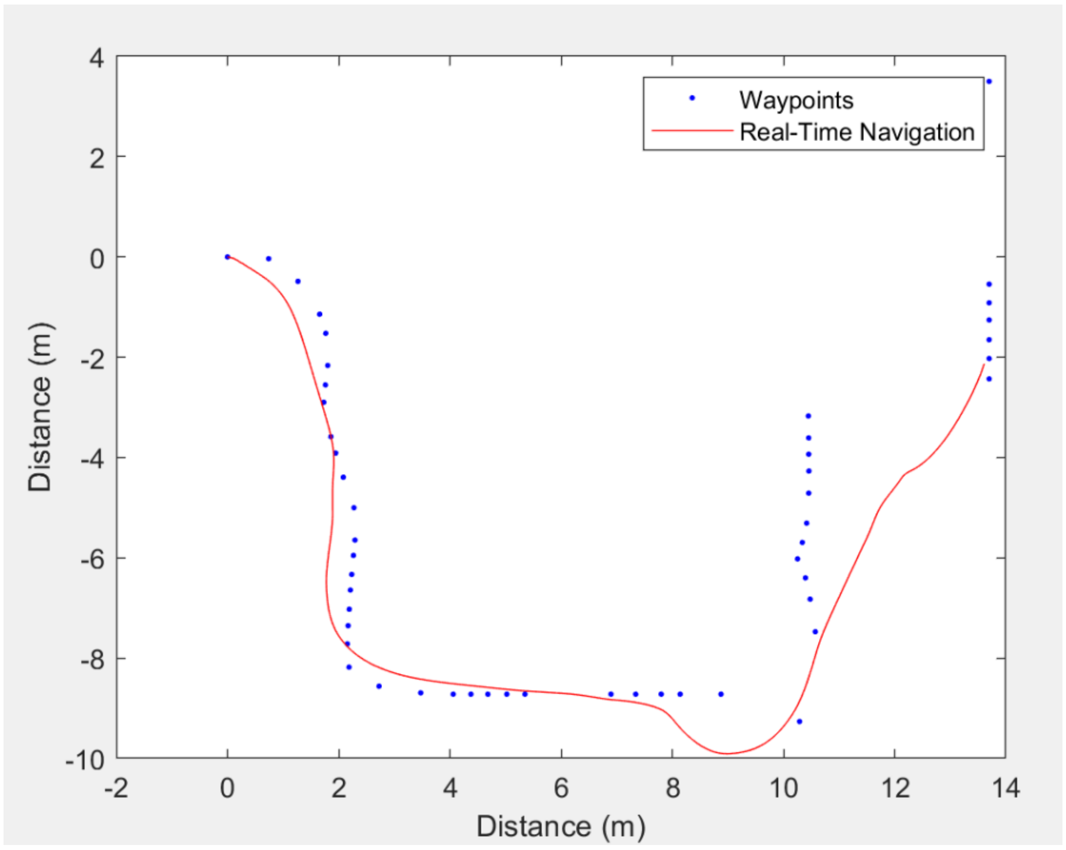


Figure 25. Waypoints and Real-Time Robot Path

### d. *Evaluation of Probability Scores*

Next, the goal probabilities were evaluated for the entire waypoint set to determine when the maximum classification level was reached. This experiment was conducted in two variations. During the first iteration, the elevators were blocked with a large piece of cardboard. This was to ensure the classification worked properly and prove it was solely dependent on image, not location in the loop. As expected, the image classification loop did not identify the elevator with the cardboard in place. The second iteration removed the cardboard from in front of the elevators. The portion of the algorithm that stopped the robot upon location identification was temporarily taken out in order to assess probabilities through the entire robot loop. As shown in Figure 26, the classification process first begins to identify the elevator around 240 loop cycles. As the robot approaches the elevator, the probabilities increase until it reaches a threshold, and the robot is too close to the elevator for reliable classification. As the robot approached the elevator, the resulting camera snapshots were too close to the elevator to identify more than color and some lines. The robot then turned and moved away from the elevator, accounting for the decrease back to zero probability.

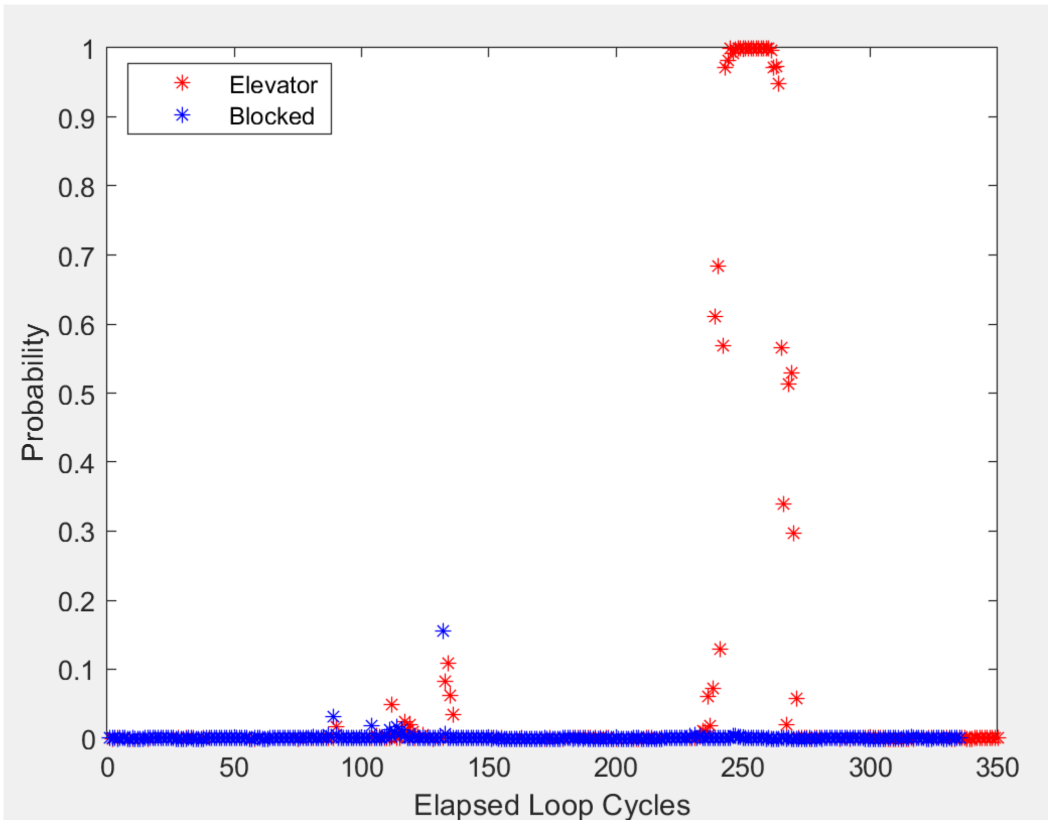


Figure 26. Goal Probabilities over Time

Of note, in both testing rounds there is a cluster of low-probability false positives around 125-140 loop cycles. The reason for this was a set of double doors the robot drove past. This demonstrates that the image classification was susceptible to being fooled by similar objects, however, it was robust enough to keep the false positive probabilities low.

## C. SUMMARY

This research tested image classification using a convolutional neural network and integrated it into an autonomous navigation solution. To use the CNN on a specific data set, it went through the transfer learning process. Multiple objects were used to test classification in order to test the robustness of the CNN, and it was ultimately trained to successfully identify elevator bay doors.

For the autonomous navigation solution, three main approaches were tested: a wandering algorithm, ROS-integrated navigation, and waypoint following. Of the three approaches, the waypoints generated from the MATLAB SLAM tool provided the most efficient navigation solution. The image classification was integrated into the waypoint navigation and successfully identified the elevators as designed.

# V. SUMMARY AND CONCLUSIONS

Applications involving AI have been the focus of academic research and commercial development to a great extent in recent years. This thesis research explored AI as a potential solution for the navigation and control of an autonomous land-based mobile robot. It was an integral part of a larger program to develop an autonomous mobile robot capable of navigating throughout the NPS campus both indoors and outdoors. Within this research, experimental work was conducted with the P3-DX mobile robot extensively utilizing ROS and MATLAB tools for both image classification and autonomous navigation. This work involved sensor integration with a webcam and LIDAR sensor as well. The combined hardware, software, and sensor suite enabled the P3-DX to navigate autonomously while using place-based image classification.

## A. ASSESSMENT OF GOALS

The first goal of this research was to determine a method for image classification that was able to be used in place-based navigation and implement that method. A convolutional neural network was trained to classify images in both still and live environments. The classification worked on common locations without requiring additional information. Since the classification was reliant on image processing, it is susceptible to changes in lighting and environment. The neural network training could be improved with a larger training database to include various lighting configurations.

The second goal was for a robot to successfully navigate to a desired location autonomously and positively identify the goal based solely on image classification. The robot autonomously navigated a pre-planned path using LIDAR for live object avoidance. When the goal location entered the camera field-of-view, the image classification correctly identified the goal location and stopped the robot.

## B. AREAS OF FUTURE WORK

### 1. Indoor Localization

A significant issue with the autonomous navigation solution is that the encoder error and wheel slippage create localization error. In order to use this system in more GPS-denied environments, the localization should be improved. This work may use localization beacons, secondary encoders, or other localization methods to improve the robot awareness of current location.

### 2. Increase Training Data

The neural network is capable of classifying thousands of images. The training data library should be expanded to include both indoor and outdoor landmarks from the NPS campus. This will allow the user to define a location based on a specific descriptor rather than "goal" and change the desired location with ease. This work will serve the end goal of the overall project of a robot autonomously navigating the entire NPS campus.

### 3. Integration of Places into Navigation Plan

The image classification algorithm could be integrated into a waypoint follower, such as the work done by Matthew Audette, to improve localization in areas on GPS vulnerability or difficult terrain [1]. When locations or obstacles are classified, they could be added into the map, generating a more accurate navigation picture. That updated map could then be used to update the overall navigation plan so the robot is able to move more efficiently through the desired space.

# APPENDIX A.  CNN TRAINING SCRIPT

## A. ALEXNET

```
% AlexNet Training Script
% Ashleigh Magee

% This script trains AlexNet to identify specific images in the
data
% folder. We can control training vs test images, how many
testing
% iterations, etc. This code also saves the trained net so it can
be loaded
% and used in testtrainedlive.m

net=alexnet;  %open alexnet
% analyzeNetwork(net)      look at net specifications
inputSize=net.Layers(1).InputSize;  % image inputs required to be
227x227x3
layersTransfer=net.Layers(1:end-3); %extracts all layers except
last 3

imds=imageDatastore('C:\Users\sweet\OneDrive\Documents\MATLAB\
Thesis\TrainingPhotos',...
    'IncludeSubfolders',true,'LabelSource','foldernames');
% labels are created from file folder names. To have multiple
class
% categories, you have to create individual subfolders within the
main
% folder with the names you want for your classes
[imdsTrain,imdsValidation]=splitEachLabel(imds,0.6,'randomized');
numClasses=numel(categories(imdsTrain.Labels))
% replace the last 3 layers in order to fine-tune for new
classification
layers = [
    layersTransfer

fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLe
arnRateFactor',20)
    softmaxLayer
    classificationLayer];
pixelRange=[-30 30];
imageAugmenter = imageDataAugmenter( ...    %randomly flip and
shift images
    'RandXReflection',true, ...             % to avoid
overfitting
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
```

```
augimdsTrain=augmentedImageDatastore(inputSize,imdsTrain, ...
    'DataAugmentation',imageAugmenter);
augimdsValidation=augmentedImageDatastore(inputSize,imdsValidatio
n,'OutputSizeMode','resize');
% investigate how it is resizing image
options=trainingOptions('sgdm', ...
    'MiniBatchSize',10, ...
    'MaxEpochs',10, ...
    'Shuffle','every-epoch', ...
    'InitialLearnRate',1e-4, ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',3, ...
    'Verbose',false, ...
    'Plots','training-progress');
netTransfer=trainNetwork(augimdsTrain,layers,options);

[YPred,probs]=classify(netTransfer,augimdsValidation);
accuracy=mean(YPred==imdsValidation.Labels)
idx=randperm(numel(imdsValidation.Files),4);
figure
% Test the training success with validation images
for i=1:4
    subplot(2,2,i)
    I=readimage(imdsValidation,idx(i));
    imshow(I)
    label=YPred(idx(i));
    title(string(label) + "," +
num2str(100*max(probs(idx(i),:)),3) + "%");
end

elev_net2=netTransfer;
save elev_net2;
```

## B. GOOGLENET

```
% GoogLeNet Training Script
% Ashleigh Magee

% This script trains GoogleNet to identify specific images in the
data
% folder. We can control training vs test images, how many
testing
% iterations, etc. This code also saves the trained net so it can
be loaded
% and used in testtrainedlive.m

imds=imageDatastore('C:\Users\sweet\OneDrive\Documents\MATLAB\
Thesis\TrainingPhotos',...
    'IncludeSubfolders',true,'LabelSource','foldernames');
% labels are created from file folder names. To have multiple
class
% categories, you have to create individual subfolders within the
main
% folder with the names you want for your classes
[imdsTrain,imdsValidation]=splitEachLabel(imds,0.7,'randomized');
net=googlenet;  %open googlenet
% analyzeNetwork(net)      look at net specifications
inputSize=net.Layers(1).InputSize;  % image inputs required to be
227x227x3
if isa(net,'SeriesNetwork')     %Makes a layer graph so we can
extract/train
  lgraph = layerGraph(net.Layers);  %necessary layers
else
  lgraph = layerGraph(net);
end

% Find layers that can be trained and replaced
[learnableLayer,classLayer] = findLayersToReplace(lgraph);
%[learnableLayer,classLayer]
numClasses = numel(categories(imdsTrain.Labels));

% Replace last layer with learnable weights. This if/else loops
allows
% different kinds of NN to be used. Most use a Fully Connected
Layer for
% the last layer, however, some use a Convolutional 2D Layer
if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
```

```
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
end
% Replace the layer in the layer graph
lgraph =
replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);

newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);

% Show new layers are connected
% figure('Units','normalized','Position',[0.3 0.3 0.4 0.4]);
% plot(lgraph)
% ylim([0,10])

% Freeze layers in order to speed up processing time and prevent
% overfitting to new data set
layers = lgraph.Layers;
connections = lgraph.Connections;

layers(1:10) = freezeWeights(layers(1:10));
lgraph = createLgraphUsingConnections(layers,connections);

% Train net - use image augmenter to resize and flip/translate
the images.
% The data augmenter prevents overfitting of data to the training
images
pixelRange = [-30 30];
scaleRange = [0.9 1.1];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange, ...
    'RandXScale',scaleRange, ...
    'RandYScale',scaleRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain,
...
    'DataAugmentation',imageAugmenter);

%Automatically resize images without performing additional
augmentation
augimdsValidation =
augmentedImageDatastore(inputSize(1:2),imdsValidation);

% Training parameters
miniBatchSize = 10;
valFrequency = floor(augimdsTrain.NumObservations/miniBatchSize);
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',10, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',valFrequency, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(augimdsTrain,lgraph,options);

%Classify test images
[YPred,probs]=classify(net,augimdsValidation);
accuracy=mean(YPred==imdsValidation.Labels)
idx=randperm(numel(imdsValidation.Files),4);
figure
% Check training with validation images
for i=1:4
    subplot(2,2,i)
    I=readimage(imdsValidation,idx(i));
    imshow(I)
    label=YPred(idx(i));
    title(string(label) + "," +
num2str(100*max(probs(idx(i),:)),3) + "%");
end

goog_net2=net;
save goog_net2;
%net=goog_net;
```

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX B. TEST TRAINED CNN

```
% Live Test of Trained CNN
% Ashleigh Magee

% This script tests the trained neural net's ability to identify
the items
% it has been trained on. Change "net" value in order to change
NN being
% used. If you run in repeated successions, clear the variables
or comment
% out the net="net" and camera=webcam lines. You cannot
initialize those
% when a connection has already been made

load('elev_net2.mat');
net=elev_net2;
clear camera;
camera=webcam('Microsoft® LifeCam HD-3000');
inputSize=net.Layers(1).InputSize(1:2);
h = figure;
TT=[];
while ishandle(h)
    im = snapshot(camera);
    image(im)
    im=augmentedImageDatastore(inputSize(1:2),im);
    [label,score] = classify(net,im);
    title(string(label) + "," + num2str(100*max(score(1,2)),3) +
"%")
    drawnow
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX C.  WANDERING PROGRAM

```
% Edited version of example_Wander_ROS.m by Dr. James Calusdian
% Edited by Ashleigh Magee to include code to utilize trained CNN
for
% image classification and flag the system when the target is
identified

% example_Wander_ROS.m
% A script to make the robot wander around in the room.  Gently
tap one of
% the bumpers to make the robot stop the wander program.
Computing a
% repulsive force based on sonar measurements, then filtering for
smoother
% operation.
%
% This script uss the ROS interface and the Robotics Systems
Toolbox
%
% To use this script:
%    1. In a terminal window, run the launch file
%            roslaunch  rosaria   p3atlaunch.launch
%    2.  In the Matlab command window, type
%            rosinit
%    3.  Next, run this script.
%
% The robot should now wander around in the room using its sonar
to avoid any
% obstacles it encounters.
%
% Copyright Naval Postgraduate School, 2015

format compact   % for command window formatting

% ROS business
global LocalOdom

% Create ROS publishers, subscribers, and service client
localOdomSub = rossubscriber('/RosAria_Node/
pose',@p3atLocalOdomCallback)
cmdPub = rospublisher('/RosAria_Node/cmd_vel','geometry_msgs/
Twist')
sonarSub = rossubscriber('/RosAria_Node/sonar')
bumperSub = rossubscriber('RosAria_Node/bumper_state')
batterySub = rossubscriber('RosAria_Node/battery_voltage')
pause(3)
cmdMsg = rosmessage(cmdPub)  % Create empty messages for
publication
```

```
fwdMAXVEL = .200;        % max robot velocity
loopCounter =0;         % used in while-loop to stop program
MAX_LOOP_COUNT = 300;
latencyValues = NaN*ones(1,MAX_LOOP_COUNT);
A = 0.45;                % filter constant for low-pass filter
A*new + (1-A)*old
resultFiltered_prev = 0;  % for low-pass filter
gainTransVel = 0.75;       % sensitivity for translational
velocity
gainRotVel = 0.05;         % sensitivity for rotational velocity,
3.5
Ffwd = [2.000;0];        % forward driving force
Ftotal = [0;0];         % total force
% create empty arrays for the X and Y sonar data
sonarX = NaN(1,16);
sonarY = NaN(1,16);

% repulsive force parameters
dC = 3.5;    % threshold distance. Compute F_rep for di<dC
ETA = 1e6;    % scales the repulsive force

% first connect to the robot
%%p3_connector('/dev/ttyS0')
pause(5)

% define minimum clearance
MIN_CLEARANCE = 0.375;   % 250 mm
rangeClearance = true;  % for use in while-loop
minRange = 3*MIN_CLEARANCE;  % initialize to something

% define the sonar angles on the robot, see manual pp 13.
%gamma = [90 50 30 10 -10 -30 -50 -90 -90 -130 -150 -170 170 150
130 90]*pi/180;

% check bumpers
bumperMsg = receive(bumperSub);
bumpersClear = isempty(find(or(bumperMsg.FrontBumpers,
bumperMsg.RearBumpers)));
if(~bumpersClear)
    disp('Bumpers not clear!!')
    disp('Exitng program')
    return
end

% Add net & image classification loop
% Initialize the system for image loop
load('testnet_cone.mat');
net=testnet_cone;
camera=webcam;
```

```
inputSize=net.Layers(1).InputSize(1:2);
h = figure;
tgtfound=false;
tgt=imdsValidation.Labels(10);  %sets the target to the desired
label name
probavg=0;
pvec=[];
i=0;
counter=0;

while(bumpersClear && rangeClearance && ~tgtfound)
    loopCounter = loopCounter+1;

    % get battery voltage from ROS
    batteryVoltage = receive(batterySub);


      % get sonar data and plot point cloud
    sonarData = receive(sonarSub);
    for ix = 1:16
        sonarX(ix) = sonarData.Points(ix).X;
        sonarY(ix) = sonarData.Points(ix).Y;
    end
    sonarRange = sqrt(sonarX.^2 + sonarY.^2);

     % compute a repulsive force based on the sonar measurements
     Frep_r = [0;0];   % robot coordinates
     for ix = 1:16
         di = sonarRange(ix);
         if di< dC
            ni = [sonarX(ix) ; sonarY(ix)];
            Frep_r = -ETA * (1/di - 1/dC)*(ni./di) + Frep_r;
         end

     end

     %myString = sprintf('Repulsive force %12.1f
%12.1f',Frep_r(1), Frep_r(2));
     %disp(myString);

     % total force
     Ftotal = Ffwd + Frep_r;

     % next low-pass filter the result to smooth the result
     resultFiltered = A * Ftotal + (1-A)*resultFiltered_prev;
     resultFiltered_prev = resultFiltered;
     %myString = sprintf('Filtered result %12.1f
%12.1f',resultFiltered(1), resultFiltered(2));
     %disp(myString);
     % use the last result to coordinate robot motion
     fwdVel = gainTransVel * resultFiltered(1);
     if fwdVel<0, fwdVel=0.050; end
     if fwdVel>fwdMAXVEL, fwdVel = fwdMAXVEL; end
     rotVel = gainRotVel * atan2(resultFiltered(2), 
resultFiltered(1));

     %p3_setTransVel(fwdVel);
     %p3_setRotVel(rotVel);
    cmdMsg.Linear.X = fwdVel ;    % Drive fowrward/backwards
    cmdMsg.Angular.Z = rotVel ;   % Turn (CCW = +)
    send(cmdPub,cmdMsg);        % Send velocity command to p3

    % Display formatted info on screen
    clc
    fprintf('\n\nForward and Rotational velocity, %6.2f m/s 
%6.2f deg/sec\n', fwdVel,rotVel)
    fprintf('Loop counter %d\n', loopCounter)
    fprintf('Battery voltage = %5.2f volts\
n\n',batteryVoltage.Data)
    fprintf('Sonar ranges (meters)...\n')
    fprintf('%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f\n', 
sonarRange)

    % make sure we are not too close to an obstacle.  If we are, 
then set
    % the "rangeClearance" to False so that we break out of the 
while loop
    index = find(sonarRange < MIN_CLEARANCE);
    if(~isempty(index))
        rangeClearance = false;
        myString = 'Not enough clearance.  Disconnecting from 
robot.';
        myString2 =sprintf('Sonar number: %u ', index(1)-1);
        disp(myString);
        disp(myString2);
    end

    % check bumpers
    bumperMsg = receive(bumperSub);
    bumpersClear = isempty(find(or(bumperMsg.FrontBumpers, 
bumperMsg.RearBumpers)));
    if(~bumpersClear)
        disp('Bumpers not clear!!')
        disp('Exiting program')
        return
    end
% image processing loop
    if counter < 3
        pause(0.01);
        counter=counter+1;
    else
        im = snapshot(camera);
        image(im)
        im = imresize(im,inputSize);
        [label,score] = classify(net,im);
        title(string(label) + "," +
num2str(100*max(score(1,2)),3) + "%")
        drawnow
        if label==tgt
            prob=max(score(1,2));
            pvec=[pvec,prob];   %create probability vector
            i=i+1;
            if i>2
                probavg=mean(pvec);
                i=0;
            elseif i>15 %clears probability vector so it doesn't
get bogged
                pvec=[];    %down with bad probability
            end
        end

        if (label==tgt && probavg>=0.8)
            tgtfound=true;
            disp('Found Desired Target, Ending Program')
        end
        counter=0;
    end
end
    % wait a little bit for robot to catch up with Matlab
    pause(.1);  % MobileSim, was 1 sec
end
% stop and disconect from the robot
```

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX D.  SLAM MAP BUILDING

```
% Convert LIDAR data to SLAM map
% Ashleigh Magee

% Load LIDAR data
load('laserScanLab2.mat');
scans=laserScanMessageArray;

% SLAM algorithm parameters
maxLidarRange = 8;
mapResolution = 20;
slamAlg = robotics.LidarSLAM(mapResolution, maxLidarRange);
slamAlg.LoopClosureThreshold = 210;
slamAlg.LoopClosureSearchRadius = 8;

% %% check 1st ten scans
% for i=1:10
%     [isScanAccepted, loopClosureInfo, optimizationInfo] =
addScan(slamAlg, scans{i});
% %     if isScanAccepted
% %         fprintf('Added scan %d \n', i);
% %     end
% end
%
% figure;
% show(slamAlg);
% title({'Map of the Environment','Pose Graph for Initial 10
Scans'});

%% Loop closure and optimization
firstTimeLCDetected = false;

figure;
for i=1:length(scans)
    [isScanAccepted, loopClosureInfo, optimizationInfo] =
addScan(slamAlg, scans{i});
    if ~isScanAccepted
        continue;
    end
    % visualize the first detected loop closure, if you want to
see the
    % complete map building process, remove the if condition
below
    i
end
figure
show(slamAlg);
```

```
title({'Final Built Map of the Environment', 'Trajectory of the
Robot'});

%% Build Occupancy Grid Map
[scans, optimizedPoses]  = scansAndPoses(slamAlg);
map = buildMap(scans, optimizedPoses, mapResolution,
maxLidarRange);

figure;
show(map);
hold on
show(slamAlg.PoseGraph, 'IDs', 'off');
hold off
title('Occupancy Grid Map Built Using Lidar SLAM');
```

# APPENDIX E. FINAL SEARCH PATH WITH IMAGE CLASSIFICATION

```
% Search Path w/ Image Classification
% Ashleigh Magee

% This script uses a series of waypoints generated by SLAM
optimized pose
% data as a search path. The GoogLeNet classification loop runs
% simultaneously. The loop includes a boolean flag which will end
the
% program upon identification of the goal location.

%% Setup and parameter initialization

% Create global variables for use in communicating with ROS
system
global Pose
global Laser

% Create ROS publishers and subscribers
poseSub = rossubscriber('/RosAria_Node/pose',@p3atPoseCallback);
cmdPub = rospublisher('/RosAria_Node/cmd_vel','geometry_msgs/
Twist');
bumperSub = rossubscriber('RosAria_Node/bumper_state');
laserSub = rossubscriber('/Laser_Scan0',@p3atLaserCallback);

pause(2); % Wait to ensure publisher is registered
% Create an empty Twist message for publication
cmdMsg = rosmessage(cmdPub);

% Variable declaration
eta = 0.002;             % repulsive force gain
rho0 = 2;            % offset from obstacle to ignore repulsive
term
k1=0.1;              % linear repulsive force gain
k2=0.7;

%% Check Bumpers
% check bumpers
bumperMsg = receive(bumperSub);
bumpersClear = isempty(find(or(bumperMsg.FrontBumpers,
bumperMsg.RearBumpers)));
if(~bumpersClear)
    disp('Bumpers not clear!!')
    disp('Exitng program')
    return
end
```

```
% Add net & image classification loop
% Initialize the system for image loop
load('elev_net.mat');
net=elev_net;
camera=webcam('Microsoft');
inputSize=net.Layers(1).InputSize(1:2);
h = figure;
tgtfound=false;
tgt=imdsValidation.Labels(10);  %sets the target to the desired
label name
probavg=0;
pvec=[];
i=0;
counter=0;
fprintf('Image System Initialization Complete\n');

%% Create waypoint follower
load('waypoints_short2.mat');   % From optimized poses
pp = robotics.PurePursuit('DesiredLinearVelocity',
0.2,'LookaheadDistance',2);

%Assign waypoints.
pp.Waypoints = waypoints;
vv=[];
ww=[];
pose_x=[];
pose_y=[];
while(bumpersClear && ~tgtfound)
    % get the laser ranges
    laser_range = Laser.Ranges;
    % angular resolution vector
    laser_angle =
(Laser.AngleMin:Laser.AngleIncrement:Laser.AngleMax)';

    % get X, Y and Theta
    pose = Pose.Pose.Pose;
    quat = pose.Orientation;
    angles = quat2eul([quat.W quat.X quat.Y quat.Z]);
    yaw = angles(1);
    x = pose.Position.X;
    y = pose.Position.Y;
    th = yaw;

    pose_x=[pose_x x];
    pose_y=[pose_y y];

    %Compute control commands using the pp object with the
initial pose [x y theta] given as the input.
    [v,w] = pp([x y th]);
    vv=[vv v];
    ww=[ww w];

    % Lidar Portion of Force
    [Flas] = LidarForce(laser_range, laser_angle, rho0, eta);
    MinLaserRange = length(laser_range(laser_range < 0.5));

    % Calculate total force
    LinVel=v+k1*Flas(1);
    AngVel=w+k2*Flas(2);

    cmdMsg.Linear.X = LinVel;
    cmdMsg.Angular.Z = AngVel;
    send(cmdPub,cmdMsg);

     % check bumpers
    bumperMsg = receive(bumperSub);
    bumpersClear = isempty(find(or(bumperMsg.FrontBumpers,
bumperMsg.RearBumpers)));
    if(~bumpersClear)
        disp('Bumpers not clear!!')
        disp('Exiting program')
        return
    end

    % Image classification loop
    tic;
    if counter < 3
        %pause(0.01);
        counter=counter+1;
    else
        im = snapshot(camera);
        image(im)
        im = imresize(im,inputSize);
        [label,score] = classify(net,im);
        title(string(label) + "," +
num2str(100*max(score(1,2)),3) + "%")
        drawnow
        if label==tgt
            prob=max(score(1,2));
            pvec=[pvec,prob];   %create probability vector
            i=i+1;
            if i>2
                probavg=mean(pvec);
                i=0;
            elseif i>8
                pvec=[];
                probavg=0;
%              %clears probability vector so it doesn't get
bogged
```

```
%                 pvec=[];    %down with bad probability
            end
        end

        if (label==tgt && probavg>=0.2)
            tgtfound=true;
            disp('Found Desired Target, Ending Program')
        end
        counter=0;
    end
end
```

# LIST OF REFERENCES

[1] G. Narula, "Everyday examples of artificial intelligence and machine learning," Emerj, August 5, 2019. [Online]. Available: https://emerj.com/ai-sector-overviews/everyday-examples-of-ai/

[2] Chief of Naval Operations, "A design for maintaining maritime superiority," Washington, DC, USA, 2018. [Online]. Available: https://www.navy.mil/navydata/people/cno/Richardson/Resource/Design_2.0.pdf

[3] Penn State, "5.3 GPS error sources | GEOG 160: Mapping our changing world." Accessed September 4, 2019. [Online]. Available: https://www.e-education.psu.edu/geog160/node/1924

[4] C. S. Hargadine, "Mobile robot navigation and obstacle avoidance in unstructured outdoor environments," M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2017. [Online]. Available: https://calhoun.nps.edu/handle/10945/56937

[5] M. R. Audette, "Interactive map making for route planning and obstacle avoidance in an unstructured outdoor environment," M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2019. [Online]. Available: https://calhoun.nps.edu/handle/10945/60406

[6] C. Lebrun, "Vision-based terrain classification and learning to improve autonomous ground vehicle navigation in outdoor environments," M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2019.

[7] A. S. Miyakawa, "Autonomous ground vehicle low-profile obstacle avoidance using 2D LIDAR," M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2019.

[8] K. Rice, "Convolutional neural networks for detection and classification of maritime vessels in electro-optical satellite imagery," M.S. thesis, Dept. of Comput. Sci., NPS, Monterey, CA, USA, 2018. [Online]. Available: https://calhoun.nps.edu/handle/10945/61255

[9] H. Ben Abdallah, "Convolutional neural networks as feature extractors for data-scarce visual searches," M.S. thesis, Dept. of Inform. Technol. Manage., NPS, Monterey, CA, USA, 2016. [Online]. Available: https://calhoun.nps.edu/handle/10945/50597

[10] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, “A guide to convolutional neural networks for computer vision,” *Synth. Lect. Comput. Vis.*, vol. 8, no. 1, pp. 1–207, Feb. 2018. [Online]. doi: 10.2200/S00822ED1V01Y201712COV015

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[12] A. Deshpande, “The 9 deep learning papers you need to know about (Understanding CNNs Part 3),” August 24, 2016. [Online]. Available: https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. doi: 10.1145/3065386

[14] C. Szegedy *et al.*, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9. [Online]. doi: 10.1109/CVPR.2015.7298594

[15] M. Jain, “Paper explanation: Going deeper with convolutions (GoogLeNet),” June 9, 2018. [Online]. Available: https://mohitjain.me/2018/06/09/googlenet/

[16] CappuccinoPC, “SlimPRO SP675P i7 mini pc.” Accessed September 3, 2019. [Online]. Available: https://www.cappuccinopc.com/slimpro-sp675p.asp

[17] MobileRobots Inc., “Pioneer 3 operations manual.” Accessed September 3, 2019. [Online]. Available: https://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pioneer.pdf

[18] Génération Robots, “Pioneer P3-DX mobile robot - Génération Robots.” Accessed September 3, 2019. [Online]. Available: https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html

[19] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[20] N. Bloise, E. Capello, M. Dentis, and E. Punta, “Obstacle avoidance with potential field applied to a rendezvous maneuver,” *Appl. Sci.*, vol. 7, no. 10, pp. 1042, Oct. 2017. [Online]. doi: 10.3390/app7101042

[21] ROS Wiki, “Move_base - ROS Wiki.” Accessed October 19, 2019. [Online]. Available: http://wiki.ros.org/move_base

[22] ROS Wiki, “Frontier_exploration - ROS Wiki.” Accessed October 19, 2019. [Online]. Available: http://wiki.ros.org/frontier_exploration

[23] S. Riisgaard and M. R. Blas, “SLAM for dummies: A tutorial approach to simultaneous localization and mapping,” MIT, May, 2004. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/36832/16-412JSpring2004/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring2004/A3C5517F-C092-4554-AA43-232DC74609B3/0/1Aslam_blas_report.pdf

[24] Mathworks, “SLAM - MATLAB & simulink.” Accessed October 20, 2019. [Online]. Available: https://www.mathworks.com/help/nav/test_nav_category_mw_cb483bf7-0321-4a84-8f5c-511855ff92b6.html

[25] Mathworks, “Pure pursuit controller - MATLAB & simulink.” Accessed October 21, 2019. [Online]. Available: https://www.mathworks.com/help/robotics/ug/pure-pursuit-controller.html

THIS PAGE INTENTIONALLY LEFT BLANK

# INITIAL DISTRIBUTION LIST