OBJECT LOCALIZATION AND IDENTIFICATION FOR AUTONOMOUS
OPERATION OF SURFACE MARINE VEHICLES

by
Konstantinos Mentzelos

Bachelor in Naval Engineering, Hellenic Naval Academy, 2006

Submitted to the Departments of MechE and EECS
in Partial Fulfillment of the Requirements for the Degrees of
Naval Engineer and
Master of Science in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology
June 2016

© Massachusetts Institute of Technology. All rights reserved.

Signature redacted
Konstantinos Mentzelos
May 18, 2016

Certified by: Chryssostomos Chryssostomidis
Professor of Mechanical and Ocean Engineering
Thesis Supervisor

Certified by: Chathan Cooke
Principal Research Engineer
Thesis Supervisor

Certified by: Joe Harbour
Professor of the Practice of Naval Construction and Engineering
Thesis Supervisor

Accepted by: Rohan Abeyaratne
Professor of Mechanical Engineering
Chair, Department Committee on Graduate Students

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students
OBJECT LOCALIZATION AND IDENTIFICATION FOR AUTONOMOUS OPERATION OF SURFACE MARINE VEHICLES

by

Konstantinos Mentzelos

Submitted to the Departments of MechE and EECS in Partial Fulfillment of the Requirements for the Degrees of Naval Engineer and Master of Science in Electrical Engineering and Computer Science June 2016

ABSTRACT:

A method for autonomous navigation of surface marine vehicles is developed. A camera video stream is utilized as input to achieve object localization and identification by application of state-of-the-art Machine Learning algorithms. In particular, deep Convolutional Neural Networks are first trained offline using a collection of images of possible objects to be encountered (navy ships, sail boats, power boats, buoys, bridges, etc.). The trained network applied to new images returns real-time classification predictions with more than 93% accuracy. This information, along with distance and heading relative to the objects taken from the calibrated camera, allows for the precise determination of vehicle position with respect to its surrounding environment and is used to compute safe maneuvering and path planning strategy that conforms to the established marine navigation rules. These algorithms can be used in association with existing tools, such as LiDAR and GPS, to enable a completely autonomous marine vehicle.
ACKNOWLEDGEMENT:

Thank you to Athina for being a great wife, friend and mother to our beautiful kids.
# TABLE OF CONTENTS

I. **INTRODUCTION** ........................................................................................................... 13

II. **ANALYSIS OF THE PROCESS** .................................................................................... 15
    A. HORIZON LINE DETECTION ....................................................................................... 15
    B. ALTERNATE APPROACH FOR HORIZON DETECTION ............................................. 23
    C. OBJECT LOCALIZATION ............................................................................................ 27
    D. IMAGE IDENTIFICATION ............................................................................................. 31
    E. DEFINITIONS OF NEURAL NETWORKS ..................................................................... 33
    F. LAYERS OF CONVOLUTIONAL NEURAL NETWORKS ................................................. 37

III. **ANALYSIS OF IMAGE IDENTIFICATION EFFICIENCY** ........................................... 52
    A. DIFFERENT NETWORKS EVALUATED ....................................................................... 52
    B. NUMBER OF TRAINSET IMAGES ................................................................................. 57
    C. MINIBATCH SIZE ........................................................................................................ 63
    D. RESOLUTION OF IMAGES .......................................................................................... 68
    E. INTRODUCTION OF A NEW CATEGORY .................................................................... 71

IV. **CONTROL AND COMMAND** .................................................................................... 75
    A. HEADING AND DISTANCE OF OBJECTS .................................................................... 75
    B. PRINCIPLES OF MARINE PROPULSION .................................................................... 77

V. **ONBOARD IMPLEMENTATION** .................................................................................. 84

VI. **RESULTS** .................................................................................................................. 87

VII. **RECOMMENDATIONS** ............................................................................................ 93
    A. IMPROVING HORIZON LINE DETECTION ................................................................. 93
    B. IMPROVING OBJECT LOCALIZATION ....................................................................... 94
    C. IDENTIFYING SHAPES ON BUOYS ......................................................................... 96

VIII. **SUMMARY** ............................................................................................................. 98

IX. **REFERENCES** ........................................................................................................... 99
LIST OF TABLES

Table 1: Results for different experiments and configurations........... 61
Table 2: Progress of the training for 32x32 and 64x64 resolution ..... 70
Table 3: Calculations for various speeds........................................ 79
LIST OF FIGURES

Figure 1: Overall Process................................................................. 15
Figure 2: From left to right the original image, blurring [3x3] and  
        Blurring [7x7] ........................................................................ 17
Figure 3: From left to right the edges from the original, the [3x3]  
        blurring and the [7x7] blurred .............................................. 17
Figure 4: Horizon line detection with the simple_version .............. 19
Figure 5: Horizon line detection with the simple_version .............. 20
Figure 6: Existence of mountains misleads the simple_version ...... 21
Figure 7: Existence of clouds from left to right misleads the  
        simple_version ..................................................................... 21
Figure 8: The sliced_version of the algorithm solves the problem   
        with mountains ..................................................................... 22
Figure 9: The sliced_version of the algorithm solves the problem   
        with clouds.......................................................................... 23
Figure 10: Example of optimized horizon line detection .............. 26
Figure 11: Example of optimized horizon line detection .............. 27
Figure 12: Example of localizing objects ....................................... 30
Figure 13: Example of localizing objects ....................................... 31
Figure 14: Different types of buoys................................................. 32
Figure 15: Graphical representation of a Perceptron .................... 35
Figure 16: Graphical representation of weight adjustment .......... 37
Figure 17: Graphical representation of training and  
        prediction process ................................................................... 38
Figure 18: Convolution of an image with two independent filters ...... 39
Figure 19: Convolution between the top first block and the first filter  40
Figure 20: Convolution between the top second block and the first filter  41
Figure 21: Convolution between the top first block and the second filter  41
Figure 22: Convolution between the top second block and the second filter  42
Figure 23: Max pooling applied to an input volume........................... 43
I. INTRODUCTION

Great progress has been achieved over the last few years in developing Unmanned Autonomous Vehicles (UAVs) that can be used for commercial, scientific, military, transportation or other reasons. UAVs can do tasks that people used to do without paying the cost of high risks for human resources (money and power), hours of training or even lives. UAVs can reach areas of high pressure, temperature and carbon dioxide and low oxygen, in other words areas with conditions that cannot support human life. In addition to that, UAVs can accomplish missions in unknown and hostile environments for exploration, rescue and support.

With this project, we focus on the Autonomous Surface Marine Vehicles and the potential to drive the navigation to the next level. So far, super-efficient micro-processors, large-scale computers for optimization and big-data analysis, sensors of extreme sensitivity, complicated algorithms and other high-end tools are being used to control the vehicles. The above technologies still involve substantial human support, observation and decision making. The goal is to give these vehicles the opportunity to analyze the obtained measurements, so that the data can be used to avoid collisions, recognize possible threats and avoid them, predict and plan a trajectory and in general navigate safely. The most important step for this achievement is to implement image recognition – object categorization for the UAV to be able to distinguish for example a ship from a buoy.

We are testing two algorithms for detecting the horizon line, because this line can help the system to narrow down the area of search and proceed with object localization and identification faster. Both algorithms perform very well and we suggest more ways to make the process fully automated. We are also testing a classic algorithm for background subtraction, so that we can extract the possible objects from every frame. This algorithm works successfully for semi urban-water environments, like the one that we tested our system and we propose a high-end way to improve the object localization process. The last step
is the object identification. We are using Convolutional Neural Networks, which are important for Machine Learning in general and they are being used in other applications such as speech recognition on smartphones, smile recognition for digital SLRs and video cameras and face recognition for security systems on highly protected buildings and for aiding investigations made by law enforcement. We trained our system to recognize four different categories of objects with their unique characteristics and explore the level of difficulty to introduce more of them. We analyze all the parameters that can improve the prediction performance and we justify the notion that every image database is unique and no previous work with similar databases can be considered applicable and adjustable.

One chapter is being used to describe the hardware, more specifically the small vehicle and the equipment installed on it, that helped us to test the system. The next chapter gives us the basic marine propulsion equations used to calculate the power needed to move the above robot vehicle. It also gives a short description of the ideas implemented to calculate the relative heading and distance of the objects, considering that only the camera was available and no other tool, like the LiDAR, was used for this project. One chapter is dedicated to the configuration of the system that we propose and results from a final test with video captured from our vehicle. The final chapter is dedicated to suggestions, important for improving the above algorithms, make the system even more autonomous and expand the capabilities of Machine Learning.
II. ANALYSIS OF THE PROCESS

In this chapter, all the steps towards a better representation of the environment of operations and a safe navigation are described. The first step is the Horizon Line Detection and the second step is the Object Localization. For both these steps, simple and old-fashioned algorithms were used with acceptable performance. The description is made for better understanding, using the built-in C++ functions that were used. The goal was to create a basic and complete system that can support the final step and ways of improved performance are being suggested at the end of the report. The final and most important step is the Image Identification where Machine Learning gives the opportunity to achieve great results on predicting the type of an object.

A. HORIZON LINE DETECTION

The horizon is the line that separates the Earth from the sky. When we are outdoors we use the horizon as a line of reference to judge the scale and the distance of objects in relation to us. As a consequence the horizon line is very important for pilots, sailors and drivers of all kinds of vehicles (cars, trains, bikes, etc...). For an autonomous vehicle the human “eyes” might be replaced by many tools like RADARs, LASERs, GPS. However, as long as we can improve the digital video technology and all the algorithms that can process the frames taken from a video recorder, cameras will remain the main “eyes” of a robotic application. This kind of algorithmic improvements we are trying to suggest with this project and detecting efficiently the horizon line is our first goal.
There are two types of Earth-sky and water-sky (sea-level) horizons, the local (geometric) horizon where trees, buildings and mountains are included and the geographic where they don’t. For this project, we care only for sea-level horizons and just because very high objects (mostly mountains) may mislead our results, we focus on detecting the geographic horizon. One major assumption, which for a geographic horizon stands as justified, is that the horizon is not just a straight line, but also the lowest straight line on a frame, meaning that any other line below the horizon line will be part of sea surge and won’t extend from side to side.

A1) SIMPLE VERSION

In order to find an algorithm that runs fast enough to process the frames of a video, we must use a greedy approach. In general, a greedy algorithm is the one that makes the locally optimal choice at each stage, with the hope of finding a global optimum [Wikipedia]. For our project, this approach is called simple_version and it proved to be accurate enough in most cases. The first step is to convert the original image into a gray-scale image, so that to have only one channel to process. That can be done with OpenCV by typing “0” flag for the imread command.

A1.1) IMAGE PRE-PROCESSING

One way of filtering images was used frequently within this project and it is useful to make its importance clear from the beginning. Blurring or smoothing is a simple and frequently used image processing operation. By smoothing the colors and eliminating the edges between the objects we can reduce the noise of the image. There are many filters to blur an image, such as the median filter, the weighted average filter and the Gaussian filter. In our project the mean filter was used, which is the simplest of all and its outcome was very satisfying for our analysis. The filter takes as parameters the size of the box which is being normalized (nrows, ncolumns). A [3x3] blurring means that a box of size [3x3] with 9
elements is used in every loop of the algorithm. The filter that is being applied to that box is:

\[
F = \frac{1}{n_{\text{rows}}n_{\text{columns}}} \begin{pmatrix}
1 & \cdots & 1 \\
\vdots & \ddots & \vdots \\
1 & \cdots & 1
\end{pmatrix}
\]

As the size of the box is bigger, so more blurred the image becomes. On Figure 2 we have the [3x3] and the [7x7] blurring of an image and on Figure 3 we compare their black & white images where the edges between the colors are defined with white pixels.

![Figure 2: From left to right the original image, blurring [3x3] and blurring [7x7]](image)

![Figure 3: From left to right the edges from the original, the [3x3] blurred and the [7x7] blurred](image)

A1.2) EDGE DETECTION

The second step is to identify the edges on the image. By edges, we mean the lines on the image that follow extreme changes in color values. That can be done with the built-in function in OpenCV called Canny that uses the algorithm
Canny86 as it was proposed by J. Canny in 1986 [1]. The results of this process can be very good if the original image is clear from clouds and mountains. A fast way to eliminate clouds and edges in general is to actually apply the blurring filter, as it was described in the previous paragraph. We can also decrease the sensitivity parameter of the Canny function, so that the outcome doesn’t give edges created by the clouds. Our goal, however, is to find an algorithm for a fully automatic vehicle, so manual modifications are not acceptable. An obvious way to get rid of the existence of mountains and clouds is to use a sliced version of the algorithm and this approach will be explained later.

A1.3) VERTICAL BOUNDARY IDENTIFICATION

Having a black & white image with all the edges defined by white color in a dark background helps us to move to the third step of the algorithm. For this step we assume that the horizon line is the lowest distinct line of the image, meaning that below this line there is only water and any other object’s edges don’t play any important role for this step. For that reason the algorithm starts from the upper most part of the black & white image (row index zero) scanning pixels column by column. The algorithm is checking the color of a pixel and if it is white, then it populates the matrix with the coordinates of that pixel. For every column it is enough to get only one pixel and when the algorithm finds the first (upper one), it breaks the loop and it proceeds with the next column. Another condition check that arises from our above assumption is that the algorithm saves the coordinates of a white pixel only if this pixel is lower than the one of the previous column.

A1.4) POINTS FITTING

For the last step, the algorithm calls the built-in function in OpenCV called fitLine that uses the least square method to fit a line across the white pixels included in the final matrix of the previous step. The outcome of this function is the starting point of the line and its inclination. All four steps together are called
the simple_version of the algorithm. On Figures 4 and 5 successful horizon line detection examples with the simple version are given.

Figure 4: Horizon line detection with the simple_version
As mentioned in the beginning of the chapter, the greedy approach proved to be very accurate and fast with clear images. The algorithm is able to avoid tall objects that can mislead the least square method like ships and mountains on the background. Even the edges of small clouds dispersed in the sky are not included in the matrix of the third step. The results can be inaccurate in two cases: a) existence of mountains in the background and b) existence of clouds that expand from the left to the right side of the image (Figures 6 and 7). For both cases the algorithm draws the horizon line higher because it identifies the changes in colors for sky-clouds and sky-land respectively.
Figure 6: Existence of mountains misleads the simple_version

Figure 7: Existence of clouds from left to right misleads the simple_version
A2) SLICED VERSION

An easy way to solve the above problem without increasing the computational time needed is to split the image in vertical layers [24]. The concept is to run the simple_version for each layer and then combining the results. We are doing that because when thin slices are being processed, instead of the whole image, the Canny function can identify those areas where the clouds are just a little more dispersed or the mountains are showing a cut. This way we can drive the starting points as low as possible. The starting points produced from the fitLine of each layer enter a new elimination analysis. This analysis finds the lowest of the starting points and those points that are close to it. The proximity of the points is a parameter that can be configured and for the purposes of this project is set to number_or_rows of the original image over 100. For the last step the lowest starting point and the closest points to it are used in the built-in function fitLine and the outcome is the final horizon line of the image. This version of the algorithm is called sliced_version. On Figures 8 and 9 we get the improved horizon line of the above images.

Figure 8: The sliced_version of the algorithm solves the problem with mountains
With the above approach we are able to get accurate results for images of all kinds. In the worst case we are getting a horizon line a few pixels (~20) above the correct one. This result is acceptable, considering the fact that our goal from the process of finding the horizon line is to narrow down the searching area of the next step. For the object localization process we want to examine the color properties of the pixels around the horizon line and not the whole image.

B. ALTERNATE APPROACH FOR HORIZON DETECTION

The algorithm of the previous chapter works very fast and it gives us the opportunity to process each frame taken from the camera. The only problem with this approach is that the outcome from the Canny function depends on a parameter for which the selection of a wrong value can be catastrophic and can give us an inaccurate horizon line. For example, for an image that represents a landscape on a bright day, a parameter 50 will be very sensitive to every color variation and it will give edges for the clouds. As a result, the algorithm will be misled and the horizon line will be placed on the clouds. For the same image a parameter 150 will be less sensitive to color variations and the clouds won’t be shown on the binary image.
A possible way to solve the above problem is to automatically adjust the parameter using other variables. In other words, GPS location, time zone and weather forecasting can be used to identify when a day is bright or cloudy and when it is getting dark at night, so that the parameter will be set accordingly.

In order to avoid the above confusion and make sure that our system is fully automated we can use an optimization method that works very well in the detection of the horizon line for Autonomous Air Vehicles [3]. With this optimization method we are trying to quantify the assumption that sky pixels \((x^s)\) will look similar to other sky pixels and that water pixels \((x^w)\) will look similar to other water pixels. In other words, we want to maximize the optimization criterion:

\[
J = \frac{1}{|\Sigma^s| + |\Sigma^w| + (\lambda^s_1 + \lambda^s_2 + \lambda^w_3)^2 + (\lambda^w_1 + \lambda^w_2 + \lambda^w_3)^2}
\]

where (a) \(\Sigma^s, \Sigma^w\) are the covariance matrices for the sky and the water pixels respectively:

\[
\Sigma^s = \frac{1}{(n^s - 1)} \sum_{i=1}^{n^s} (x^s_i - \mu^s)(x^s_i - \mu^s)^T
\]

\[
\Sigma^w = \frac{1}{(n^w - 1)} \sum_{i=1}^{n^w} (x^w_i - \mu^w)(x^w_i - \mu^w)^T
\]

\((n\text{ is defined as the number of pixels for each distribution}),\)

(b) \(\mu\) is the mean for each distribution:

\[
\mu^s = \frac{1}{n^s} \sum_{i=1}^{n^s} x^s_i
\]

\[
\mu^w = \frac{1}{n^w} \sum_{i=1}^{n^w} x^w_i
\]

and (c) \(\lambda^s, \lambda^w\) are the eigenvalues of the covariance matrices.

For this method too, we assume that the horizon line is a straight line. This line can be defined from the general line equation:

\[
y = ax + b
\]
As a result we want to maximize the objective function/criterion \( J \) such that:

\[-\frac{\pi}{2} \leq a \leq \frac{\pi}{2}\]

and

\[0 \leq b \leq \text{number}_{\text{of}}_{\text{rows}}\]

The optimization was set up with C++ using the libraries `<nlopt.hpp>` and `<Eigen/Eigenvalues>`. With the first library we were able to initialize the optimization with two random variables (\( \text{opt opt(nlopt::LN_COBYLA,2)} \)) and define the objective function (\( \text{opt.set_min_objective} \)). COBYLA stands for Constrained Optimization BY Linear Approximations and it is an algorithm for derivative-free optimization with non-linear inequality and equality constraints, by M.J.D. Powell [15]. With the first library we could also define the inequality constraints (\( \text{opt.add_inequality_constraint} \)) and finally solve the optimization with the command \( \text{opt.optimize(x, minf)} \) where the vector \( x \) has the values for the random variables and \( \text{minf} \) the optimized value for the objective function. The second library was used as part of the objective function, where we had to calculate the covariance matrices (\( \text{calcCovarMatrix} \)) of the two distributions (sky and water pixels), the eigenvalues of the covariance matrices (\( \text{eigen} \)) and the determinants (\( \text{determinant} \)).

The first runs were executed with the original resolution of the image. With the hardware that we had available (Chapter V), the original resolution was very high and multiplications with large matrices need too much to be executed. In order to achieve better running time, we had to downsize the image resolution and then blur it. After pre-processing the images the running time for every frame was finally the same with the greedy algorithm of the previous chapter.

This method proved to be very accurate and after decreasing the resolution of each frame taken from a real-time video, the method also proved to be very fast. Taking into consideration that this method doesn’t depend on any manually fixed parameter, we can say that it is the obvious way for a fully automated system. However, the sliced version of the previous chapter was used when the system was tested, because of its simplicity. Both methods need improvements and in Chapter VI “Suggestions” another method is proposed that
can solve the horizon line detection problem with improved performance. On Figures 10 and 11 two examples of successful horizon line detection with the optimization method are given.

Figure 10: Example of optimized horizon line detection
C. OBJECT LOCALIZATION

On the previous two chapters, we described two methods to efficiently detect the horizon line. The next step towards a fully automated system that can identify objects and make the necessary decisions for safe navigation is to actually find all the objects around the vehicle. For our implementation we accept that the system can localize objects that are not important such as buildings. However, we know that the image identification step will recognize those unnecessary objects and get rid of them. We claim that the efficiency of our algorithm is acceptable and on the "Suggestions" chapter we suggest a method to improve it.
**C1) IMAGE PRE-PROCESSING**

The first step is to crop the part of the grayscale image extended a specific number of lines above and below the horizon line, as it was detected with the algorithm described in the previous chapter. The specific number is defined again as the **number_of_rows** over 100. The reason why we use only this part of the image is that we care about the variation of colors only around the horizon line and objects in the sky (planes and clouds) are not important for navigational purposes. We can also say that camera is placed at such height that objects below the horizon line are actually very close to the ship and their identification would be successful in earlier moments of the trip.

**C2) BINARY IMAGE**

The next step is to perform a reduction of the grayscale cropped image to a binary image. In other words, we want to find a threshold value and pixels of the grayscale image around the horizon line and with a color value above that threshold are taking the color white and the rest of the pixels are taking the color black. This way, we are getting a black & white image different than the one from Canny function.

In order to find the threshold value we can use Otsu’s Method, named after Nobuyuki Otsu [2], which is an algorithm that tries to separate the pixels into two classes (foreground and background pixels). For this method we have to create the graylevel histogram of the cropped image, using the built-in function in OpenCV called calcHist. Then, the algorithm finds the threshold, by maximizing the inter-class variance:

\[ \sigma^2(t) = \omega_b(t) \ast \omega_f(t) \ast [\mu_b(t) - \mu_f(t)]^2 \]

where \( \omega \) is the class probability:

\[
\omega_b(t) = \sum_{i=0}^{t-1} p(i) \\
\omega_f(t) = \sum_{i=t}^{L-1} p(i)
\]
(p is defined as the number of pixels for a specific color multiplied by the value of the color and L is 256 the number of grayscale colors for 8-bit images)

Also μ is the class mean:

\[ \mu_b(t) = \sum_{i=0}^{t-1} \frac{ip(i)}{\omega_b} \]

\[ \mu_f(t) = \sum_{i=t}^{L-1} \frac{ip(i)}{\omega_f} \]

The class probabilities and means are computed iteratively.

The result of the above process is a binary image, where the distinct and important objects are colored as white. The final step is to put a frame around those objects and save all the frames as independent images that can be used for the next step of the process, which is called image identification. On Figures 12 and 13 the algorithm is applied to two different images and gives us the black & white image and the original image with the horizon line and the detected objects.
Figure 12: Example of localizing objects
D. IMAGE IDENTIFICATION

In an autonomous system we are trying to simulate the human brain with algorithms, so that to apply a decision making process which is independent, accurate and robust. Once the system has located the possible objects, it needs a way to identify and use them for creating a complete representation of the environment. The final goal can be not only to navigate safely, but also in case of a navy ship to destroy the unfriendly objects.

For this project, TORCH [9] was used as a platform for Machine Learning applications. This is a LUA [10] based library, which is also used with other software such as Scikit-learn, Shogun, MiPack and H2O. A competition for image recognition is the training of a system and the prediction based on CIFAR-10 [11], which is an image database with ten categories and 60000 images. This is one of the many image databases created for image recognition competitions and it was the most applicable for this project, because one of the ten categories is SHIPS. That gave the opportunity to find a large number of already processed images for our system. Searching on the Internet, someone can find other people’s networks, specially designed for CIFAR-10 and written for TORCH. These networks showed high accuracy with the image database created for marine operations and that was the reason why TORCH was used for Machine Learning. One more advantage is that C is the core of LUA and it was really easy to call functions of LUA from C++ scripts.
For an autonomous vehicle that navigates at sea, it is very important to be able to identify buoys, ships and rocks. Buoys have various colors and shapes on them that give the proper information for restricted and safe navigation. For example a boat must keep the red buoys on the right side when it navigates towards the port and vice versa with a green buoy. Other examples are the orange circle on a white buoy that indicates speed limits, wash restrictions, etc and the orange diamond on a white buoy that marks random hazards such as shoals and rocks (Figure 14 from Pennsylvania Boating Handbook [21]). It is also very important for an autonomous system to be able to identify sail boats from engine boats, because the autonomous vehicle is power-driven and when it meets head-on a sail boat, it must “take early and substantial action to avoid collision by stopping, slowing down, or changing course” (International Maritime Organization [12]). In other words, the autonomous vehicle is, according to navigational rules, the “give-way vessel” and the sailboat is the “stand-on vessel”. On the contrary, when the autonomous vehicle meets a powerboat, both of them are “give-way vessels” and they must take actions and keep to the starboard side.

Figure 14: Different types of buoys
For the purposes of this project the system was trained to identify four categories of objects. A collection of 1800 images: 450 buoys, 450 sail boats, 450 navy ships and 450 engine boats were used to create a database of object images. Part of the Machine Learning process analysis was the introduction of one more category. The fifth category (bridges) was introduced step-by-step with just a small number of images at the beginning and more images as the analysis was going on. The goal of this analysis was to identify a minimum number of images that are needed, so that the training of a category gives accuracy more than 90%. The results of the analysis are given in Chapter IIIE.

The next two chapters present the general definitions of Neural Networks and their important layers, in order to completely understand how they work, before we analyze the parameters that affect their performance.

E. DEFINITIONS OF NEURAL NETWORKS

In order to make the concept of Convolutional Neural Networks more intuitive, we must first understand more general terms such as Machine Learning, Neural Networks, Backward and Forward Propagation Mode, Supervised Learning and learning rules. Then, we will be also able to understand more specific terms and methods like Convolution, Max Pooling, Normalization and Gradient Descent.

Machine learning is using data-driven algorithms (the data “tells” what the good answer is) that analyze the data and try to improve the system’s own understanding by detecting patterns in the data.

Neural Networks (or Artificial Neural Networks) are a Machine Learning technique that is inspired by neuronal structure of biological systems like the central nervous system and the human brain. A simple definition of a Neural Network is given by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen:
"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs”.

These processing elements are called “neurons” and they are organized in “hidden layers”. The connections between neurons have numeric weights that can be tuned based on experience, making Neural Networks adaptive to inputs such as the pixels of an image (input layers) and capable of learning. In other words, the most important process is the system training using methods such as Unsupervised and Supervised Learning (backward propagation mode). The training is divided into iterations, or “epochs”, where an iteration ends when the system uses every image of the training database just once. Once the system is trained to a satisfactory level, it can be used for predictions, which are given from the output layer (forward propagation mode).

E1) PERCEPTRON

An easy way to understand the concept of neurons is to describe the Perceptron [27], which is an artificial neuron developed in 1950s and 1960s by the scientist Frank Rosenblatt [28], inspired by earlier work by Warren McCulloch [29] and Walter Pitts [30] and [31]. The Perceptron is the simplest artificial neuron and just because the function that calculates the output is not continuous in zero, it cannot be used for optimization learning algorithms (Gradient Descent). This function is known as Activation function and in case of a Perceptron it is the step function (with derivative the Dirac delta function). It is more common to use other models with Activation functions like ReLU and Sigmoid, which are described in Chapter IIIF. A Perceptron takes several binary (0,1) inputs \( x_1, x_2, \ldots, x_n \) and produces a single binary output. Rosenblatt introduced the “weights” \( w_1, w_2, \ldots, w_n \) that are real numbers assigned to each input, in order to express the importance of every input to the calculation of the output. The neuron’s output is going to be 0 or 1, whether the weighted sum \( \sum w_i x_i \) is less or greater than a threshold value (real number). This threshold value and the weights are the
parameters of the neuron, calculated during the training process. On Figure 15 a graphical representation of a Perceptron with inputs, weights and output is given.

![Perceptron Diagram](image)

**Figure 15: Graphical Representation of a Perceptron**

Instead of the threshold value, Bias $b$ is being widely used, which is actually the negative of the threshold value. The mathematical expression for the neuron's output becomes:

$$output = \begin{cases} 0, & \text{when } \sum w_i x_i + b \leq 0 \\ 1, & \text{when } \sum w_i x_i + b > 0 \end{cases}$$

**E2) LEARNING PROCESSES**

The key difference between Supervised and Unsupervised Learning is that for the first the examples must be labeled. In order to better understand this, we can use an example of face recognition. We want to create a system that will be able to predict whether or not an unseen image is a face. For the Supervised method the training of the system must be done with a database of images and the user must explicitly say which ones are faces and which ones don't. During the evaluation process the system predicts the label for the training images and corrects the weights accordingly. On the contrary, for the Unsupervised method the database is not labeled. The algorithm tries to cluster the data into different groups that have common characteristics. There are also Machine Learning techniques such as, Semi-Supervised, Active and Reinforcement, but their definition is beyond the scope of this project, mainly because in this case, only Supervised Learning was used.

The most common Supervised Learning method, which sometimes is also used for Unsupervised Learning, is the Backward Propagation Mode, or else
Backpropagation. As mentioned in the previous paragraph, this method requires a known, desired output (label) for each input. For each iteration the system makes a guess as to what it might be the output for the specific input. It then compares the prediction with the right answer (label) and adjusts the weights of the node connections according to how far its prediction was from the label. Once the training is over the system enters the Forward Propagation Mode, where there is no feedback and no error signal to evaluate a potential solution. The system uses the connection weights defined from the training, in order to predict not only the labels of the unseen images (new inputs) in the test set (prediction process), but also the labels of all the images in the train set (validation process). The performance of the system on the validation process helps us to understand when the training is over and the system has taken advantage of all the available features given from the images on the train set.

During the Backward Propagation Mode, the system needs an optimization algorithm in order to adjust the connection weights and find a realistic solution with the lowest possible error away from the theoretical solution, given from the label. One of the optimization methods, which are used in Neural Networks, is the Gradient Descent, where the algorithm takes steps proportional to the negative of the gradient of the function at the current point. The cost function/criterion that we are trying to minimize is the quadratic cost function, Mean Squared Error (MSE):

\[
MSE = \frac{1}{n} \sum_{i=1}^{n} (y - t)^2
\]

The Gradient Descent learning rule, used to update the weights, is the Delta Rule (Wikipedia):

\[
\Delta w = \alpha x (t - y)f'
\]

where, \( \alpha \) is a small constant called learning rate, \( f' \) is the derivative of the neuron’s Activation function (Chapter IIIF), \( t \) is the object output, \( y \) is the actual output (from the activation function) and \( x \) is the input.
E3) ALTERNATE CRITERIA

There is another criterion used for optimization, the Cross Entropy Criterion [33] and another learning rule, the Hebb's Rule [34], but their description is beyond the scope of this project.

The last definition, valuable to be mentioned, is the Momentum. This is a parameter that helps us to train the system faster. It simply adds on the right side of the Delta Rule, a fraction of the weights calculated during the previous epoch. This addition not only drives with larger steps the optimization towards the global minimum, when the gradient keeps pointing in the same direction, but also eliminates the oscillations when the gradient keeps changing direction, because of a not so well-conditioned train set. The use of momentum arises the need of decreasing the learning rate of the Delta Rule.

On Figure 16 a graphical representation of the weight adjustment through the optimization process is given for better understanding of all the above.

![Graphical representation of the weight adjustment](image)

Figure 16: Graphical representation of the weight adjustment

F. LAYERS OF CONVOLUTIONAL NEURAL NETWORKS

On Figure 17 a graphical representation of the training and prediction process is given for better understanding of the Machine Learning technique for image identification. We also give a snapshot of how a Neural Network looks like with its important layers that will be analyzed in this Chapter.
F1) CONVOLUTIONAL LAYER

The Convolutional layer is the most important layer of a Convolutional Neural Network because its properly trained filters can be activated, when they identify some specific type of feature at some spatial position of the input. The main idea is that we apply onto the input (for example an image) a K number of filters (depth of the layer), which have a predefined small size \([F \times F]\) (usually three or five), called receptive field. We slide each filter across the width and the height of the input and we calculate the convolution, which is given by the below formula:

\[
c_{ij} = \sum_{l=0}^{d} \sum_{a=0}^{F-1} \sum_{b=0}^{F-1} x_{ab}^l f_{ab}
\]

where \(d\) is the third dimension of the input (if it is a color image, then \(d=3\)), \(x\) the values of the input (pixels in case of an image) and \(f\) the values of the filter. On Figure 18 (from LeNet – DeepLearning Documentation) we can see the results of the convolution between an image and two independent filters. Someone could say that “a randomly initialized filter acts very much like an edge detector” [22].

Figure 17: Graphical representation of training and prediction process
Figure 18: Convolution of an image with two independent filters

If the input, or else input volume, has dimensions $[N \times N \times d]$, then the output of the above process is a 3D volume with dimensions $[N-F+1 \times N-F+1 \times K]$. This way, we get $(N-F+1)^2 K$ neurons and each one of them is connected with an $[F \times F \times d]$ region of the input through numerical weights, which are adjusted during the training process. As a result each neuron has $F^2 d$ weights. For example, we have an image $[32\times 32\times 3]$, which is a typical RGB CIFAR-10 image and we choose a convolutional layer with two filters of size $[5\times 5]$. The output of the convolutional layer is a 3D volume with dimensions $[28\times 28\times 2]$ with 1568 neurons and each neuron has 75 weights, for a total number of weights equal to 117600. Even for this simple example the total number of weights that have to be adjusted in one layer of one epoch is very large. In order to decrease the number of weights and the running time especially for the training, we make one reasonable assumption: the weights used to identify a feature at some spatial position $(x_1, y_1)$ are useful to identify the same feature at a spatial position $(x_2, y_2)$. In other words the weights remain the same for one slice of the input volume. For the above example, then, we have only $3\times 75=225$ weights.

There are also other parameters that can affect the convolution layer. One of them is the stride $S$, which is used in order to avoid overlapping for the receptive fields between columns and rows. Stride 1 means that if the top left element of the input volume that convolute with one filter during the first iteration is the $(0,0)$, the next one on the same column is $(0,1)$ and the next one on the
same row is (1,0). On the contrary, when the stride is 2, the next element is (0,2) and (2,0) respectively. Another parameter is the zero-padding $P$, which defines the number of columns and rows with elements zero that are added in the beginning and at the end of the input volume. This number helps the user to control the size of the output volume. For example if we have $N=5$, $F=3$ and $S=3$, we cannot have $P=0$ because there aren't enough columns and rows. We have to use $P=2$ for a total number of nine columns and nine rows. With these two parameters, the calculated number of neurons for a convolutional layer is slightly different than above. The output volume has dimensions $[(N-F+2P)/S+1 \times (N-F+2P)/S+1 \times K]$ with a total number of neurons $((N-F+2P)/S+1)^2K$. For the above example of an input image $[32\times32\times3]$, with two filters of size $[5\times5]$, stride equal to 3 and zero-padding equal to 3 the output volume has size $[12\times12\times2]$, with 288 neurons and 225 weights.

On Figures 19-22 (from CS231n Convolutional Neural Networks for Visual Recognition [20]) a step-by-step visual representation of an input $[5\times5\times3]$ convoluted with two filters $[3\times3]$, stride equal to 2 and zero-padding equal to 1 is given as an example.

![Convolution](image)

Figure 19: Convolution between the top first block and the first filter
Figure 20: Convolution between the top second block and the first filter

Figure 21: Convolution between the middle first block and the first filter
F2) MAX POOLING LAYER

Another important layer of a Convolutional Neural Network is the Max Pooling layer. This layer is usually inserted between two successive convolutional layers, in order to reduce the size of the transfer volume and as a consequence to reduce the number of weights that the system has to calculate and adjust. The main idea is to apply a filter for every slice of the input volume. These filters don’t have weights as the ones at the convolutional layer, but they do have a predefined small size \([F \times F]\) and a stride \(S\). They are being slide across the width and the height of every slide and they are taking the max over the elements of a region \([F \times F]\). If the input volume has dimensions \([N \times N \times K]\), then the output of this layer is a volume of size \([(N-F)/S+1 \times (N-F)/S+1 \times K]\).

On Figures 23 and 24 (from CS231n Convolutional Neural Networks for Visual Recognition [20]) a visual representation of a \([2x2]\) max pooling layer with stride equal to 2, applied to an input volume with dimensions \([224x224x64]\) is
given as an example. The output volume has size [112x112x64]. We can also see the calculations for only one slide.

![Max pooling applied to an input volume](image)

**Figure 23:** Max pooling applied to an input volume

![Calculations of a single slide](image)

**Figure 24:** Calculations of a single slide

---

**F3) ACTIVATION FUNCTION LAYER**

ReLU or Rectified Linear Unit, is an Activation function that is applied to every neuron and introduces non-linearity to a Convolutional Neural Network [32]. As mentioned on the previous chapter, an Activation function works as a decision making function that calculates the output of a neuron and determines presence of a particular feature. Zero output means that the neuron doesn't see the feature and one means that it does. A linear Activation function, such as the linear perceptron, can be applied but the large number of neurons that must be used in computation and the unstable convergence are problems that make the training of the system inefficient. The goal of the training is to adjust the weights by small increments and see what happens. With a linear Activation function, small changes occurring in the weights cannot be reflected, because the function...
swings between zero and one. This is the reason why all the modern systems use Activation functions with some non-linearity in them. ReLU (Figure 25 from Wikipedia) is not the only non-linear activation function. There are others like the Hyperbolic Tangent and the Sigmoid, but ReLU simplifies Backpropagation, makes learning faster and avoids saturation issues, because the gradient doesn't vanish as we increase $x$. ReLU is defined as:

$$f(x) = \max(0, x)$$

where $x$ (here and below equations) is the biased weighted sum as with the Perceptron ($\sum w_i x_i + b$).

![Graphical representation of ReLU](image)

**Figure 25: Graphical representation of ReLU**

Recent researches [4] are trying to explore the efficiency of alternate versions of ReLU, such as the Leaky ReLU (LReLU) which is defined as:

$$f(x) = \max(0.01x, x)$$

and it allows a small non-zero gradient, when the neuron is not active and the SoftPlus which is defined as:

$$f(x) = \ln(1 + e^x)$$
and it is a smooth approximation of ReLU with non-zero gradient anywhere. Despite the fact that these versions might prove to be better than ReLU, for this project only ReLU was used.

A Sigmoid function (range of values \([0,1]\)) is a function that produces a curve with “S” shape (Figure 26), such as the logistic function [Wikipedia]:

\[
S(x) = \frac{1}{1 + e^{-x}}
\]

In general, this function is real-valued and differentiable, having a non-negative or non-positive first derivative, one local minimum and one local maximum. The reason of the popularity of this function in Neural Networks is because it satisfies a property between the derivative and itself such that it is computationally easy to perform:

\[
\frac{d}{dx} S(x) = S(x)(1 - S(x))
\]

![Figure 26: Graphical representation of Sigmoid function](image)

The Hyperbolic Tangent is related to the Sigmoid as follows:

\[
1 - 2S(x) = -\tanh \frac{x}{2}
\]

and as a symmetric version (range of values \([-1,1]\)) of a Sigmoid function, it converges faster [13].
The fourth important layer of a Convolutional Neural Network is the Normalization layer which can be applied to the whole set of inputs for his layer or a batch of this set. This method is used in order to reduce the phenomenon of "covariate shift", which is simply the degradation of performance for a system because the distribution of the unseen images varies from the distribution of the data/images used to train the model. The same thing can happen internally during the training with changes in the distribution of internal neurons of a very deep Neural Network. The last phenomenon is called "internal covariate shift" [8] and reducing it can give faster training time. The normalization layer subtracts the mean from the inputs and then divides them by the standard deviation:

$$
\mu = \frac{1}{m} \sum_{i=1}^{m} x_i
$$

$$
\sigma = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2}
$$
A scale and a shift is also applied in order to restore the "representation power" of the system and parameters $a, b$ are adjusted and learned along with the weights of the network.

$$y_i = ax_i + b$$

Normalization is used also as a pre-processing layer and more details and graphs are given later in this chapter.

**F5) DROP OUT LAYER**

One more layer, that is widely used, is called Dropout. It addresses overfitting which is the main problem of the Convolutional Neural Networks. This problem exists because the networks are very complex in order to search for features all over the image. In such complex networks the number of weights/parameters to be adjusted can reach one million and at the same time the train set is limited with, in the best case (CIFAR-10), fifty thousand images. During the training, the system might focus on specific features of the images that are actually noise and don't exist on the unseen images. This sampling noise will not affect the accuracy of the system on the train set during the validation process, but it will not let the system to generalize well. In other words the system is overfitted with unimportant features. The layer dropout is solving the problem by dropping out units from the network [6]. "By dropping out a unit, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability $p$ independent of other units, where $p$ can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks." This method can be used for the training of multiple different networks simultaneously and then averaging out the common weights.

Other methods to control the capacity of the network and prevent overfitting are the L1 and L2 regularization and the Max Norm Constraints. These
are not described because they were not used in any of the network for this project.

F6) FULLY CONNECTED LAYER

The Fully Connected Layer is the last important layer and the step towards the conversion of all the weights adjusted in the previous layers to a one-dimensional array that gives the probability for every class that the system is being trained. A Fully Connected layer takes all neurons in the previous layer and connects it to every single neuron it has.

F7) PRE-PROCESSING

Before applying the above layers to a Convolutional Neural Network, it is very important to apply data augmentation and pre-process all the images not only the ones of the train set, but also every unseen image that is being used for testing and for the actual implementation. The advantage of pre-processing is that our network will learn faster and achieve better performance. A simple way to explain that is to think of the below case. Imagine that the colors for one input image range between 120 and 255 (for 8-bit system) and the colors for another image range between 0 and 120. Our network should be able to learn to use smaller input weights for the first image and larger weights for the second. But it is a lot of work for the network to cope with different ranges. With this kind of differences, we also imply that one variable is more important than the other. Our network will probably find correct weights, but we can make its job easier, by giving it data in such a way that all the weights can remain in similar, predictable ranges.

With Data Augmentation, the system actually converts every image to an array of images. Elements of this array are samples of the original image. One sample can be for example of size [24x24] and start from row one (1) and column one (1). Another sample can be of size [28x28] (resized to [24x24] to match the dimensions) and start from row four 4 and column four 4 (Figure 28). We can also mirror the images vertically and horizontally and produce more samples.
Each sample can play its part to the whole procedure of training and predicting. This way the Neural Network can identify multiple objects from each image and this is part of the suggestion for improvement described in Chapter VIB. One more advantage of this process is that we increase the size of the train set and more images means better training.

![Data augmentation of an image into 18 samples (no mirroring applied)](image)

Figure 28: Data augmentation of an image into 18 samples (no mirroring applied)

Data pre-processing, called **Normalization** (like the Normalization layer), has two steps: 1) mean subtraction across every individual image of the database and 2) scaling of all the dimensions (3 for images) by dividing with the standard deviation. With the first step the cloud of data is being centered on the origin and the second step all the dimensions have the same range of values.
The second step seems to be worthless for image identification, because for 8-bit systems have already a specific range 0-255. However this step is being used to address very dark and very light images. Very dark images might have a range of colors 0-100 and very light images 150-250. After the normalization, all the images will have the same range, which for our example can be 100-200.

![Figure 29: Steps of Normalization (from CS231n Convolutional Neural Networks for Visual Recognition [20])](image)

There is also another way to pre-process the data, which is called Principal Component Analysis (PCA) and ZCA whitening. The main idea of this process is to remove the underlying correlation in the data. The first step is the same with Normalization and the data are centered on the origin. The next step (PCA) is to compute the covariance matrix, which gives their correlation. Two important properties of this matrix are: (1) the elements of the diagonal are actually the variances and (2) the matrix is symmetric and positive semi-definite. If we compute the SVD factorization of this matrix, we can get the eigenvectors and the singular values, which are equal to the eigenvalues squared because of the property (2) of the matrix. The eigenvectors are a set of orthonormal vectors and they can be regarded as basis vectors. So, we can project the zero-centered data into this eigenbasis, in other words we rotate the data so that the new axes are the eigenvectors. We can also pick only the first few eigenvectors, discarding the dimensions along which the data has no variance.

The final transformation is whitening the projected data from every dimension are being divided by the eigenvalue to normalize the scale. "The
The geometric interpretation of this transformation is that if the input data is a multivariable Gaussian, then the whitened data will be a Gaussian with zero mean and identity covariance matrix" [16].

Figure 30: PCA and ZCA Whitening (from CS231n Convolutional Neural Networks for Visual Recognition [20])
III. ANALYSIS OF IMAGE IDENTIFICATION EFFICIENCY

In this chapter, we explore the efficiency of the training versus several are variables such as the number of images on the train set, the batchsize and the image resolution. We care about efficiency not only in terms of accuracy of prediction, but also in terms of speed and size of memory needed for the system. Many experiments were run in order to get sufficient data and compare all the possible configurations. In Chapter D, the results from the training with four networks of multiple complexities are shown and in Chapter E we are trying to identify the least amount of images needed in order to introduce a new category of images for achieving 80% of accuracy in prediction. The learning rate and the momentum are the only two parameters, described in Chapter IIE, which were not part of our comparison analysis. The explanation for this decision is the fact that the values, which were used, proved to be the ideal for all the image identification applications that we found on the Internet. A learning rate equal to one and a momentum equal to 0.9 give the opportunity for fast and efficient training.

A. DIFFERENT NETWORKS EVALUATED

Part of our analysis was to use different Neural Networks with the same database and compare their ability to predict. The good performance of a network with a database doesn’t guarantee the good performance with another database. In other words, there are many variables that we have to control when we design an image identification system and just adding layers to the Convolutional Neural Network and overloading the processor with unnecessary computations is not always enough to improve the performance. For real-time applications not only the prediction accuracy, but also the processing time are important. Especially for an autonomous marine vehicle, a system that can process 10 frames per second is more preferable than a system that is 1% more accurate but it can process 2 frames per second. Less running time gives us the opportunity to increase the speed and the range of our vehicle.
Searching the Internet, we were able to find 6 different architectures for training a system with the image database CIFAR-10. Three of them are part of the work done from Nagadomi in Japan [25], who as for the moment has the fifth best prediction efficiency, with 94.15%, for the image recognition contest on CIFAR-10 and two of them are part of the work done from Sergey Zagoruyko [26], who has achieved 92.45% on CIFAR-10. We were able to experiment and use only the three networks from Nagadomi. Because of incompatibility issues that we were not able to resolve, we could not decrease the number of categories (from 10 to 4) and keep the two networks from Zagoruyko able to operate with our database. With more experience these networks can be modified and contribute to a future analysis. We can say that the next goal is to increase the number of categories that our system will be able to recognize and by making them 10, these two networks can be immediately available.

A1) SIMPLE ARCHITECTURE

In order to check how these architectures can perform with our database of four categories, we picked the 1500-300 for training/testing configuration and on the next chapter we discuss why this configuration can produce more realistic results. The sixth architecture, defined as Network4 for the rest of the chapter, is the simplest of all the architectures and it gave us the opportunity to experiment with TORCH and learn more about the design of Convolutional Neural Networks in general. It was rather simple to adjust this network to our database. The architecture has only two layers of convolution – max pooling, the Activation function is Hyperbolic Tangent and the data are only normalized and not augmented (Chapter IIIF). Despite that simple architecture, the system was able to achieve more than 55% of accuracy with CIFAR-10, a performance which was not even close to the desirable but it looked very promising considering the fact that it took only 5 minutes to setup the system and start the training.
A2) TRADITIONAL ARCHITECTURE

The next network that we used, defined as Network3, is the simplest out of the three Nagadomi’s networks and it achieved a maximum 92.77% and an average 90.57% of accuracy. It is a traditional Convolutional Neural Network with five layers of convolution, two max pooling, dropout with probability 50% (half of the neurons are being deactivated) for the fully connected layer and the activation function is ReLU (four layers). We use the term “traditional” to distinct those architectures from more recent ones, which are being used to address the problem with multiple variations of the same feature. The traditional networks are using generalized linear models (GLM) and as the word “generalized” implies these models are a flexible generalization of ordinary linear regression. With linear regression we can predict the expected value of a given unknown quantity (random variable) as a linear combination of the observed values (predictors). This means that a constant change in a predictor leads to a constant change in the random variable. In cases where the random variables have a normal distribution, for example the height of all men of a town, linear regression predicts with high accuracy. For other cases where the random variables can have distributions such as Poisson or Bernoulli, the generalized linear models are more flexible by allowing the linear model to be related to the random variables via the activation functions (Sigmoid, Hyperbolic Tangent and ReLU from Chapter IIIF) and by allowing the magnitude of each measurement to be a function of its predicted value.

A3) NETWORK IN NETWORK ARCHITECTURE

The next architecture, defined as Network2, is an example of “Network in Network” (NiN) architecture [14]. Its difference from the traditional networks is the use of micro networks instead of generalized linear models. In other words, inside a convolution layer there are multiple smaller networks consisting of fully connected layers with non-linear activation functions. These smaller networks are called multilayer perceptron (MLP) and can increase the level of abstraction for the model. By abstraction we mean the ability of the network to identify multiple
features from the same concept, for example a layer can identify the edges, another layer can identify the shapes and another layer can identify the colors. In order to achieve that with traditional GLM, we need many individual filters only for one convolution layer. However, having too many filters means extra computations for the next layer that has to combine all the outputs of the previous layer. Network2 is relatively simple with seven convolutional layers, two max pooling, one average pooling and seven layers of activation function (ReLU). It achieved a maximum prediction performance of 93.33% and an average of 90.55%. The fact that both Network2 and Network3, as long as Network1 that will be discussed in the next paragraph, are using ReLU as activation function verifies the opinion, expressed in Chapter IIF, that this activation function works better than old-fashioned ones like Step and Sigmoid functions.

A4) ADVANCED ARCHITECTURE

The last architecture that we used, defined as Network1, is the most advanced network provided by Nagadomi. This network has a NiN design, but much more complicated than Network2. It has eleven convolutional layers, three max pooling, three layers of 50% dropout, two layers of 25% dropout and ten layers of activation function (ReLU). The experiments showed that this architecture can predict with 93.66% maximum accuracy and 90.22% average.

The above results don’t prove that Network1 is the most advance among the three architectures from Nagadomi. To better justify this opinion, we have to say that the processing time per frame needed for this network is much less than the other three. More specifically for Network1 we need 5.5 minutes per iteration of the training process (1500 images), which means 0.22 seconds per image. On the contrary, for Network2 and Network3, we need 7 and 10 minutes respectively. For the actual prediction Network1 needs 0.14 seconds to process a frame and return an answer. Our goal is not only to create a system that can predict accurately, but also to do that fast enough, in order to have a real time
processing system. On Figure 31 the results from the above analysis are given graphically as a summary.

Figure 31: Results from the experiments with different architectures
B. NUMBER OF TRAINSET IMAGES

The most important parameter to achieve good training is the size of the database. For practical reasons it is not always very easy to gather all the images that you need at once. There are Internet sites where people not only have collected a huge amount of images, divided into categories and subcategories, and make them available to everyone, but also they offer them in different formats for various applications. One good example of such an Internet site is “Image-Net” [17], where someone can find images with frames around the objects; images that are used for real-time object localization and identification, as it is better described on the last chapter “Suggestions”. However, it is not guaranteed that you can find the images from a specific category, which is important for your system. For this project the category that felt in this case, was the buoys.

B1) 800 IMAGES

In order to analyze the importance of the train set size for the prediction performance on a Convolutional Neural Network, we used Network1 with minibatch size 64, because that was the most advance architecture that we had. In the early stages of the system training, the database had only 800 images, 200 for every category. Some of them were used as training images and the rest of them as testing images. To be more specific 3 different configurations were used: (1) 500-300, (2) 600-200 and (3) 700-100, with the first number defining the train set and the second the testing. With every configuration, three experiments were executed and in order to make the analysis as unbiased as possible, for each run with the same configuration the images were being shuffled and the sets had different images. Even this small number of images was enough to give us 82-83.5% of average accuracy for the unseen images. After 20 epochs the system achieved approximately 99% on the validation set and this observation means that there was no plenty of room for training improvement with such a small database. There were two other reasons that we
needed more images. The first reason was that our test set had only 100 images for some experiments and 200 for some others. This was both good and bad for our results. Good, because there were occasions where the test set contained easily identifiable images and the accuracy for those occasions was extremely high (~96%) and bad, because this high accuracy was unrealistic. Apart from that, we can say that for each error the system was losing 1% and 0.5% of its accuracy respectively and this way it was difficult to improve the overall performance. The second reason was that the system responded well and predicted with high accuracy just two categories (95%-100%), navy ships and buoys. For the other two categories, an average performance of 85% and 90% was implying that we needed more images to help the system improve its local prediction capability. The covariance matrices, as part of the detailed results that we were able to get from each experiment, showed that some of the engine boats were mistakenly predicted as sail boats and some of the sail boats were mistakenly predicted as engine boats and navy ships. The system needed more images to analyze and identify important features that will make both categories more predictable, such as the gray color for the navy ships and the sail for the sail boats.

**B2) 1200 IMAGES**

For the second set of experiments the database images were increased by 400, 100 for each category. That gave us more flexibility to create experiments of different configurations. The larger database helped the system to learn better, identify the important features of each category and finally improve its performance. More specifically the maximum accuracy reached 91.33% and the average was more than 85%.

For some occasions, such as the configuration 1100-100, the small test set gave us great accuracy (94%) and as it was mentioned in the previous paragraph, these performances were considered unrealistic because of the high possibility to get easily identifiable images in one set. Another way to justify this thought is the configuration 1500-300. Despite the fact that the train set is larger
than 1100, the maximum accuracy (93.66%) was worse, because the test set was a better mix of images.

On Table 1, apart from maximum and average performance of each experiment, the ratio of number of training images to the number of testing images is labeled. The initial goal of this comparison was to prove that higher the ratio, higher the overall performance of the system. That idea was abandoned quite early. A quick comparison between configurations 1500-200 and 900-100 will show that the key variable, which affects the prediction capability of the system, is the size of the train set. It is commonly known that the train set should have approximately the same number of images as the number of neurons for the Convolutional Neural Network. For a very complicated and advanced architecture this number can be more than a million and it needs a lot of work to great a database that big. Bottom line, as closer to that number as possible it is better and we can get even more accurate results.

B3) 1800 IMAGES

The last set of experiments was executed with the maximum number of images for this project. 450 images per category were enough to test the system to its limit, verifying that the whole project was successful. The configurations that were used are: 1) 1400-400, 2) 1500-300, 3) 1600-200 and 4) 1700-100, with the first number defining the train set and the second the testing. The third and the fourth configurations gave us very good results with a maximum performance of 96% for just one experiment and an average of the maximum performances, between all the experiments, equal to 94.25%. For the reasons that were mentioned earlier in this chapter, these results were considered less realistic. The first two configurations gave us results that illustrate better the real nature of our database, because the number of images for both the train and the test set was large enough and the maximum performance remain close to 93.66% for all the experiments with these configurations.
On Table 1 we present all the results for each iteration, experiment and configuration and on Figure 32 we are trying to summarize the results with a graphical representation of the maximum and average for every configuration.

Figure 32: Summary of the results for different configurations
<table>
<thead>
<tr>
<th>Train</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>1</td>
<td>72.3</td>
<td>72.3</td>
<td>74.7</td>
<td>78</td>
<td>74.3</td>
<td>63.5</td>
<td>76.5</td>
<td>76</td>
<td>73.1</td>
<td>75</td>
<td>77.7</td>
<td>73.8</td>
<td>76</td>
<td>76.5</td>
<td>80.5</td>
<td>73.3</td>
<td>82.3</td>
<td>73.7</td>
</tr>
<tr>
<td>2</td>
<td>77.3</td>
<td>77.3</td>
<td>81.3</td>
<td>78.3</td>
<td>78.6</td>
<td>76</td>
<td>77.5</td>
<td>71</td>
<td>79.5</td>
<td>76</td>
<td>77.3</td>
<td>77.1</td>
<td>83.5</td>
<td>82.5</td>
<td>82</td>
<td>82.7</td>
<td>86.7</td>
<td>83.3</td>
</tr>
<tr>
<td>3</td>
<td>78</td>
<td>79.7</td>
<td>77</td>
<td>79.3</td>
<td>78.5</td>
<td>80</td>
<td>82</td>
<td>82.5</td>
<td>82</td>
<td>81.6</td>
<td>64.7</td>
<td>62.3</td>
<td>86</td>
<td>66.7</td>
<td>84.9</td>
<td>85.8</td>
<td>87.8</td>
<td>83.3</td>
</tr>
<tr>
<td>4</td>
<td>83.7</td>
<td>87</td>
<td>78.7</td>
<td>81.3</td>
<td>82.7</td>
<td>80.5</td>
<td>82.5</td>
<td>82.5</td>
<td>83</td>
<td>82.1</td>
<td>86.7</td>
<td>86.7</td>
<td>83</td>
<td>83.3</td>
<td>64.9</td>
<td>91</td>
<td>88</td>
<td>89</td>
</tr>
<tr>
<td>5</td>
<td>84</td>
<td>87.3</td>
<td>86.3</td>
<td>84.3</td>
<td>85.5</td>
<td>86.5</td>
<td>81.5</td>
<td>86</td>
<td>87</td>
<td>85.3</td>
<td>86.3</td>
<td>86.3</td>
<td>82.7</td>
<td>84.9</td>
<td>91</td>
<td>89.5</td>
<td>88</td>
<td>83.8</td>
</tr>
<tr>
<td>6</td>
<td>84.3</td>
<td>87.7</td>
<td>89</td>
<td>84.7</td>
<td>86.4</td>
<td>84</td>
<td>86.5</td>
<td>86</td>
<td>84</td>
<td>85.1</td>
<td>85</td>
<td>84.7</td>
<td>86</td>
<td>86.5</td>
<td>89</td>
<td>89.3</td>
<td>90.5</td>
<td>89.6</td>
</tr>
<tr>
<td>7</td>
<td>87.7</td>
<td>86.3</td>
<td>86.7</td>
<td>88</td>
<td>87.2</td>
<td>84</td>
<td>82</td>
<td>85</td>
<td>87.5</td>
<td>84.6</td>
<td>86.7</td>
<td>86</td>
<td>84.3</td>
<td>88</td>
<td>86.2</td>
<td>91.8</td>
<td>91.3</td>
<td>91.3</td>
</tr>
<tr>
<td>8</td>
<td>88</td>
<td>86.7</td>
<td>85.3</td>
<td>85.7</td>
<td>86.4</td>
<td>86.5</td>
<td>85.5</td>
<td>85.3</td>
<td>88.5</td>
<td>86</td>
<td>87.7</td>
<td>86.7</td>
<td>88</td>
<td>87</td>
<td>87.3</td>
<td>91.5</td>
<td>90.5</td>
<td>89.8</td>
</tr>
<tr>
<td>9</td>
<td>86</td>
<td>88.3</td>
<td>89.7</td>
<td>88</td>
<td>88</td>
<td>87</td>
<td>85</td>
<td>88</td>
<td>88</td>
<td>87</td>
<td>87</td>
<td>86.5</td>
<td>90</td>
<td>88.3</td>
<td>90.5</td>
<td>89.6</td>
<td>91.3</td>
<td>91.3</td>
</tr>
<tr>
<td>10</td>
<td>89</td>
<td>83</td>
<td>89</td>
<td>89</td>
<td>87.5</td>
<td>85.5</td>
<td>83</td>
<td>89.5</td>
<td>84.5</td>
<td>85.6</td>
<td>87.3</td>
<td>90.3</td>
<td>89.3</td>
<td>86.7</td>
<td>88.4</td>
<td>93</td>
<td>91</td>
<td>92.3</td>
</tr>
<tr>
<td>11</td>
<td>83</td>
<td>85.3</td>
<td>88.7</td>
<td>88.3</td>
<td>86.3</td>
<td>85</td>
<td>86</td>
<td>84</td>
<td>88</td>
<td>88.3</td>
<td>85.3</td>
<td>89.7</td>
<td>87</td>
<td>87.8</td>
<td>91.8</td>
<td>88.3</td>
<td>92.3</td>
<td>90.6</td>
</tr>
<tr>
<td>12</td>
<td>89</td>
<td>88.7</td>
<td>88</td>
<td>86.7</td>
<td>88.1</td>
<td>85.5</td>
<td>85.7</td>
<td>86.5</td>
<td>88</td>
<td>88.4</td>
<td>87.3</td>
<td>91.3</td>
<td>88.3</td>
<td>87.3</td>
<td>88.6</td>
<td>91.3</td>
<td>91.8</td>
<td>93</td>
</tr>
<tr>
<td>13</td>
<td>89.3</td>
<td>87.7</td>
<td>88.3</td>
<td>88.7</td>
<td>86.5</td>
<td>86.5</td>
<td>84.5</td>
<td>86.5</td>
<td>85.5</td>
<td>88.3</td>
<td>87.3</td>
<td>87.3</td>
<td>88</td>
<td>87.5</td>
<td>92.5</td>
<td>90.3</td>
<td>91.5</td>
<td>91.4</td>
</tr>
<tr>
<td>14</td>
<td>90.7</td>
<td>88</td>
<td>91.3</td>
<td>91</td>
<td>90.2</td>
<td>87</td>
<td>85</td>
<td>87.5</td>
<td>84</td>
<td>85.3</td>
<td>83</td>
<td>83.7</td>
<td>88.3</td>
<td>87</td>
<td>85.5</td>
<td>91.8</td>
<td>92</td>
<td>91</td>
</tr>
<tr>
<td>15</td>
<td>89.5</td>
<td>88</td>
<td>88.5</td>
<td>87</td>
<td>86.3</td>
<td>83</td>
<td>86.3</td>
<td>86.7</td>
<td>89.3</td>
<td>87.1</td>
<td>91.8</td>
<td>90.3</td>
<td>86.8</td>
<td>90.3</td>
<td>89.7</td>
<td>98.3</td>
<td>90.5</td>
<td>90.5</td>
</tr>
<tr>
<td>16</td>
<td>87.7</td>
<td>85.7</td>
<td>89.3</td>
<td>86.3</td>
<td>87.2</td>
<td>92.5</td>
<td>92.3</td>
<td>92.5</td>
<td>92.4</td>
<td>91.7</td>
<td>97.3</td>
<td>87</td>
<td>88.3</td>
<td>87</td>
<td>87.8</td>
<td>87</td>
<td>88.8</td>
<td>89.7</td>
</tr>
<tr>
<td>17</td>
<td>89.3</td>
<td>88</td>
<td>90</td>
<td>89</td>
<td>89.1</td>
<td>90.5</td>
<td>93.3</td>
<td>92.6</td>
<td>92.2</td>
<td>91.3</td>
<td>86.3</td>
<td>89</td>
<td>88.5</td>
<td>86.5</td>
<td>87</td>
<td>88</td>
<td>87</td>
<td>87.1</td>
</tr>
<tr>
<td>18</td>
<td>82.5</td>
<td>93.3</td>
<td>93</td>
<td>92.9</td>
<td>92.5</td>
<td>93.3</td>
<td>93</td>
<td>92.9</td>
<td>92.5</td>
<td>93.3</td>
<td>92.9</td>
<td>92.5</td>
<td>93.3</td>
<td>93</td>
<td>92.9</td>
<td>92.5</td>
<td>93.3</td>
<td>92.9</td>
</tr>
</tbody>
</table>

| verag | 84.5 | 84.7 | 85.3 | 85.1 | 84.9 | 83.1 | 84.4 | 84.7 | 83.9 | 85.4 | 85.9 | 85.4 | 85.4 | 86.2 | 86.4 | 86.4 | 86.6 | 87.6 |
| Max   | 90.7 | 88.7 | 91.3 | 91 | 90.4 | 89.5 | 88 | 89.5 | 88.9 | 89.3 | 91.3 | 90 | 88.3 | 90 | 93 | 93.3 | 93.1 | 91.7 | 89.3 |
| Ratio | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 | 1.67 |

Table 1: Results for different experiments and configurations
<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
</tr>
<tr>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
</tr>
<tr>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
</tr>
<tr>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
</tr>
<tr>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
</tr>
<tr>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
</tr>
<tr>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
<td>787</td>
</tr>
</tbody>
</table>

Table 1: Results for different experiments and configurations
C. MINIBATCH SIZE

In Chapter IIIE we describe the Gradient Descent as an optimization method to adjust the weights of the neurons, so that to minimize the prediction error of our system. The main idea is to follow the negative gradient of the function at the current point. There are two different ways to compute the gradient, Batch and Stochastic Gradient Descent (SGD), and the expected form of the error function helps us to pick one of them. With the first method we use the whole dataset to compute the gradient and this method is great for convex or relatively smooth error manifolds. The training needs in general less epochs and the correction moves directly to an optimum solution, either local or global. On the other hand, SGD uses a single sample (pure stochastic) or more often a minibatch of samples. SGD works better for error manifolds that have many local minima. In this case, the noisier gradient of a small batch helps the model to skip a local minimum and move to an area, which is hopefully more optimal. In other words, we can say that the overall prediction performance is better. Computationally this method is more memory efficient, because we don’t have to save in RAM large datasets and minibatches can be stored, loaded and worked much easier and faster. However, the computational advantage is leveraged by the need to perform many more iterations to achieve the best accuracy. In general SGD is a better method and we used it through all the system trainings.

In this chapter, we are trying to show the importance of setting the right minibatch for each application. For this purpose, we run experiments using CIFAR-10 database and not our database, with ships and buoys, because the large number of images can represent more accurately the existence of many local minima that mislead the training. We also used Network4, as it is described in Chapter IIID, because its simple architecture cannot absorb the differences in the configuration and a simple modification can drive to more obvious changes onto the results and performance. The other three networks have very complicated architectures and perform great from the very first epoch, without giving the opportunity to improve the prediction accuracy by just increasing/decreasing the minibatch size. Another reason that we used
Network4 is that it was the only one with a very flexible minibatch configuration. The other three networks accepted only minibatch sizes that are multiples of 32. The not so high-end graphics card could not manage computations of more than 64 samples and as a result we would run experiments with only 32 and 64 minibatch size. With Network4, we were able to explore sizes of 1, 2, 4, 8, 16, 32 and 64 samples.

The results from running pure stochastic gradient (1 sample minibatch) were the expected. The system achieved good performance (56.57%) from the first epoch and the accuracy on the validation set reached 100% in only 20 epochs. We referred to 56.57% as a good performance, considering that this architecture is the simplest of all and the best performance, recorded from the early experiments on CIFAR-10, was approximately 67%. Increasing the minibatch to 2, we noticed a slight improvement on both the average (from 64.6 to 65%) and the maximum prediction accuracy (from 66.15 to 66.74%). In addition to this improvement we could say that the model performed worse during the first epoch (50.68%) and it needed 29 epochs to reach 100% on the validation set. That came to confirm that the smaller the size, the higher the noise of the gradient and the model bounced more drastically from minimum to minimum. Running the experiment with a size of 4 samples, we found out that the maximum accuracy was still improving by reaching 67.08%. On the contrary, we could not say the same for the average percentage (64.79%), which decreased slightly and gave the impression that we found the best minibatch configuration. The impression proved to be true with 8 samples. Both maximum and average performances were lower than before (66.87% and 64.37% respectively) and this fact verifies the claim that higher the size of the minibatch doesn’t necessarily mean better overall performance. The performance depends on the number of local minima that exists on the train set. Each train set has unique features and there is no global rule for what size of minibatch we should use for every one of them. A small minibatch can skip the unimportant minima and fall on the global one very quickly. On the other hand, the big jumps from minima to minima can lead the model to skip the global minimum and never
return to it after many iterations. This is a reason that we prefer bigger minibatches. Less noise means smaller jumps between minima and higher probability to reach the global minimum. But a very big minibatch size for our database can lead to a very slow training, so slow that the model won’t be able to look through all the points of the error graph.

![Figure 33: Training progress for different minibatch sizes (100 iterations)](image)

The last comments were verified with the experiments of 16, 32 and 64 samples for minibatch. The average performance decreased very much because the number of iterations was not enough for the complete training of the system. With 16 samples the model reached 92% on the validation set after 100 epochs and it gave 66.84% for maximum and 63.3% for average performance. On the
contrary both 32 and 64 samples needed more than 100 iterations and they were able to achieve only 79.12% and 64% on the validation set respectively.

Because of the above results with Network4, we wanted to test the best network we had (Network1) with our database and 32 samples to check if we could get better results. The results for image identification, described in Chapter IID, and the results from the analysis of train set size (Chapter IIIA), image resolution (Chapter IIIC), different networks (Chapter IIID) and introduction of a new category (Chapter IIIE) were taken with minibatch size equal to 64, because that was the maximum size that our graphics card could support and because when we started the experiments, our experience was limited and the initial idea was that higher the size, better the performance. We run three experiments with 32 samples, in order to re-shuffle the database and get more realistic results. The results showed that there is no valuable difference between the two configurations. The maximum performance was the same (93.66%) and the average performance was around 90% for all the runs. Even the training time was similar. In other words, there was no reason to change the minibatch size and re-run the comparison analysis that we had done at that point.
The above results imply that a lower minibatch size might perform better. However, we set a limit of 100 iterations for training, so the systems with sizes 64 and 32 didn’t have the chance to finish their training. There is a chance that after 200 iterations, those systems could perform better. The only thing that we can see for sure is that every database is unique and someone has to try all the configurations to find the best.
D. RESOLUTION OF IMAGES

In Chapter II F, a detailed description of the layers for a Convolutional Neural Network is given, in order to have a complete understanding of the training and the prediction of a system. The convolutional layer is the most important layer, because the adjusted weights of the filters can identify the important features of the image. A low resolution image loses the ability to represent all the details and especially those features that are relatively small to the size of the image. For example, a buoy on a sea far from the camera will look like a simple dot on a low resolution image of this landscape. As a consequence, the system won’t be able to localize the dot and translate it to a buoy and the overall efficiency of the system drops down. For some cases this is not a big problem, but for some others, like the safe navigation of an autonomous marine vehicle, the non-identification of a buoy as soon as possible might have unexpected consequences. On Figure 35 we compare the different resolutions for the same image.

Figure 35: From left to right an image with 32x32, 64x64 and 128x128 resolution

Another more sophisticated way to justify the importance of a high resolution image is the data augmentation process, described in Chapter IID. A higher resolution image gives the opportunity to take more samples from a single image and convert the image to a long array of elements to be analyzed. More samples means more features to be identified at the same time. For the experiment described in the next paragraph the [32x32] image gave 36 samples and the [64x64] image 80 samples.
In order to create an experiment and test the accuracy of the above claim, two databases were used, one database with images of size [32x32] and one with images of size [64x64]. Both databases were created from the same high resolution images after resizing and blurring them to make the color transition smoother. To force the experiment produce unquestionable results and prove the advantage of the high resolution images, two configurations were chosen. The size of the databases was small, only 100 images per each category were imported and the Convolutional Neural Network used was not one of the most complicated (Network4). With these two configurations we expected to get a not so good accuracy for [32x32] images and a very good performance of the system with [64x64] images. From the 400 images, 300 of them assigned as train set and the rest were the unseen images of the test set. Three runs of the experiment with 20 epochs for each one of them were executed so that to re-shuffle the images two times and get more realistic results by averaging the numbers. This number of epochs was used because the [64x64] database proved to need only 20 epochs to achieve almost 100% accuracy on the validation set.

The results showed that the initial claim was justified. To be more specific, the training of the database with high resolution images gave a maximum accuracy of 94% and an average accuracy of 92%. The system was able to achieve this performance after the first 5 runs. The rest 15 epochs didn’t actually improve the performance not only globally, but also locally where two categories, navy ships and buoys, had almost 100% accuracy and the other two categories performed more weakly with accuracy around 90%. The above two observations could easily express the need for a larger train set, with more features to be identified that will help the system to predict more accurately. On the other hand, the training of the database with images of size [32x32] was unsatisfactory. The low resolution of the images and the very small train set gave to the system a maximum accuracy of 77% and an average accuracy of 33.75%. The very low average accuracy can be explained by the fact that the system was literally inactive for the first 16 epochs with accuracy 25% for both the validation and the
test set. With the term “inactive, we mean that the system is unable to predict and it was actually assigning all the images to just one category because there were no significant features to be identified at the beginning. Those inactive periods were not part of the calculations being made in order to find the average. At the 17th epoch the system was finally able to predict accurately some images, but 3 more iterations were not enough to achieve the numbers of the high resolution images.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>64x64</th>
<th>32x32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train size</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Test size</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>94</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>94</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>92</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>91</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>93</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>94</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>92</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>94</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>92</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>92</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>94</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>93</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>92</td>
<td>25</td>
</tr>
<tr>
<td>15</td>
<td>91</td>
<td>25</td>
</tr>
<tr>
<td>16</td>
<td>93</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>92</td>
<td>49</td>
</tr>
<tr>
<td>18</td>
<td>93</td>
<td>72</td>
</tr>
<tr>
<td>19</td>
<td>93</td>
<td>77</td>
</tr>
<tr>
<td>20</td>
<td>93</td>
<td>77</td>
</tr>
<tr>
<td>Average</td>
<td>92.7</td>
<td>33.75</td>
</tr>
<tr>
<td>Max</td>
<td>94</td>
<td>77</td>
</tr>
</tbody>
</table>

Table 2: Progress of the training for 32x32 and 64x64 resolution

The above results leave no room for further exploration but there is a trade off. The training of the database with high resolution images lasted for 20 hours. More specifically every training epoch needed 52 minutes to be completed and every testing epoch needed 5 minutes. There is no reason to be conservative with the available time for training, since we accept the fact that it is time-worthy.
However the time for prediction is very valuable. For real time applications, where high definition videos have a processing rate of 30, 60 or even 120 frames per second, the time needed for the prediction of one single image must be small. For our experiment, the testing epoch of 100 images lasted 5 minutes. That gives us only 0.33 frames per second, which is not acceptable for a high technology application. On the other hand, the running time for testing 100 images of low resolution was 15 seconds and the gives us almost 6 frames per second. This rate can be accepted considering the fact that the systems used on robotic applications are more power than the computer used for this experiment.

Both configurations can improve their performance by (1) importing more images on the train set, (2) decreasing the number of samples taken from every image, (3) increasing the number of epochs and (4) setting the image resolution not too high, not too low. Bottom line is that the image resolution has a positive influence to the prediction performance and a negative influence to the time response performance of the system.

E. INTRODUCTION OF A NEW CATEGORY

The implemented system for this project was tested in real time on an autonomous marine vehicle. The system was trained for identifying four categories of possible objects, navy ships, sail boats, engine boats and buoys. The vehicle is supposed to navigate for research purposes in the littoral, in other words close to coasts and through rivers. That is the reason why the above categories are not enough for a system that has to be completely autonomous and navigate around areas with multiple objects that can threaten and change its course. Goal of this chapter is to introduce step by step a new category to the system, in order to give a sense how that can happen towards a more complete system. An acceptable introduction of a new category is one that achieves locally over 80% of prediction accuracy and it doesn’t affect negatively not only the global accuracy of the system, but also the local accuracy of the categories that pre-exist.
The fifth category was images from bridges of all kinds. This category is easily distinguishable from the other four categories especially because of the architecture-shape of the objects. Someone can say that this category cannot be a representative example of importing a new set of images for prediction to the system, because the distinct differences on the shape, naturally, won't affect the local performance of the system. In other words, there is no way that the system will accidentally identify a buoy as bridge, or a bridge as ship. It would be better to pick a category that can really change the overall performance of the system. However, this category was picked for the promising applications in the future. It is a challenge for the system to be able to answer the questions (1) is the bridge up or down, (2) is the bridge high enough for the boat to go under it and (3) is it a bridge or the front gate of a flood barrier? It is very possible that Machine Learning can help answering these questions. For example, we can split the images of bridges into two different categories, one for open bridges and one for close ones and train the system to recognize both configurations. We can also draw two lines for all the bridge images of our database, one at the plane where the supports touch the water surface and the other at the upper point of the inner arc. The system after the training will be able to identify a bridge and when it does, it will also draw the above lines on the unseen image. Furthermore, the distance between those two parallel lines can be computed and the outcome will be the available height below the bridge. Under these thoughts, it seemed to be very useful and helpful to create a database of bridges that can be used eventually for solving the above problems.

For the experiments we used Network1 with minibatch size equal to 64. We imported all the images from the initial four categories and 1500 of them created the train set and the rest 300 the test set. We must say one more time that the epochs were the system was inactive were not used for the calculations of the average value. For the first run we downloaded 25 images for bridges (20 for training and 5 for testing). We knew that with such a small number of images the results could not be very good. The system was able to identify a few of the bridges only after 13 iterations and the best performance was 60%, which was
really promising for just 20 images. In general the rest of the categories were not affected by the introduction of a new category and for the epoch with maximum performance, only one ship and one navy ship were mistakenly considered to be bridge. For the second run we increased the number to 55 (50 for training and 5 for testing). The results were much better, with the maximum prediction percentage reaching 80% and only either one buoy or one ship was mistakenly identified as bridge. The most important observation was that the system was activated sooner (9th epoch) and this fact improved the average local performance from 42.85% to 51.66%.

Figure 36: Local average performance for different configurations (Train set/Test set)
For the next run we introduced 25 more images. A total number of 80 images for bridges gave us results that were very close to the desired. With 70 images for training, the system was able to identify more important features of the bridge architecture and it reached 80% for prediction. The system was inactive only for 4 epochs and the average performance was 53.75%. With 30 more images (100 for training and 10 for testing), the system showed only improved average performance with 57.5%. Two more runs were executed with 150 images. The first had 120 images for training and the second 130. Both runs got activated one iteration earlier with average performance 61.37 and 67.94 respectively. A unexpected observation from the last two runs was that the local performance of the system was decreased. More specifically the initial four categories showed in some cases local performance almost 2% less, because some of the images (2 for ships, 3 for navy ships and 1 buoy) were mistakenly predicted as bridges. This phenomenon can be explained from the fact that more images on one category means increased number of features to be identified and as a consequence the system needs more and more images to process the new features. In other words, increasing the number of images by just 10 or 20 can make things worse.

The ideal would be, as mentioned in Chapter IIE, to have the same number of images for all the categories. However, the important conclusion from the above experiments is that for a database with around 500 images per category, 100 images are enough to achieve acceptable performance for a new category.
IV. COMMAND AND CONTROL

In this chapter, we are trying to combine all the above algorithms and finalize the design of the system. In order to do that we need to translate the information taken from the camera (how a sailboat can affect our course?), mitigate other information (how far is the sailboat?) and transform them into actual commands (angle of the rudder) that the system will use to navigate. The first chapter tries to solve the problem of finding the relative heading and the distance of the objects by using only the frames taken from the camera. The second chapter enumerates a list of equations that can be implemented into the system so that we can calculate the power needed for the vehicle to reach a specific speed.

A. HEADING AND DISTANCE OF OBJECTS

The first step for autonomous navigation is to identify all the objects that are in the same area with your marine vehicle. These objects can force the vehicle to change its course and speed in order to avoid a collision with another boat, to protect divers or swimmers, to protect the vehicle itself from not so deep waters and other cases that can make the navigation unsafe. However it is not enough just to predict the type of the object. The vehicle has to know how far and in which direction, to be able to plan its course. In this chapter, the methods to approximate the important information of speed and relative heading are being described towards the complete system that was used.

For general application, it would be easier to use onboard tools for placing the autonomous vehicle and the surrounding objects on the area map. First of all a LiDAR could be used, which is a sensor that can create 3D high resolution maps by illuminating the objects with LASER light. Every point of the 3D representation can give its distance and its relative heading to the origin of space, where our vehicle is being placed. Another well known tool is GPS which can place the autonomous vehicle and other objects on the area map, by combining the object information taken from LiDAR and simple geometry. Using
information from other sensors like sonar, thermometer and speedometers the
vehicle can design its own course and correct speed-heading configurations with
positioning feedback from GPS. As it will be described in Chapter V, there is
already software, called MOOS-ivp, which can manipulate the above data and
control an autonomous vehicle. For our implementation we used only the data
from the camera, mainly because the goal of the project was object identification.
We wanted to test the efficiency of the horizon line detection algorithm, the object
localization algorithm and the image identification with Machine Learning in the
simplest way possible. The methods were very sensitive to sea-state, but the
final test was done with video from the river, where the water was very calm. The
methods responded very well under these conditions and their use was
acceptable considering the lack of the above tools.

In order to calculate the distance of the objects, we assumed that we know
the height of the objects. Having the height of the object known we use the below
equation to approximate the distance of the object:

\[ d = \frac{f(\text{mm}) \times h_{\text{real}}(\text{mm}) \times \text{res}(\text{pixels})}{h_{\text{measured}}(\text{pixels}) \times h_{\text{camera}}} \]

where, \( f \) is the focal length and \( \text{res} \) is the vertical resolution of the camera, \( h_{\text{camera}} \)
the height of the camera from the water surface, \( h_{\text{real}} \) the known height of the
object and \( h_{\text{measured}} \) the measured height of the object on the image.

The most important step to create an algorithm that can approximate the
relative heading of a object is the calibration of the camera. The camera was
placed outdoors on the sailing pavilion of the university. This area was very good
because it was large enough to give us the opportunity to place objects in various
distances away from the camera. Another reason was that the existence of water
and other objects, such as boats and buoys and the light/weather conditions
gave us an environment similar to the environment of operations. The objects
were placed in various headings and distances from the camera and every time
the center of the object area, projected on the image, was the point from where
the headings were calculated. The collection of points was used to create
imaginary diagonals, for which the point of origin was on the lower row of the
image. Three diagonals for headings 10, 20, 30° to the right of the image centerline and another three to the left were created by using MS EXCEL. For every layer between two diagonals, the points assumed to have the same relative heading from the camera. All the groups were plotted on a graph, by using the x and y values of their points. For each group, a trendline (default function of EXCEL) was formatted as a linear function to approximate the function that goes through all the points. The equations from all the trendlines were used in C++ in order to create the heading layers. A layer between each diagonal represented a specific heading and all the points on this layer assumed to have the same heading from the camera. Then, the object localization algorithm was putting frames around the possible objects, was calculating the center of the frame area and that center was used to find the right layer and the heading from the camera.

Figure 37: Results from camera calibration

B. PRINCIPLES OF MARINE PROPULSION
The vehicle that was used to test the system is a test-platform for many other applications that are being developed at the same time. It is valuable to know the limits of the vehicle, so that you can configure the system accordingly. For this project, the maximum speed of the vehicle evaluates the processing capabilities of the object localization and identification system. In this chapter we describe the equations that were used in order to calculate the total resistance of the vehicle and the power needed to overcome the resistance and develop a desired speed.

The first step is to calculate the Total Resistance/Drag of the vehicle. The shape of the hull can be considered as a catamaran, where two cylindrical shape pieces with half spheres on both sides are touching the water surface and give the necessary stability to the vehicle. The Total Resistance \( R_T \) is divided into two parts, (a) the Pressure-Form Drag that arises as the fluid impacts with a half sphere that is moving with its cross section perpendicular to the vector of the fluid motion and (b) the Skin Friction Drag, which arises from the friction of the fluid against the “skin” of the side surface.

For the calculation of the Pressure Drag in Newton’s (N) the equation below was used:

\[
R_d = C_d \times 0.5 \times p \times V_s^2 \times A_c
\]

where,

- \( C_d \rightarrow \) Drag coefficient \( \approx 0.295 \) for a half sphere
- \( p \rightarrow \) Density of the fluid (999.972 kg/m\(^3\) for fresh water at 20°C)
- \( V_s \rightarrow \) Speed of the vehicle (m/s)
- \( A_c \rightarrow \) Area of the cross section (m\(^2\)) \( = \frac{\pi r^2}{2} \)

For the calculation of the Skin Friction Drag in Newton’s (N) the equation below was used:

\[
R_f = C_f \times 0.5 \times p \times V_s^2 \times A_s
\]

where,

- \( C_f \rightarrow \) Friction Coefficient
\[ A_s \rightarrow \text{Area of side surface (m}^2) = \pi rL \]

For our vehicle, each pontoon has length (L) approximately 10 feet (3m) and diameter (d) 16.75 inches (0.21m). The Friction Coefficient is dependent on Reynolds Number (Re). Therefore, in order to calculate the Friction Coefficient, the 1/7 power law, derived by Theodore von Karman, was used. The below equation is valid for Reynolds Numbers in the range \((5\times10^5,10^7)\):

\[ C_f = \frac{0.058}{Re^{0.2}} \]

The equation for Reynolds Number is:

\[ Re = \frac{V_pL}{\mu} \]

where,

\[ \mu \rightarrow \text{Viscosity of the fluid (0.001 kg/s for fresh water at 20}^\circ\text{C)} \]

The calculations for speeds from 1 to 10 m/s are given in the Table 3 and the Figure 38 shows us the Total Resistance in Newton's (N) for various speeds.

<table>
<thead>
<tr>
<th>Speed (m/s)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reynolds Number (x10^6)</td>
<td>3.00</td>
<td>6.00</td>
<td>9.00</td>
<td>12.00</td>
<td>15.00</td>
<td>18.00</td>
<td>21.00</td>
<td>24.00</td>
<td>27.00</td>
<td>30.00</td>
</tr>
<tr>
<td>Friction Coefficient</td>
<td>0.0029</td>
<td>0.0026</td>
<td>0.0024</td>
<td>0.0022</td>
<td>0.0021</td>
<td>0.0021</td>
<td>0.0020</td>
<td>0.0019</td>
<td>0.0019</td>
<td>0.0019</td>
</tr>
<tr>
<td>Pressure Drag (N)</td>
<td>10.22</td>
<td>40.87</td>
<td>91.96</td>
<td>163.48</td>
<td>255.43</td>
<td>367.82</td>
<td>500.65</td>
<td>653.91</td>
<td>827.61</td>
<td>1021.74</td>
</tr>
<tr>
<td>Friction Drag (N)</td>
<td>2.91</td>
<td>10.12</td>
<td>21.00</td>
<td>35.25</td>
<td>52.67</td>
<td>73.14</td>
<td>96.52</td>
<td>122.75</td>
<td>151.74</td>
<td>183.42</td>
</tr>
<tr>
<td>Total Drag (N)</td>
<td>13.12</td>
<td>50.99</td>
<td>112.96</td>
<td>198.73</td>
<td>308.11</td>
<td>440.96</td>
<td>597.17</td>
<td>778.86</td>
<td>979.34</td>
<td>1205.16</td>
</tr>
<tr>
<td>Effective Power (W)</td>
<td>13.12</td>
<td>101.98</td>
<td>338.88</td>
<td>794.91</td>
<td>1540.54</td>
<td>2645.76</td>
<td>4180.21</td>
<td>6213.27</td>
<td>8814.08</td>
<td>12051.58</td>
</tr>
<tr>
<td>Thrust Power (W)</td>
<td>11.28</td>
<td>87.66</td>
<td>291.29</td>
<td>683.28</td>
<td>1324.19</td>
<td>2274.2</td>
<td>3593.16</td>
<td>5349.71</td>
<td>7576.27</td>
<td>10359.11</td>
</tr>
</tbody>
</table>

Table 3: Calculations for various speeds
From the Total Resistance we can get the power needed to spend in order for the vehicle to have a specific speed. This is the Effective Power ($P_E$) in Watts, meaning the power that effectively moves the vehicle:

$$P_E = R_T V_s$$

The required Thrust ($T$) of a ship exceeds the resistance, because the propeller draws its water along the hull and rudders and thus creates added resistance. This efficiency is called Hull Efficiency ($n_H$) and for the calculation we need (a) the Thrust Deduction ($t$), which is defined as the difference between the total Thrust of $k_p$ number of propellers and the Total Resistance as a fraction of the total Thrust and (b) the Wake Factor ($w$), which is the difference between ship’s speed and advance speed ($V_a$) in front of the propeller, as a fraction of ship’s speed. Combining all the above we can calculate the Thrust Power ($P_T$) that is the power that the propellers output.

$$t = \frac{k_p T - R}{k_p T}$$

$$w = \frac{V_s - V_a}{V_s}$$

$$n_H = \frac{1 - t}{1 - w}$$
In order to calculate the Wake Factor we use the graph of Figure 39, which gives the value of the Wake Factor for different values of the Block Coefficient \((C_b)\). The above relation is applied to Full Bodied Hulls with large Block Coefficients [35] and a single propeller, like the pontoons of our vehicle. The Block Coefficient is defined as:

\[
C_b = \frac{V}{LBD}
\]

where, \(V\) is the volume of the submerged hull, \(B\) is the Beam of the ship and \(D\) the Draft.

Figure 39: Wake Factors for Full Bodied Hulls
Given the Wake Factor, we can calculate the Thrust Deduction using the below relationship for ships with a single propeller:

\[ t = 0.6w \]

On Figure 40 we can see the graph of Thrust Power in Watts (W) per speed in m/s.

![Thrust Power (W)](image)

**Figure 40: Thrust Power (W) per Speed (m/s)**

We know that the propeller, the transmission system and the engine have efficiency less than one, because of the friction loses. For our vehicle, the propeller has 3 blades and 12 inches diameter and the shaft is very short, so the efficiency is almost one. In general the propeller has efficiency \( n_p \) and the motors \( n_m \). With the below equations, we can calculate the input power to the propeller \( (P_P) \), the input power to the transmission system \( (P_S) \), which in this case is the same with \( P_P \) and the electrical power \( (P_m) \) needed for the motors to move the vehicle.

\[
\begin{align*}
P_P &= \frac{P_T}{n_p} \\
P_P &= P_S \\
P_m &= \frac{P_S}{n_m}
\end{align*}
\]

However, for our vehicle the specifications manual of the motors gives us the output power in the propeller, which is 2240 Watts per motor and we don’t
have to do the last three calculations. Given that the above calculations are per shaft, we can conclude that our vehicle can reach the speed of approximately 6 m/s (11.5 knots).
V. ONBOARD IMPLEMENTATION

The goals for this chapter are to describe all the hardware that was used for testing the algorithms and training the system. From the beginning it was known that a laptop must be used to control the autonomous marine vehicle. So, a MacBook Pro with a 15-inch screen, a 2.2Ghz dual-core i7 processor, 4GB 1333Mhz RAM and an AMD Radeon HD 6750M 512MB graphics card was the main computer. This laptop was used to create and test the algorithms for horizon line detection and object localization. It was also used for the calibration of the camera for calculating the distance and the relative heading of the objects. Despite the fact that this laptop had a graphics card, it was not good for Machine Learning applications. More specifically, the Neural Networks that we used were created to perform faster on a computer with NVIDIA graphics card. NVIDIA is a well-know brand that produces CUDA software for graphics accelerators and that way the graphics card can be used for executing convolution and other layers of the Neural Networks efficiently and without occupying the Random Access Memory (RAM). In order to execute all the experiments for network comparison and parameter analysis of the image identification process and train a network that would be used for the final test, we were equipped with a more powerful desktop computer. This computer had a 2.7Ghz quad-core i5 processor, 8GB 2133Mhz RAM and an NVIDIA GT 730 2GB graphics card. In order to use the laptop for real-time prediction during the final test, we had to change the code of one of the Neural Networks so that it can be processed only on CPU.

Another piece of equipment was the camera. We used a web-camera from Logitech (C-920) with maximum resolution [1280x720] and processing capability of 30 frames per second. There was no need for High-Definition resolution, because all the frames were either converted to gray-scale images and then blurring was applied to them for horizon line detection and object localization, or they were downsized to [32x32] for prediction. For both cases all the extra information from an HD-camera would be lost and that is the reason why a medium quality camera was enough. Another advantage of this camera was the
USB connection for which there was not a need for extra power supply. The camera was placed 1.6m above the waterline and it remained on this height during the whole test, because the calibration was implicitly done for this location on the vehicle.

In order to run our software with the laptop and control the vehicle, we used MOOS (Mission Oriented Operating Suite) and more specifically MOOS-IvP (Interval Programming). MOOS is a C++ cross platform for robotic research, created by Oxford Mobile Robotics Group [18]. The main idea is that it has a publish-subscribe architecture. A single MOOSDB serves multiple MOOS applications by essentially handling the mail published and subscribed for by each application. A MOOS community is a collection of applications connected to a single MOOSDB. A similar architecture that contains all the components of MOOS, plus additional modules created by MIT, is called MOOS-IvP [19]. In other words, MOOS-IvP is a superset of MOOS.

The last component of our implementation is the actual vehicle. This vehicle has been used for an autonomous navigation competition in Singapore and for research at SeaGrant of MIT. It is a catamaran type marine vehicle with an upper deck to support the equipment, support cross-bars and two inflatable pontoons contacting the water with approximately 10 feet length and 16.75 inches diameter. These dimensions were used for the resistance calculations in Chapter IVB. Main components of the vehicle are:

1) 2 Motors for propulsion, type Torqeedo Cruise 4.0.
2) 2 Batteries Torqeedo 26VDC, 108Ah, marine Lithium-Ion cells, which are installed on the pontoons, in order to lower the center of gravity of the vehicle.
3) 1 battery Optima 12VDC, 55Ah AGM that provides power to the computers, radio and sensors.
4) Computer subsystem consisting of two Portwell Technology Nano 6060-E3845 single board computers that run REx 4, which is a navigation system based on the MOOS-IvP platform. The operating system for these computers is Ubuntu 12.04.
5) GPS Hemisphere Vector 102, which is a dual antenna that returns latitude, longitude, heading, yaw, pitch and roll.

6) LiDAR Velodyne HDL-32e, which creates a three-dimensional point cloud of radius 80m, as a representation of the environment.

7) Camera, as it was described earlier.

8) Winch subsystem that includes a motor, controller, drive chain and gearbox and it is used to execute specific commands such as "travel at specified speed", "travel to specified position", "emergency stop" and "pause".

9) CTD Sensor (SeaBird Electronics SBE 49 FastCAT) that is connected to the winch and provides real-time salinity, temperature and depth data.

10) "Fish Finder" sonar (Lowrance Mark 4 Chirp fish finder/chartplotter with HST-WSBL depth/temperature transducer) that was used as an inexpensive tool for bathymetry, bottom composition and vegetation mapping.

Figure 41: Rex 4 marine autonomous vehicle
VI. RESULTS

In this chapter we present the work done during the final test. We first explain how we decided to configure our system and then we give some results with videos taken by the vehicle.

Our plan was to set the train/test configuration first and then find the other parameters that will maximize the performance. In order to have not only a sufficient train set for high accuracy, but also a sufficient test set for testing the system as close as possible to the limits, the database was divided into 1500 training images and 300 testing images. Then, we used the analysis presented with Chapter III, in order to pick (a) **Network1** as the best architecture that combines good prediction performance and fast frame processing time for the four categories we wanted to train our system, (b) 64 as minibatch size that is the best for the future work that will be done with the same system, but larger database and more categories, [32x32] as the resolution for the images that achieved 140msec of processing time per frame and an overall capability of processing 6 frames per second.

The results of the experiments that were run with the above configuration showed that the maximum prediction accuracy for the unseen images was 93.66% and the average (global) performance was around 90%. The system responded very well and it remained inactive only for the first epoch. The second epoch gave an approximately 50% of accuracy and at the third iteration the system achieved accuracy more than 70%. 20 epochs were proved to be enough because the final accuracy on the validation set (Chapter IIIE) was on average 99.6%. Out of curiosity, one experiment with 40 epochs was executed and only the maximum accuracy changed slightly (94%). As a consequence, there was no reason to increase the number of epochs.

Global performance is very important but someone has also to check the local performance of the system. Our system was able to predict navy ships, engine boats and buoys with accuracy 96%, 96% and 97% respectively. This performance is very good and goes beyond the performance that we expected
before we started this project. On the contrary, the maximum accuracy for sail boats was 85%. The system was mistakenly identifying approximately 7 out of 75 sail boats as power boats. Someone could say that the sail is a very distinct feature for the sail boats and it should be very difficult to be identified as something else. There is a very logical explanation for this phenomenon. Part of the sail boat images in our database were images of sail boats for which the sail was down and the boats were using the engines for navigation. Therefore, under these circumstances the sail boat is actually an engine boat and it must be considered as one for the international maritime regulations. It could be easier for the system to transfer those images in the database of engine boats and that is a suggestion – point for attention for the future.

The importance of Data Augmentation (Chapter II F) was interpreted and a simple increase of the samples taken from every image was used to improve the performance of the 1500-300 configuration. The number of samples was set to 100 instead of 36 (two times 18 because of the mirroring) and the results were very interesting. The maximum accuracy was 94.33%, which is not much better than the 93.66%. However, the increase of the average accuracy was substantial (from 90% to 92.5%) and this improvement can be translated as an ability of the system to perform well under many different weight adjustments. The system was active from the first epoch, where it achieved accuracy 80% on the test set. The running time per training epoch was three times more (15 minutes), which is acceptable, but the time for predicting one image was two times more and that means the ability of the system to process on a real time application was dropped down to 3-4 frames per second. For that reason we decided to keep the number of samples at 36.

One last observation from the experiments that were run with 1500-300 configuration, was the importance of having the same number of training images for all the categories. In two occasions where the train set had more images for one or two categories and less for the rest, the average performance decreased by 2%. Locally the difference of the performance between the categories was more distinct and the well trained categories achieved 95-98% accuracy in
comparison to the 80% of the untrained categories. That was the reason why we used a C++ script in order to create the training database. The images were distributed in different folders, according to their type and C++ shuffled, picked and labeled them by checking that all four categories had the same number of images for every set.

For the final test we kept the system configuration as it was described in the previous paragraphs and we train the system using all 1800 images of the database. We uploaded videos taken in the Charles River with our vehicle and the system was able to process approximately 5 frames per second, given that each frame returned 3 or 4 objects. More specifically we needed 100-110 milliseconds for the horizon line detection and objects localization and 30 milliseconds for prediction of an object. Each frame was being pre-processed, so that the resolution was decreased to [32x32] and a [3x3] blurring was applied to make the color variation smoother despite the lower resolution. On Figure 42 ten targets form different frames are given as an example. For each object the system returned the array on the right with the probabilities per category. The category with the highest probability is the prediction output of the system. Six out of ten objects were predicted accurately even in case of objects too far from our vehicle. From the other four objects, two engine boats were mistakenly predicted as buoys, mostly because of the fact that the frame had small dimensions to cover the whole object. This problem can be fixed with Machine Learning as it is described in Chapter VII B [5]. Another object was mistakenly predicted as an engine boat instead of buildings. Training the system to identify more than four categories can solve this problem. The last object that represented a sail boat, was predicted as a buoy, mostly because of the distance from our vehicle.
<table>
<thead>
<tr>
<th>Engine</th>
<th>99.66</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navy</td>
<td>0</td>
</tr>
<tr>
<td>Sail</td>
<td>0.34</td>
</tr>
<tr>
<td>Buoy</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engine</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navy</td>
<td>0</td>
</tr>
<tr>
<td>Sail</td>
<td>0</td>
</tr>
<tr>
<td>Buoy</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engine</th>
<th>99.57</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navy</td>
<td>0</td>
</tr>
<tr>
<td>Sail</td>
<td>0.43</td>
</tr>
<tr>
<td>Buoy</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engine</th>
<th>95.08</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navy</td>
<td>0</td>
</tr>
<tr>
<td>Sail</td>
<td>4.92</td>
</tr>
<tr>
<td>Buoy</td>
<td>0</td>
</tr>
<tr>
<td>Engine</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
</tr>
<tr>
<td>Navy</td>
<td>0.007</td>
</tr>
<tr>
<td>Sail</td>
<td>12.95</td>
</tr>
<tr>
<td>Buoy</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engine</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Navy</td>
<td>0.03</td>
</tr>
<tr>
<td>Sail</td>
<td>15.28</td>
</tr>
<tr>
<td>Buoy</td>
<td>0.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engine</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Navy</td>
<td>0</td>
</tr>
<tr>
<td>Sail</td>
<td>0</td>
</tr>
<tr>
<td>Buoy</td>
<td>68.44</td>
</tr>
</tbody>
</table>
Figure 42: Targets from ten frames with probabilities per category

<table>
<thead>
<tr>
<th></th>
<th>Engine</th>
<th>Navy</th>
<th>Sail</th>
<th>Buoy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine</td>
<td>0</td>
<td>0</td>
<td>2.68</td>
<td>97.32</td>
</tr>
<tr>
<td>Navy</td>
<td>0.02</td>
<td>0.14</td>
<td>3.8</td>
<td>96.04</td>
</tr>
</tbody>
</table>
VII. RECOMMENDATIONS

In this last chapter, we will try to evaluate the work done in this project. The efficiency of our system was very high and the results from the final test showed that the autonomous navigation is not only possible, but also in the upcoming future an unmanned marine vehicle will be able to travel around the world and execute the most complicated tasks. This project was a first step and there is room for improvement. We suggest Machine Learning in order not only to improve the horizon line detection and the object localization of our implementation, but also to give an extra ability to our system, that of identifying the different patterns of buoys.

A. IMPROVING HORIZON LINE DETECTION

Machine Learning and especially Convolutional Neural Networks proved to be a very powerful tool, especially when image identification and object categorization is very important. It gave us the opportunity to train a system for four categories of objects (buoys, navy ships, sail boats and engine boats) and predict with accuracy close to 95%. It also gave us the opportunity to introduce new categories and this process was described in Chapter IIIE. The efficiency and the accuracy of a Convolutional Neural Network showed that the system could be trained and used for other important tasks where prediction with high accuracy is needed.

In Chapter IIA, two fast but not 100% accurate methods for horizon line detection were described. The greedy approach (sliced_version) is trying to explore the edges of an image and calculate the location of the horizon line, assuming that this line is the lowest possible horizontal line on an image of a landscape with sea, sky and possibly land. This algorithm is very sensible to the weather conditions (sunny or cloudy day, night) where parameters of the Canny function have to be set accordingly, in order to find the edges of the image accurately. The second and more sophisticated method is to use an optimization algorithm for maximizing the correlation between pixels of the same category.
(sky and water). The problem with this method was the existence of mountains that can mislead the optimization and draw a line that separates the mountains from the sky.

Convolutional Neural Networks can be the most accurate and the most sophisticated solution for the horizon line detection. Two ways of using Machine Learning will be described, one fast and one slow for training but with high probability for 100% accuracy. For the fast approach, we can train a system by creating a database of four categories: landscapes on sunny days, on cloudy days, on rainy days and at night. Then, our system will be able to identify real-time weather conditions without using external forecasting tools and set the parameters for the Canny function automatically.

The second way to use Machine Learning for the horizon line detection is more time and work demanding for the training process, but a well-trained system can give great results not only for weather condition identification as mentioned earlier, but also drawing a line accurately for both sky-water and sky-land-water landscapes. Again, we have to create a database for four image categories of landscapes on different weather and light conditions. This time we have to draw the horizon lines for every image of the train set, using a very distinct color, such as red or yellow. With this database our system will be trained to draw the horizon line using the same color on the unseen images. On top of that, if the database contains images where the horizon line is extremely inclined, for example when a boat is navigating on higher sea-states with more intense waves, our system will be able to identify those cases and draw the horizon line accurately with higher probability than the sliced_version and the optimization methods, when there are mountains on the landscape.

B. IMPROVING OBJECT LOCALIZATION

The introduction to this chapter could be the same as the one from the previous chapter. Machine Learning is suggested to improve also the efficiency of the object localization process. In Chapter IIIE Otsu's Method was used to separate the objects from the background and use them for identification with
Machine Learning. This method is very fast and can be used in a real-time application but for complex landscapes, like the one where an autonomous marine vehicle can operate, drastic changes in colors (sky-mountains-sea) can mislead the results. In fact the presence of mountain is the reason that the system won’t be able to identify small objects. However, in order to defend the work done in this project, we can say that (1) the system can be configured so that to use the horizon line and narrow down the area were important objects might exist and (2) the system can be configured so that to localize parts of the land on the background. With the first claim, it is justified that for this type of landscapes Otsu’s Method is applied on this part of the image, where the majority of the dark colors are gathered. That means that the method is not influenced by the lighter colors of the sky and the sea and it can more accurately assign a threshold that distinguishes the objects from the rocky background. With the second claim, it is justified that even in the case where the rocky background is not completely removed, the system will put a frame around parts of the land and it will send those frames for image identification. In that step, a well-trained system will be able to identify the important features even when the cropped image hasn’t being zoomed in an object. This is how the Convolutional Neural Networks works and the overall system’s efficiency is increased. However, our goal is to always find a better solution and Machine Learning can be the answer.

Recent efforts [5] are using Convolutional Neural Networks not only to identify and categorize object, but also to draw lines and frames around the identified objects. The system is trained with a database where all the objects have frames of distinguished color around them (for example red for navy ships and blue for buoys). Then, with a more complicated code, than the one used for this project, the system is able to identify multiple objects on an image, draw a frame with their specific colors and export the center of area for all the frames. Especially the last attribute of the system is very important, as described in Chapter IV, for approximating the distance and the relative heading of the object using only a camera and in case of a ship, for calculating the probability of collision with the autonomous vehicle.
C. IDENTIFYING SHAPES ON Buoys

The last suggestion is as important as the previous ones. With this suggestion our goal is not to improve the efficiency of the algorithms used for this project, but to expand the opportunities of our system. In case of buoys and maybe for other categories that will be introduced eventually, such as lighthouses, it is not enough to say that for example two out of three objects are buoys. It is very important to identify the color of the buoy and the pattern on each buoy, a pattern that gives the necessary information for safe and restricted navigation. A first approach for identifying the color of the buoy can be done by analyzing the FFT of an image. This process is good for specifying if the buoy is a red one (right side entering the port), a green one (left side entering the port), a white one (information for regulations and restrictions) or a yellow one (scientific). However, the system should be able to identify the patterns on the white buoys.

Identifying the pattern of such a small object far away from your boat is a very tricky task. We must assume that the system doesn’t have to be very efficient at least not until the buoy is close enough. In other words, we can design a system that triggers the Neural Network for identifying the pattern at some point and only if the buoy is a white one. Training the system is almost the same procedure as the one used for our project and the ones suggested in the previous chapters. We have to create a database with four categories in order to cover all the possible patterns of white buoys and three more to cover all the possible colors. We have to pay attention when we create the database and the images from each buoy should be taken from all the possible angles (360º) and all the possible distances. The network that will be used, should very intense with many neurons and the images should have a much higher resolution than [32x32], so that the system will be able to learn the features of the shapes (curves for a circle or straight lines for triangle and square) and the shape of the letters and numbers. The above necessities will drive the running time of prediction to be surprisingly high (one or half frame per second), but if we design the system to run multiple tasks at the same time, it will be able to keep...
processing frames and identifying multiple objects with higher frequency, while it is waiting for the prediction of a white buoy.
The goal of this project was to create a basic structure that would give to a surface marine vehicle the ability to operate autonomously and locate, identify and use accordingly the objects that create the environment of operations. Machine Learning gave us the opportunity to achieve prediction performance of more than 93% for a system trained to identify four different categories of objects and with only 450 images on each category. In order to make the system as more complete as possible, two algorithms for horizon line detection and background subtraction were used, in order for the system to be able to localize the objects around it.

The very good performance of Machine Learning techniques and specifically the Convolutional Neural Networks, gives us the motivation to create a larger database for more generalized training that will give the opportunity to the system to predict with better performance not only four categories, but even more. This way we can cover all the possible objects that an autonomous marine vehicle will find during an operation. We can also use Machine Learning to improve the process of the horizon line detection, a process that is very important for airplanes too.
IX. REFERENCES

2. A Threshold Selection Method from gray-level histograms, Nobuyuki Otsu (1979)
7. Understanding Convolutional Neural Networks
17.http://www.image-net.org/


29. Embodiments of Mind, Warren McCulloch (1965)


31. A Logical Calculus of Ideas Immanent in Nervous Activity, Warren McCulloch et Walter Pitts (1943)

32. Some Intuition about Activation Functions in Feed-Forward Neural Networks, Alexandres Dalyac (2014)


34. The Organization of Behavior, Donald Hebb (1949)