DESIGN OF AN UNDERWATER PIPE INSPECTION ROBOT

A Project Report

submitted by

VINAYAK S P

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
MAY 2017

THESIS CERTIFICATE

This is to certify that the thesis **DESIGN OF AN UNDERWATER PIPE INSPECTION ROBOT**, submitted by **Vinayak S P**, to the Indian Institute of Technology, Madras for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of the thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr K.SRIDHARAN
Research Guide
Professor
Dept. of Electrical Engineering
IIT MADRAS, 600036

Place: Chennai

Date : 12th May, 2017

ACKNOWLEDGMENTS

It gives me great pleasure in expressing my sincere and heartfelt gratitude to my project guide Dr K. Sridharan for his excellent guidance, motivation and constant support throughout my project. I consider myself extremely fortunate to have had a chance to work under his supervision. It has been a very learning and enjoyable experience to work under him.

My heartfelt appreciation to my team mate D Aravind and lab mates Bhaskar, Yashraj and Sharad for spending their invaluable time with me in discussing about the project and answering my queries.

I am immensely thankful to Dr Tommy Huynh, Associate Lecturer in Department of Engineering, La Trobe University for providing me with real time videos of underwater pipe inspection which were a valuable resource to test the algorithm.

Words cannot express how grateful I am to my parents and sister for their constant support and encouragement and to the Lordships Sri Radha Krishna for Their blessings due to which I was able to think in the right direction and complete the project successfully.

ABSTRACT

An Autonomous Underwater Vehicle(AUV) is a robot which travels underwater without requiring input from an operator. Most AUVs are equipped with a video camera and lights. Additional equipment is commonly added to expand the vehicle's capabilities.

Design and construction of an AUV has numerous challenges to offer starting with water proofing, static and hydrodynamic stability, propulsion, power consumption and control and navigation. A blend of technologies like image processing, remote communication and embedded systems are employed in the vehicle.

This thesis examines the design of the AUV which can be grouped in three verticals - mechanical, electrical and the crack detection algorithm.
The AUV captures images at certain intervals using a camera and they are recognized for presence of crack using an on-board microcontroller.

The crack detection is achieved by a machine learning algorithm called logistic regression. Towards the end, a study was made to detect cracks using image processing library called OpenCV(Open Source Compute Vision).

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Autonomous robots perform a variety of tasks including surveillance, rescue, cleaning and searching for a specific item. They are generally equipped with a variety of sensors and a high level computer for high level tasks besides possibly a microcontroller for low-level tasks.

1.1 MOTIVATION

Power plants use mild steel underground pipelines that carry makeup water in and out of the condenser as shown in the figure 1.

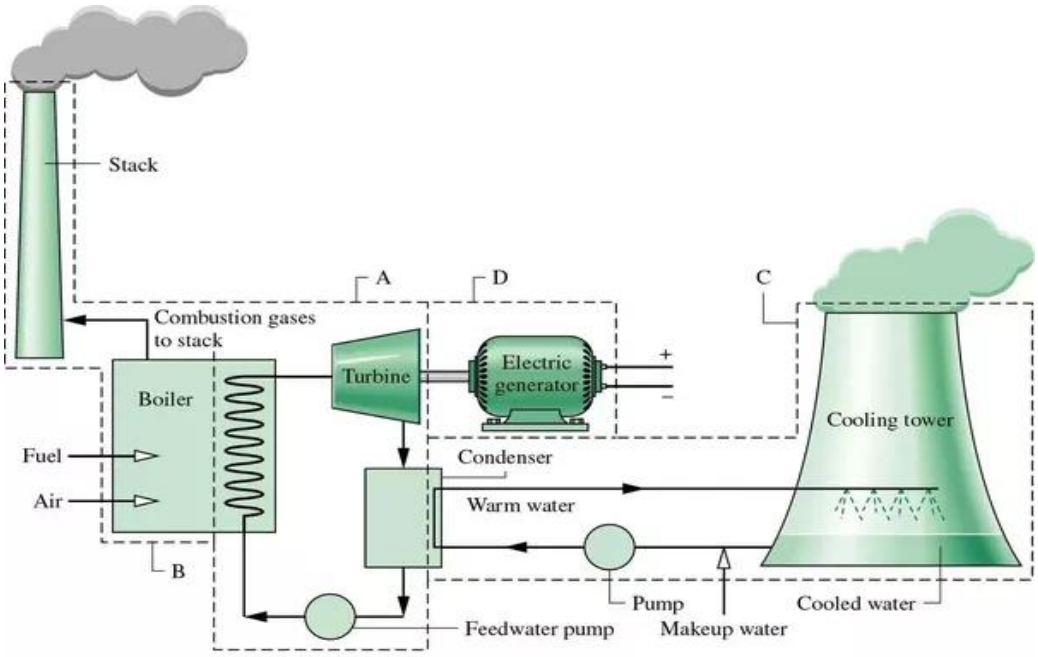

Figure 1 A schematic of Power Plant

This makeup water flows through the walls of the condenser and cools the steam inside it by heat exchange.
The visit to National Thermal Power Corporation Ltd., Ramagundam gave us an idea of the section of pipeline to be inspected as shown in Figure 2.

Figure 2 A Section of Pipeline to be inspected at NTPC

The above figure depicts a mild steel pipeline covered with concrete which primarily runs underground spanning over a distance of more than 5km in the power plant.

These mild steel pipelines of the plant being several decades old, get eroded by constant flow of water throughout the year. This leads to formation of cracks through which the water seeps out of the pipe. With time, the size of the crack would increase thereby leading to reduction of required flow and permanent damage to pipeline.

Previous attempts of manual labour to detect such cracks had proven to be risky, time consuming, ineffective and costly. The proposed Autonomous Underwater Vehicle was designed for NTPC to inspect the pipelines without human intervention.

1.2 OBJECTIVES

The objectives of the thesis are as follows:

1. To design and construct a prototype of an Autonomous Underwater Vehicle.

2. To implement an efficient algorithm to detect cracks in images.

3. To optimise memory consumption by intelligently storing the images captured by AUV inside the pipeline only if it is recognized as a crack.

1.3 CONTRIBUTIONS OF THE THESIS

1. A waterproofed and mechanically stable prototype of the AUV.

2. Principal component analysis based dimensionality reduction technique for image recognition.

3. Study on a Compute Vision algorithm for detecting cracks.

1.4 CHALLENGES IN CONSTRUCTING THE AUV

Water sealing is one of the major challenges while constructing underwater Robots. The payload that consists of microcontrollers and battery must be protected from water.

There are various machine learning techniques such as logistic regression, neural networks, support vector machine, etc., that can be employed in image recognition. It is therefore a challenge to choose the right technique which is fast, efficient in terms of accuracy and has minimal computational requirements.

1.5 ORGANIZATION OF THE THESIS

Next chapter presents the details of the mechanical design of the Autonomous Underwater Vehicle.

Chapter 3 presents the electrical components utilised on board the Autonomous Underwater Vehicle.

In chapter 4, details of the logistic regression method along with principal component analysis for dimensionality reduction are presented.

In chapter 5, a brief study on Boost and Adaboost agorithms used in Compute Vision are also presented.

Chapter 6 summarises the thesis and discusses the future directions of work.

CHAPTER 2

MECHANICAL DESIGN

2.1 INTRODUCTION

In this section, we present the various components used to build the mechanical structure and their advantages. The design ideas originate from the work of [3].

We begin with the description of the Pressure Hull shape. We then present the free body diagram of the AUV and analyse the forces acting on it underwater. The design's preliminary drawings in Google Sketchup tool are shown and finally the current design is shown.

2.2 DESIGN OF PRESSURE HULL

Hull provides a waterproof enclosure at atmospheric pressure for the electronic payloads of the AUV. Generally, several factors influence the design of the hull.

A cylindrical shaped hull has been incepted in the design because of the following reasons:

1) It is a good structure to resist hydrostatic pressure.
2) It has a good hydrodynamic form which helps in reducing the drag on the vehicle and,
3) It provides sufficient space for placement of electronics.

The hull material should have good resistance to corrosion, have a high strength to weight ratio and must be affordable. Considering above factors, acrylic plastic is chosen as the material for the hull of the AUV. The added advantage of acrylic is that it is transparent and allows monitoring of the electronic components inside.

2.3 MECHANICS IN WATER

To stabilize the mechanical structure underwater, the centre of buoyancy should be above the centre of mass. This would ensure that the rolling motion is inhibited.
Consider a body fully immersed in water as shown in the Figure 3. It is stable when C_M and C_B are aligned and unstable when C_M and C_B are not aligned. Therefore, when the body is unstable, a righting moment acts to restore the body in its stable position.

$$R_M = (F_M + F_B) * G_Z/2$$
$$R_M = (1/2) * (F_M + F_B) * (G_M \sin(ø))$$

where F_M is the weight, F_B is the buoyancy, G_M is the metacentric height and ø is the roll angle. So the given expression implies that greater the R_M, i.e. greater the metacentric height, greater is the stability.
Since the supporting structure was made from pvc tubes, to achieve the above principle extra weight (here we used sand) in the pipe was added.

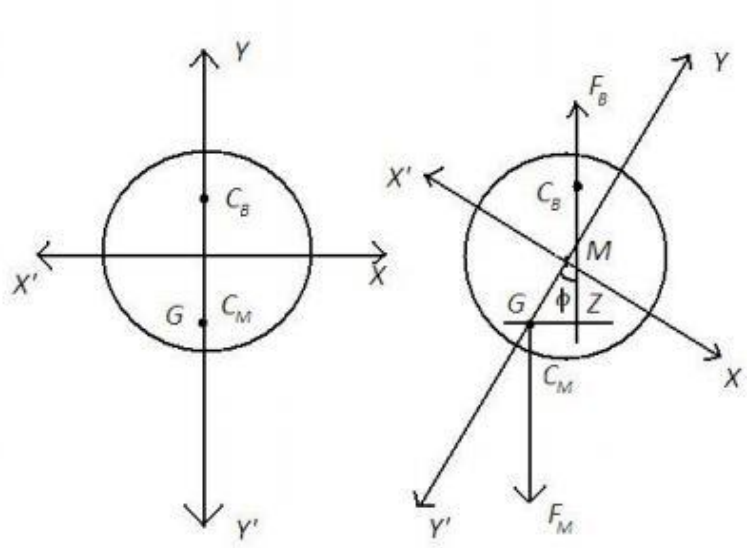

Figure 3 Free body Diagram

2.4 PRELIMINARY DRAWING IN SKETCH UP

Figures 4 and 5 are the Google Sketchup Drawings depicting the structure of the Autonomous Underwater Vehicle and placement of various components on the AUV.

Internal Battery, Processor, Camera and Electronic Speed Controllers(ESC) are placed inside the waterproofed acrylic hull.
The cords coming out of the AUV are the external power supply that is for emergency purposes and fibre optic cable for live camera video feed.

The best choice for the base would be an aluminium exoskeleton because of its light weight. Ultrasonic sensors are placed at the front to avoid obstacles inside the pipelines.

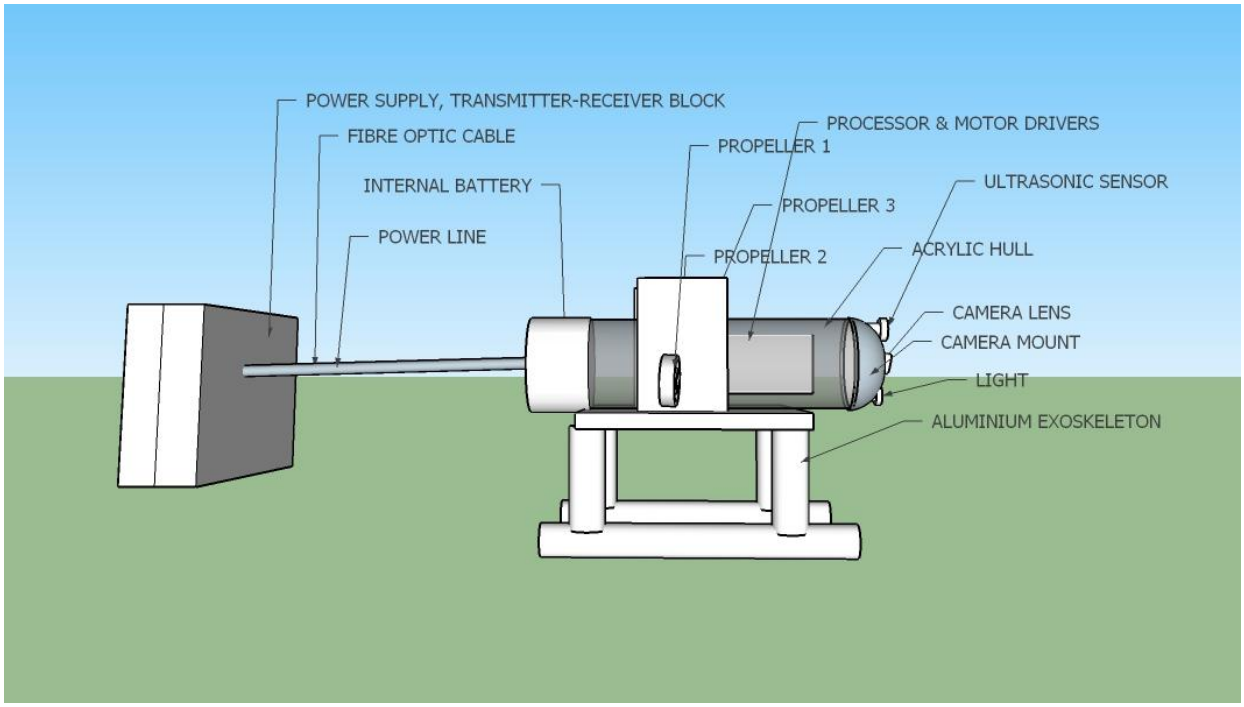

Figure 4: Side View of Design

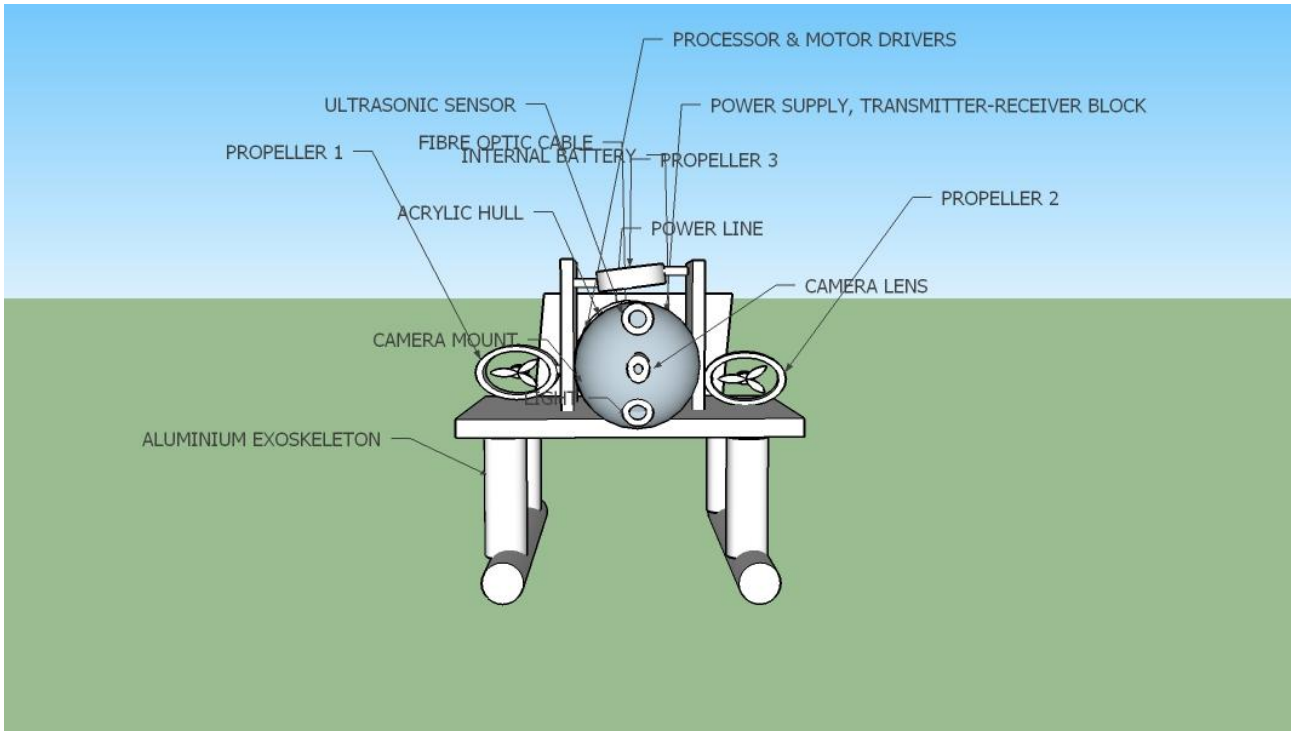

Figure 5: Front View of Design

2.5 THE CURRENT DESIGN

The following Figures 6 and 7 show the AUV that we constructed. The base of the AUV on which the hull and propellers rest is an acrylic board. Couplers are used to join the motors to the propellers. The propellers used were pc fans. The couplers and the metal stand to hold the motors were machined in the college laboratory.

PVC pipes were chosen to build the frame because of its low cost compared to aluminium. Proper adhesives were used to make the body rigid.

The sealing was achieved using a 90mm MTA(male threaded adapter pipe)-FTA(female threaded adapter pipe).

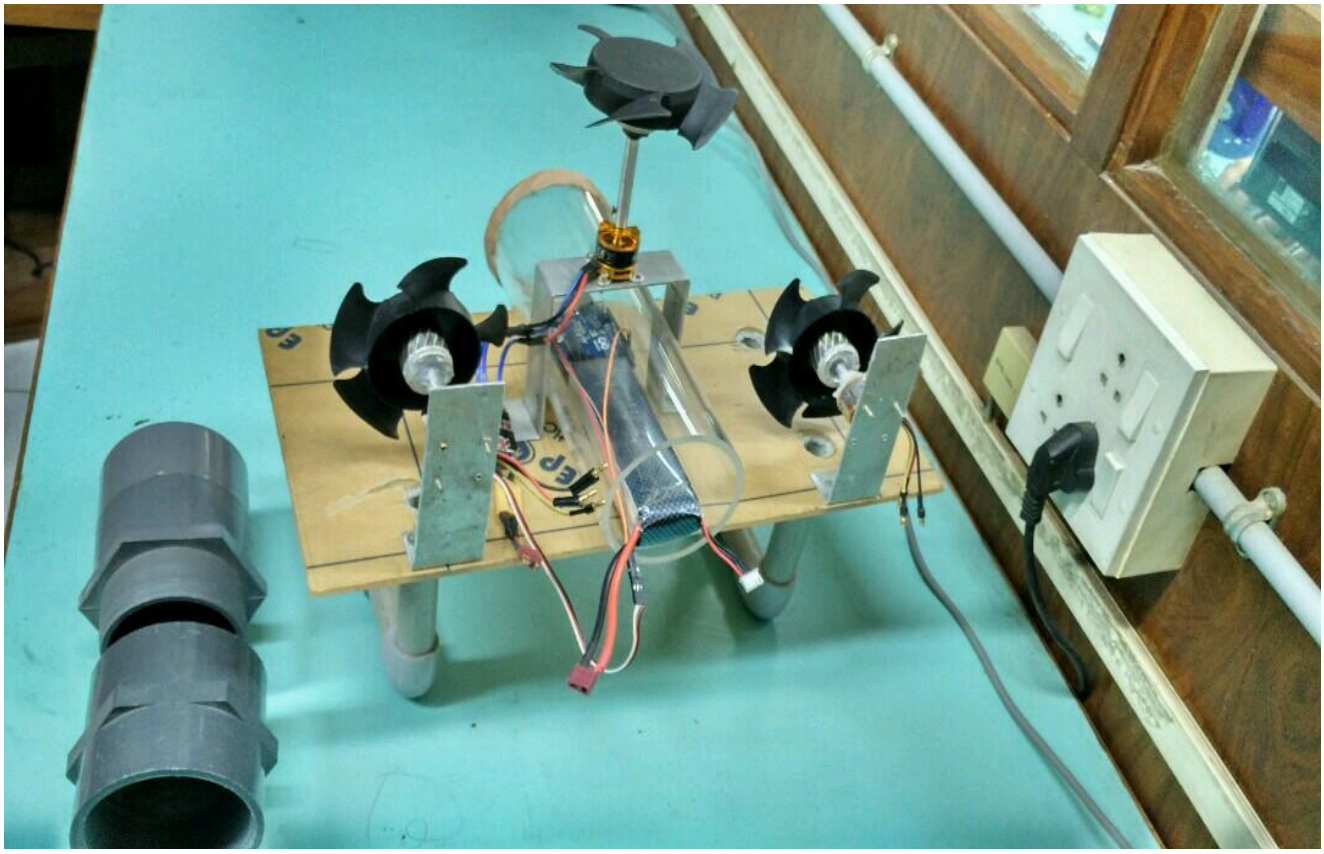

Figure 6: The AUV Design

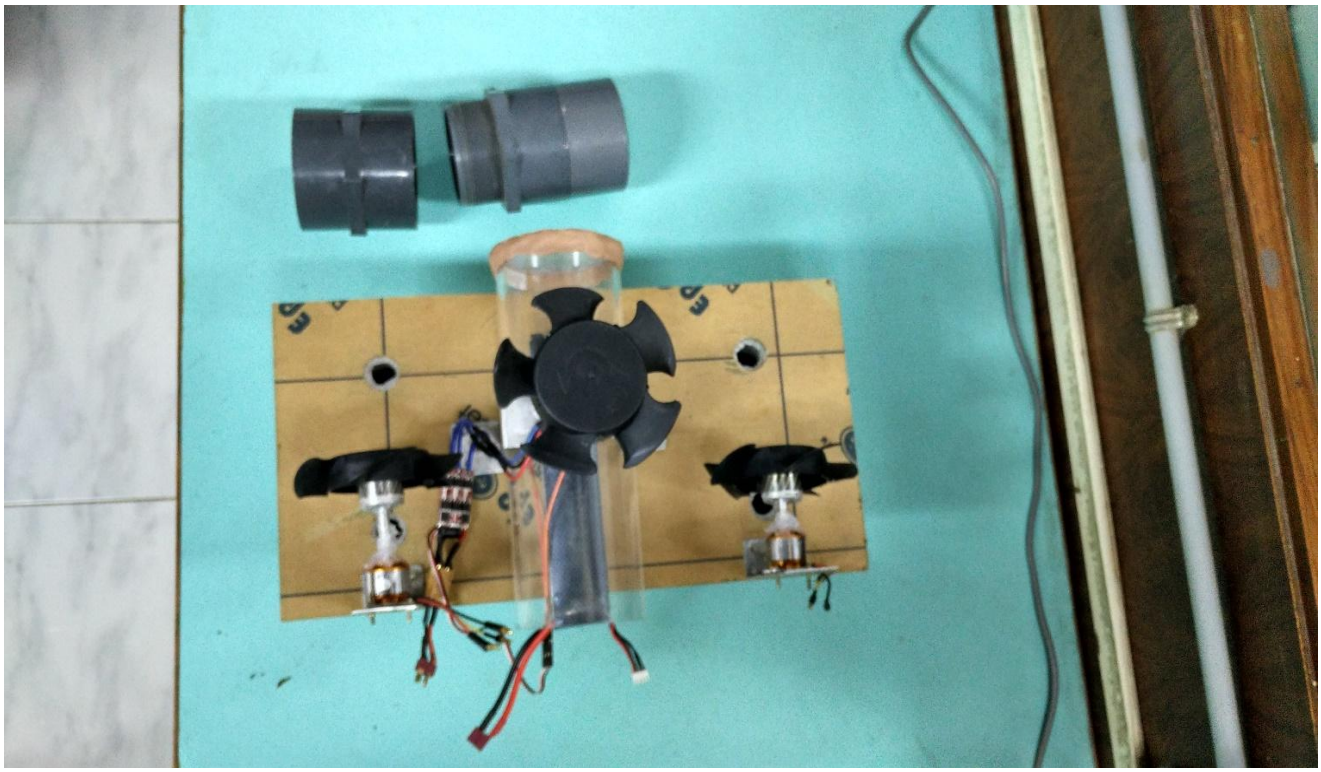

Figure 7: Top View of the AUV

2.6 SUMMARY

The proposed design for the AUV uses a cylindrical acrylic hull supported by a PVC frame. The sealing was accomplished by a MTA(male threaded adapter pipe)-FTA(female threaded adapter pipe) combination along with a dummy made from pvc to facilitate easy detachability.

Provisions were made in the body to ensure that weight of the pvc frame can be adjusted to ensure the centre of mass remains below the centre of buoyancy to avoid rolling motion.

The next chapter presents the details of the electrical and electronic components utilised by the AUV for motion and image recognition.

CHAPTER 3

ELECTRICAL DESIGN

3.1 INTRODUCTION

The electrical design involves the choice of motors and their corresponding Electronic Speed Controllers (ESCs), the calibration and speed control of the motors, the choice of the battery to power the motors and the on board microcontrollers.
The propulsion system consists of three brushless dc motors and their corresponding Electronic Speed Controllers (ESCs). The ESCs are calibrated using the code shown in Appendix A. The speed control code (Appendix A) adjusts the duty cycle of the Pulse width Modulated(PWM) wave that is fed into the ESCs thereby controlling the speed of the motors.

The following sections describes the components chosen and the speed control method in detail.

3.2 BATTERY

Li-Po battery is used to power the Autonomous Underwater Vehicle. Because of its higher power-to-weight ratio, it can drive all the three brushless dc motors and can also be accommodated easily inside the hull.
But, the disadvantages of using Li-Po battery is that powering the microcontroller is risky since it has high discharge rate. Also, extra protection from water is required for the Li-Po battery apart from the sealing of the hull due to it being prone to hazards.

The power management system of the AUV has been designed for an endurance of 15 minutes at maximum continuous load using 4200mAh, 3-cell, and 11.1V DC Lithium-Polymer (Li-Po) batteries.

3.3 THRUSTERS

Three 1200kv dc brushless motors are used as thrusters (Figure 8). The brushless dc motors work smoothly underwater because of the insulation on the armature winding. These three thrusters provide three degrees of freedom which are yaw, ballast/heave and sway motion.

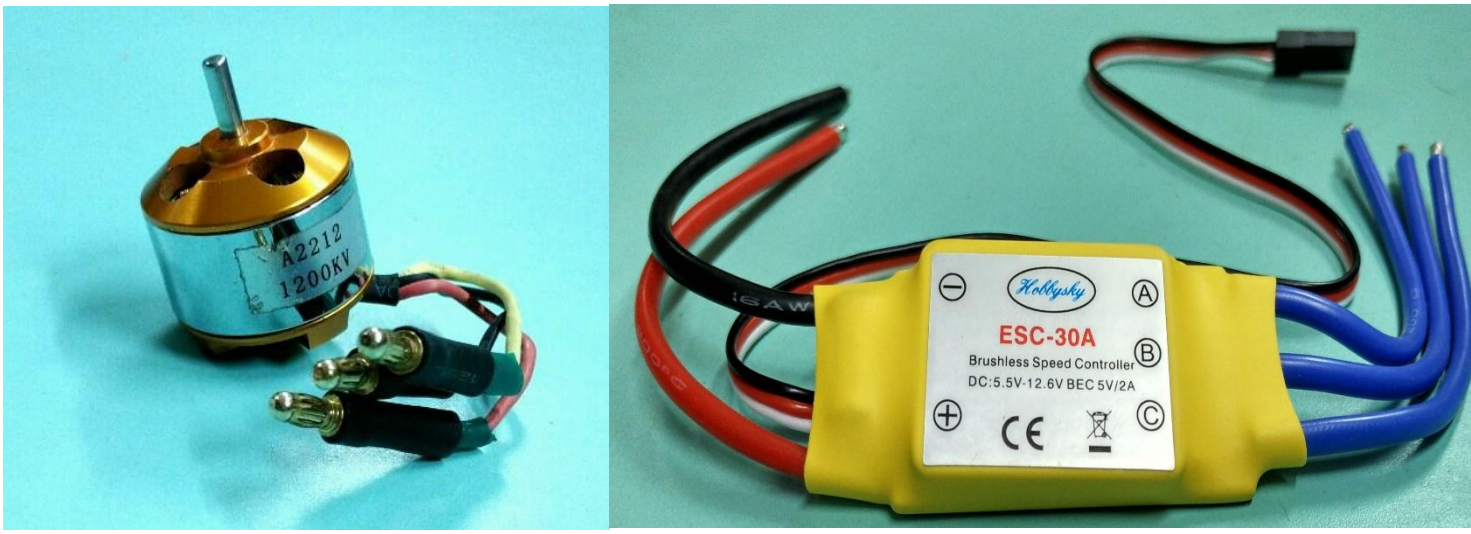

Figure 8: Brushless DC Motor and ESC

The pwm signal from the microcontroller (Arduino Uno) are fed to the ESCs which in-turn regulate the speed the motors.

3.4 ARDUINO UNO

Arduino Uno board was used to calibrate the ESCs with pwm signals. The ESC requires 20ms time period pwm waveforms from the microcontroller with the pulse width ranging between 1000us to 2000us as shown in Figure 9.

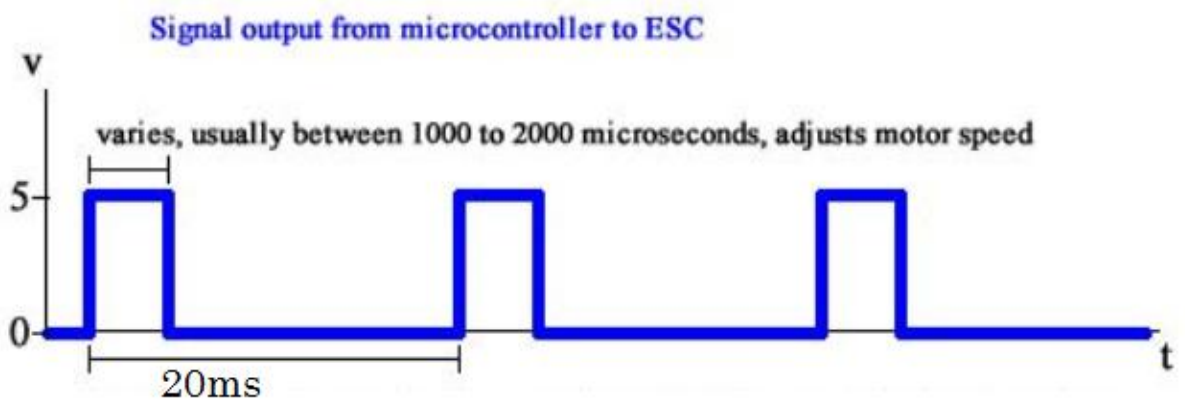

Figure 9: PWM Signal Duration

The ESC was calibrated between 1000us(0 rpm) upto 1500us(full speed).
Arduino Uno was also used to generate pwm signals for ESCs to control the rpm of the motors. After several experiments, for a pwm width of 1050us minimum speed was achieved and was used for running the AUV. The calibration and speed control codes used are shown in Appendix A.

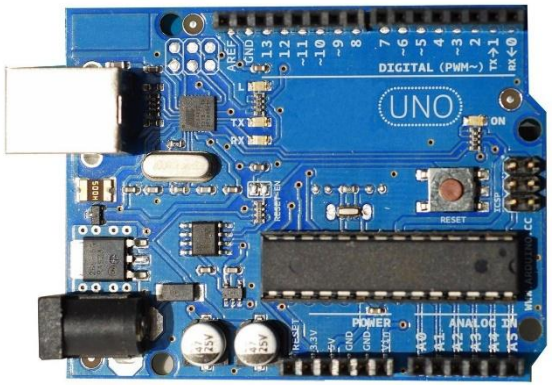

Figure 10: Arduino UNO

3.5 CAMERA

Our AUV uses a waterproof GoPro HERO+ camera(Figure 11) which is capable of capturing video feed with 1080p resolution. It has an ultra wide field of view and also supports a video resolution of 720p. It has a USB 3.0 interface for transferring the video feed. If we are shooting in 720p, there is a Super-View option to capture video

with an even wider field of view. The camera provides a consistent frame rate of 60 fps.
The captured images are stored in a waterproof class-10 SD Card with 32 GB capacity with write speed of 30Mbps.

Figure 11 : GoPro Hero+ Camera

3.6 BEAGLEBONE

Beaglebone Black is a System-on-Chip (SoC) developed by Texas Instruments Inc. It has an ARM Cortex A8 CPU with a memory of 256MB and a RAM of 256MB.
It runs on Debian based Linux operating system and supports the necessary libraries and wheels for Python. Since it has USB and HDMI ports also , it can be interfaced with the GoPro Camera. The images captured by the camera are accessed by Beaglebone and the required crack detection is performed on board. Those images that do not have crack can be erased from camera SD card using Beaglebone.

3.7 SUMMARY

The electrical components used in the Autonomous Underwater Vehicle are presented in this chapter. The procedure to calibrate the ESCs and control the speed of the motors are also described.

The role of the two microcontrollers(Arduino and Beaglebone) in calibration , speed control and image recognition is also elaborated.

The next chapter presents the details of principal component analysis and logistic regression methods along with experimental results of various tests conducted in image recognition.

CHAPTER 4

CRACK DETECTION USING LOGISTIC REGRESSION

4.1 INTRODUCTION

The problem of predicting an image for presence of crack or no crack is a binary classification problem.
We used logistic regression, which is one of the most popular and most widely used learning algorithms today for binary classification. It is a supervised learning method for classifying data into discrete outcomes.

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.
Example:
(a) Regression - Given a picture of a person, we have to predict their age on the basis of the given picture
(b) Classification - Given a patient with a tumour, we have to predict whether the tumour is malignant or benign.

Logistic regression includes both regression and classification.

4.2 BINARY CLASSIFICATION

In binary classification problem the variable that we're trying to predict is a variable ‘y’ that we can think of as taking on two values either zero or one i.e., an image has a crack or no crack, a tumour is malignant or benign.
The variable with value zero is called the negative class and the and variable with value one is the positive class. So, we infer ‘0’ as the absence of crack, and ‘1’, the positive class with the presence of crack. The assignment of the two classes - positive and negative to zero and one is arbitrary.

4.3 NOTATIONS

n = number of features
m = number of training examples
$X \in R^{mxn+1}$
$x^{(i)}$ = input for the i^{th} training example
$x_j^{(i)}$ = value of the feature j in the i^{th} training example
where, $0 < i < m$, $0 < j < n$
$\theta \in R^{nxnum\_class}$ = weight matrix, num_class = 2 for binary class
$y \in R^m$ = A *m* -dimensional vector representing the y-values for each example.

Each training example is a n by n pixel grayscale image. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The n by n grid of pixels is unrolled into an n^2 -dimensional vector which are the rows of vector X shown:

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

4.4 HYPOTHESIS FUNCTION

Hypothesis function is the function we're going to use to represent our hypothesis when we have a classification problem.
We would like our logistic regression to output values that are between 0 and 1. So we use a hypothesis function that satisfies this property, that is, our predictions are between 0 and 1.

For this purpose, we define our hypothesis function as :

$h_\theta(x^{(i)}) = g(x^{(i)} * \theta)$ where g is called the sigmoid function or, the logistic function

$$g(z) = \frac{1}{1 + e^{-z}}$$

where $z = x^{(i)} * \theta$

The following image shows what the sigmoid function looks like:

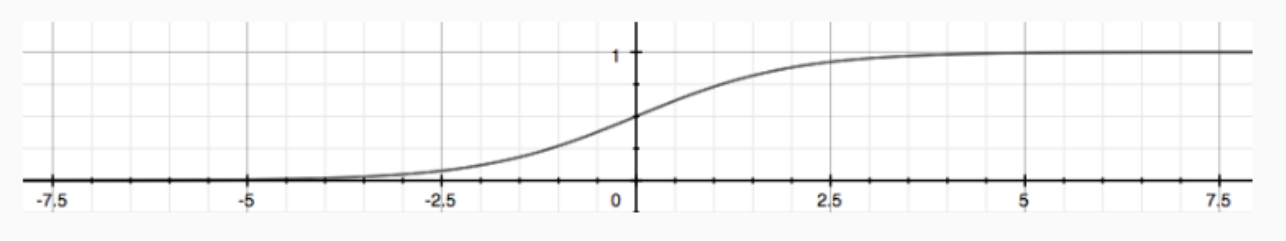

Figure 12 : Sigmoid Function

Now, we define the cost function as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] .$$

For y = 1, the parameters used in $h_\theta(x^{(i)})$ are θ_{nx1} and they are optimized by gradient descent method to maximize
$h_\theta(x^{(i)})$ which in turn minimizes J(θ).

Similarly, for y = 0, the parameters used in $h_\theta(x^{(i)})$ are θ_{nx2} and they are optimized to maximize (1- $h_\theta(x^{(i)})$) which in turn minimizes J(θ).

This above cost function was chosen to make the optimization problem to be a convex function. This helps in finding the global minima easily since there are no local minima in convex functions.

4.5 GRADIENT DESCENT

Given the cost function, in order to fit the parameters, we need to find the parameters θ that minimize J(θ).

The way we're going to minimize the cost function is using gradient descent.
If we want to minimize it as a function of θ, our usual template for graded descent where we repeatedly update each parameter is by taking θ_j , updating it as itself minus alpha(learning rate) times the derivative term as shown:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

The above term is simplified as :

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

α = The learning rate. It should not be very high since it would lead to divergence.

4.6 THE PROBLEM OF UNDERFIT AND OVERFIT

Under-fitting, or high bias, is when the form of our hypothesis function ‘h’ maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features.
At the other extreme, overfitting, or high variance, is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated

function that creates a lot of unnecessary curves and angles unrelated to the data.

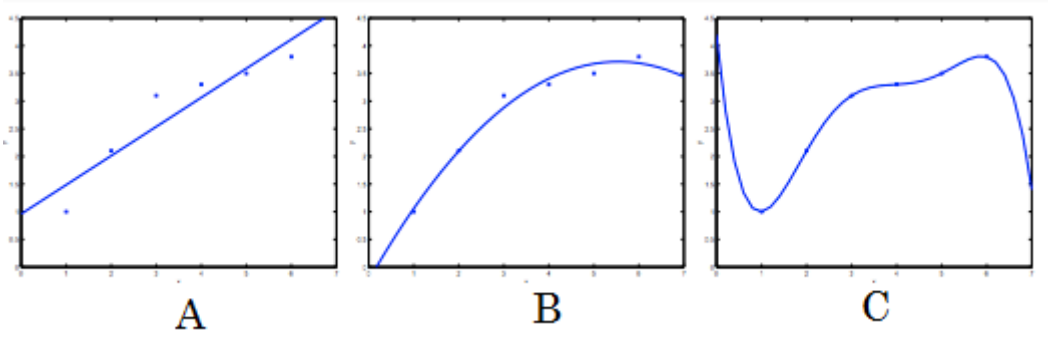

Figure 13 : A – Underfit Curve, B – Perfect Fit, C – Overfit Curve

4.7 REGULARIZATION

When the number of features are high, the learned hypothesis gives parameters in such a way that J(θ) is zero. But this tries too hard to fit the training set and fails to provide a general solution that is, apply to new examples.

There are two ways to deal with over-fitting :

1) Reduce number of features by manually selecting which features to keep.
   But, in reducing the number of features we lose some information so, we need to select those features which minimize data loss.

2) Regularization
   In this method, we keep all features, but reduce magnitude of parameters θ.
   This works well when we have a lot of features, each of which contributes a bit to predicting y.

In regularization, small values for parameters corresponds to a simpler hypothesis(we effectively get rid of some of the terms). A simpler hypothesis is less prone to overfitting.

With regularization, we modify the cost function to shrink all the parameters by adding a term at the end as shown below:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2.$$

where, λ is the regularization parameter

λ controls a trade off between our two goals which are to fit the training set well and also keep parameters small.

Now, the updated partial derivative of regularized logistic regression cost for θ_j is defined as :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \quad \text{for } j \geq 1$$

After computing the J(θ) and the gradient values, we used an advanced function minimization routine called ***fmincg.***

It takes the cost function, gradients, lambda, initial parameters θ, y and number of iterations as inputs and optimizes the cost function and gives the optimal parameters (θ_{opt}) as output.

This routine also picks a good learning rate, and so the optimization ends up converging much faster than gradient descent.

It lets the algorithms scale much better to very large machine learning problems, such as if we had a very large number of features.

4.8 DECISION BOUNDARY

After training the parameters over all training examples, they are optimized in such a way that a decision boundary is formed where y = 1 class is separated from y = 0 by a hyperplane $X*\theta = 0$ as shown below:

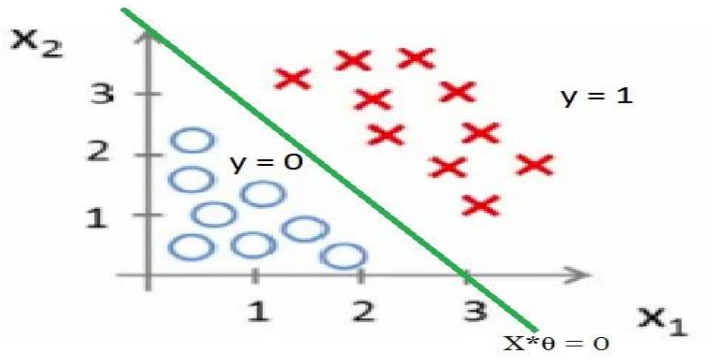

Figure 14 : Decision Boundary formed after training

So, the training examples are separated in such a way that for y = 1, $X*\theta \geq 0$ and for y = 0, $X*\theta < 0$.

4.9 PREDICTION

Consider the following:
The test set (X_test) is of dimension : R^{pxn},
The optimal weight matrix obtained after training is θ_{opt} ,
and
$h_\theta(x^{(i)}) = g(X\_test*\theta_{opt})$ is a R^{px2} matrix with each row representing one training example and each column representing its probability of belonging to the classes defined by y.

The maximum probability (P) in each row is selected and if $P \geq 0.5$, the test example is classified as belonging to class y = 1 else,
the test example is classified as belonging to the class y = 0.

4.10 EXPERIMENTAL RESULTS

We obtained 309 images with cracks and 148 images without cracks from Google to test our algorithm. All of the images were converted to grayscale beforehand. A few samples of images used in training and test set are as shown :

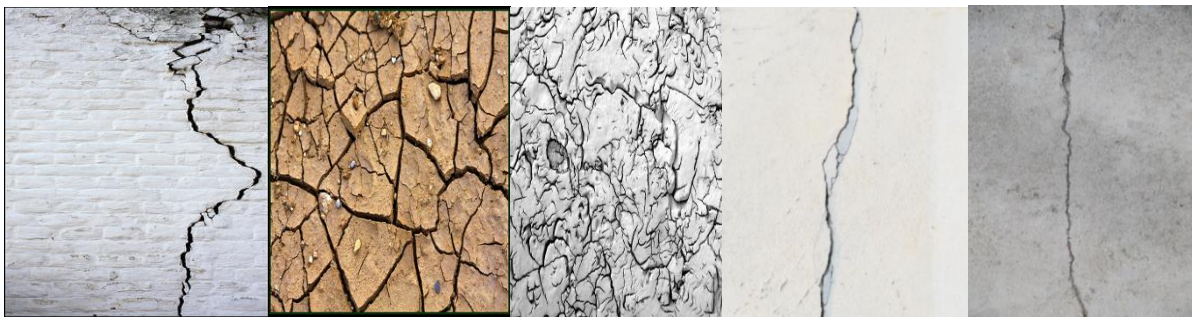

Figure 15

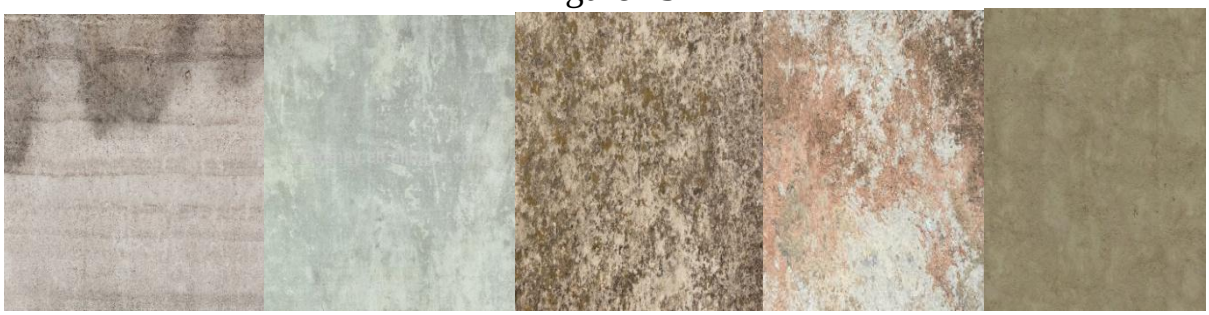

Figure 16

Figure 15 and Figure 16 show the images used in training set for images with cracks(Class 1) and without cracks(Class 0) respectively.

Figure 17

Figure 18

Figure 17 and Figure 18 show the images in test set

4.11 CHOOSING A CROP SIZE FOR IMAGE

The primary requirement for logistic Regression is that all the images used for training and test be of same dimensions.
Since the image data obtained were all of different dimensions we decided to crop the images into smaller images to increase the number of images to increase data which is a favourable trait in machine learning.

Let us assume that all images were split into 100*100 pixel images. Then each pixel is considered to be a feature which will be modelled. Thus there are 10000 features per cropped image and logistic regression works best when the number of training examples per training class are more than or as large as the number of features. In situations where collections of such an enormous amount of images is very difficult, splitting the image is the best alternative.
The new training set now contains 8800 images with cracks and 10019 images without cracks. The same was done for test set images.

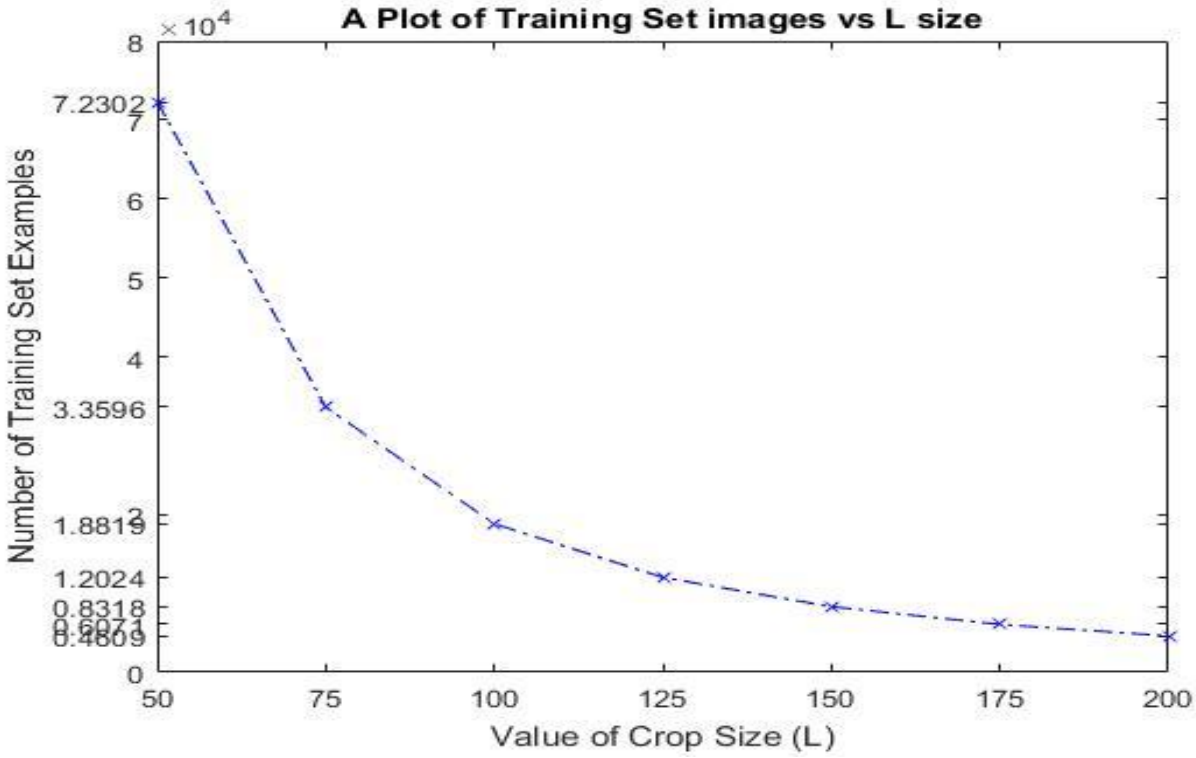

Figure 19

Logistic regression was run on this training data and the prediction accuracies obtained on the test set are as shown:

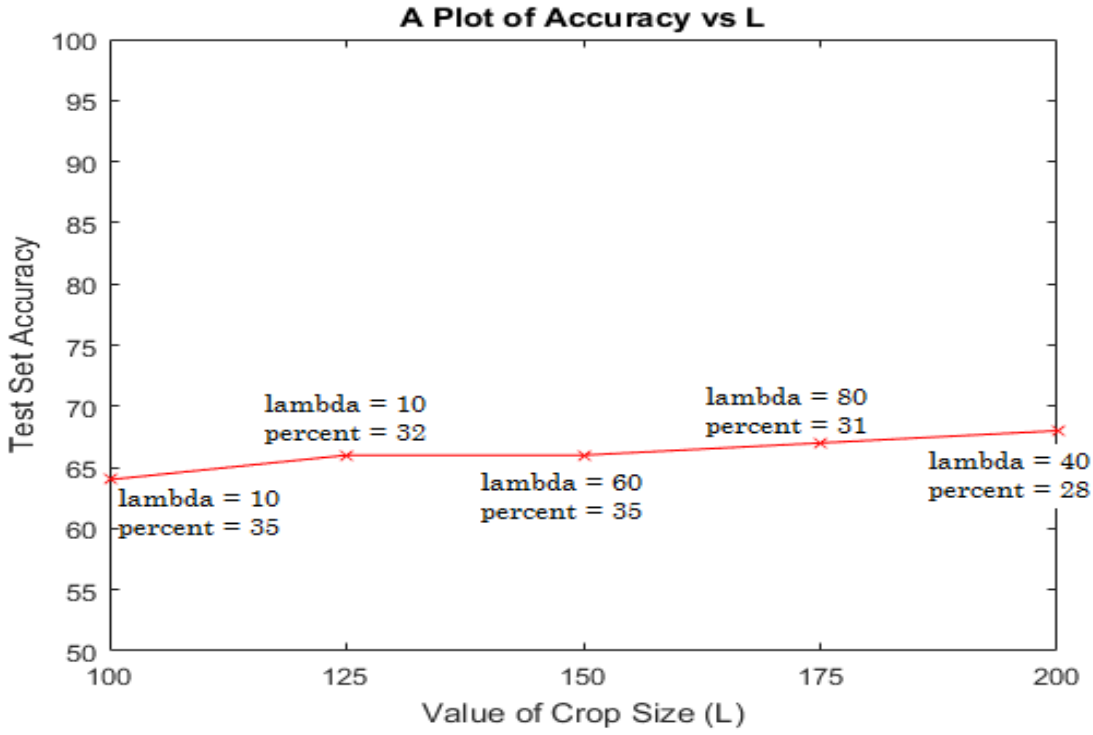

Figure 20

To understand the term ‘percent’ in Figure 20, consider the following:

Given, an image (I) is of pxq pixels size,
The image I is cropped into LxL pixel images (each image called as ‘block’ and L is the crop size) and
The number of blocks per image(B) = $\frac{AreaofImage(I)}{10000} = \frac{p*q}{10000}$

For a given image to be predicted,
B = number of blocks into which it is split into
C = number of blocks where y = 1 is predicted

If, C $\geq \frac{percent*B}{100}$ then the complete image is classified as having a crack
Else,
the entire image is classified as having no crack.
Thus, percent denotes the threshold for number of blocks per image to be classified as having crack or no crack.

4.12 REASONS FOR LOW ACCURACY PREDICTION IN THE CROPPING METHOD

While the amount of data is certainly increased by cropping the images, the amount of meaningful data does not increase.
This is because, while cropping images for training, many cropped images will resemble non-crack images which will be modelled as images with crack during training.

Regularization could not handle the large amount of outliers generated by cropping as above.

4.13 PRINCIPAL COMPONENT ANALYSIS (PCA)

When the images are all of same size, principal component analysis helps to reduce the number of dimensions/features in an image. This helps to speed up the linear regression process.

Consider the following example of data where we would like to reduce the dimension of the 2-D points to 1-D :

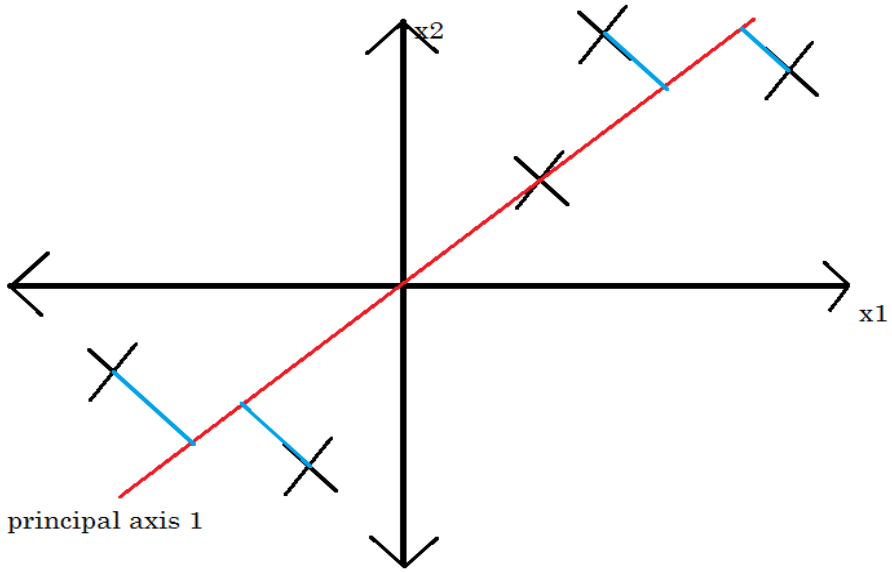

Figure 21 : Projection of 2-d data onto Principal axis 1

PCA tries to find a lower dimensional surface, which is a line in this case, onto which to project the data so that the sum of squares of the perpendicular projections (blue line segments) is minimized. The length of those blue line segments is sometimes also called the projection error.

In contrast to the red line, we can also project our data onto a different line(green line) as shown in Fig below.

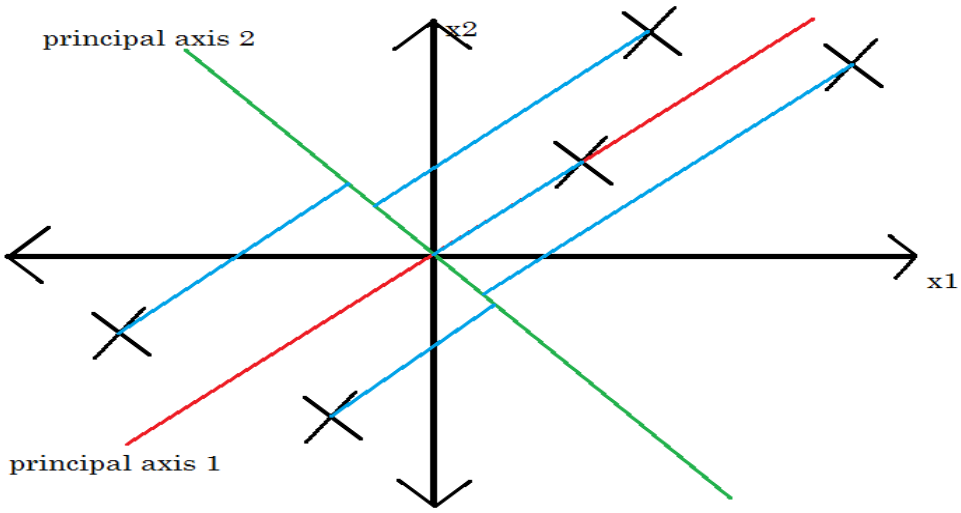

Figure 22: Projection of 2-d data onto Principal axis 2

This green line is a much worse direction onto which to project the data because the projection errors, that is the blue line segments, will be huge. And so that's why PCA will choose something like the red line rather than the green line.

The above is a case of reducing data from two-dimensional to one-dimension. Generally, we have n-dimensional data and we'll want to reduce it to k-dimensions. In that case we want to find not just a single vector onto which to project the data but k-dimensions onto which to project the data so as to minimize this projection error.

4.14 PCA ALGORITHM

The algorithm assumes the input data is in the form of a matrix $X \epsilon R^{mxn}$.
Where, m = number of input images
n = number of pixels per image (features)

The first step is data pre-processing wherein each feature is mean normalized.

Consider the 'm' images to be $x^{(1)},x^{(2)},\ldots,x^{(m)}$ and each image having 'n' features.
Then, $\mu_j = \sum_{i=1}^{m} x_j^{(i)}$
And replace each $x_j^{(i)}$ with $x_j - \mu_j$

Now, we compute the covariance matrix
$\sum = (1/m)* X'*X$
This matrix has the covariance of all the features.

For example, if X were a 3X3 matrix, we have:

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} \end{bmatrix} = \begin{bmatrix} | & | & | \\ \text{feature1} & \text{feature2} & \text{feature3} \\ | & | & | \end{bmatrix}$$

And $X^T = \begin{bmatrix} - & \text{feature1} & - \\ - & \text{feature2} & - \\ - & \text{feature3} & - \end{bmatrix}$

Thus, $\sum = 1/m * \begin{bmatrix} \text{feature1}^2 & \text{feature1} * \text{feature2} & \text{feature1} * \text{feature3} \\ \text{feature2} * \text{feature1} & \text{feature2}^2 & \text{feature2} * \text{feature3} \\ \text{feature3} * \text{feature1} & \text{feature3} * \text{feature2} & \text{feature3}^2 \end{bmatrix}$

The next step is to compute the Singular Value Decomposition (SVD) of ∑.

$$[U,S,V] = \text{svd}(\sum)$$

The matrix ∑ being symmetric always, the U and V vectors are orthogonal and identical and are the principal axis that we require.

Thus, $U = \begin{bmatrix} | & | & | \\ u^{(1)} & \dots & u^{(n)} \\ | & | & | \end{bmatrix} \epsilon R^{nxn}$ are the principal axis.

We choose the first 'k' columns of this matrix U and obtain the reduced co-ordinates of the original data as :

$$X\_new = X * U(:,1:k)$$

where, $X\_new \epsilon R^{mxk}$ is the reduced image data.

We choose a 'k' such that maximum variance of the original image data is retained. To retain 99% of the variance we choose a 'k' such that

$$\frac{1/m \sum_{i=1}^{m} ||x^{(i)} - x_{\text{new}(i)}||^2}{1/m \sum_{i=m}^{m} ||x^{(i)}||^2} \leq 0.01 \ (1\%)$$

Where the numerator is the summation of the 2-norm distance between each image and the new reduced image and denominator is the variance of original image data.

In MATLAB, all the steps above can be computed using the command 'pca' which performs mean normalisation and SVD giving the following return values:

[U,projection,latent,tsquared,explained] = pca(X)

Where projection(:,1:k) = X_new
and, the term 'explained' describes the percentage of variance explained by each principal component.
For example, to choose k such that 99% of the variance is captured,
In Matlab we use,

$$\text{sum}(\text{explained}(1{:}k)) \geq 99\%$$

and choose the minimum value of k that satisfies the above equation.
This equation is same as the above equation for k.

4.15 EXAMPLE OF PROJECTING 2-D DATA TO 1-D

Consider a 2-D dataset of 5 points as - $\begin{bmatrix} 1 & 2 \\ 5 & 5 \\ 7 & 9 \\ 11 & 13 \\ 12 & 15 \end{bmatrix}$

Thus, our input data $X \in R^{5x2}$
We would like to reduce this data to $\in R^{5x1}$ by projecting the data onto the 1^{st} principal axis.

By using [U,projection] = pca(X) we obtain the U and projection matrix as:

$U = \begin{bmatrix} 0.6388 & 0.7694 \\ 0.7694 & -0.6388 \end{bmatrix}$

Where the first two columns are the two principal axis and,

$$\text{projection} = \begin{bmatrix} -9.1923 & -0.4265 \\ -4.3290 & 0.7347 \\ 0.0261 & -0.2816 \\ 5.6588 & 0.2408 \\ 7.8364 & -0.2674 \end{bmatrix}$$

Where the first column is the projection of our 2-D data on first principal axis. The method to obtain these projections is illustrated from Figure 23 to Figure 28.

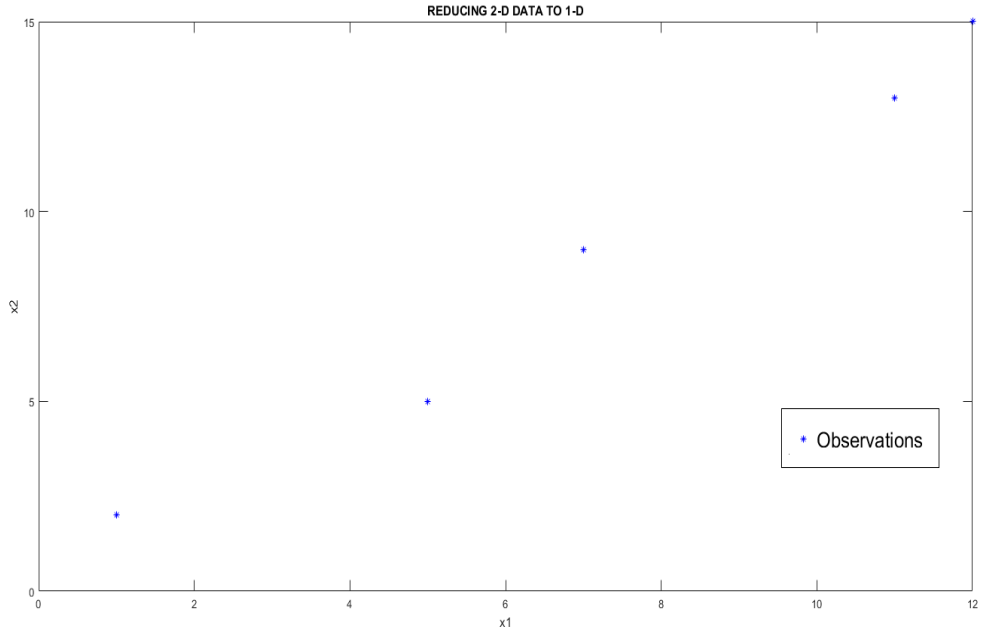

Figure 23: Observations plotted in 2-D

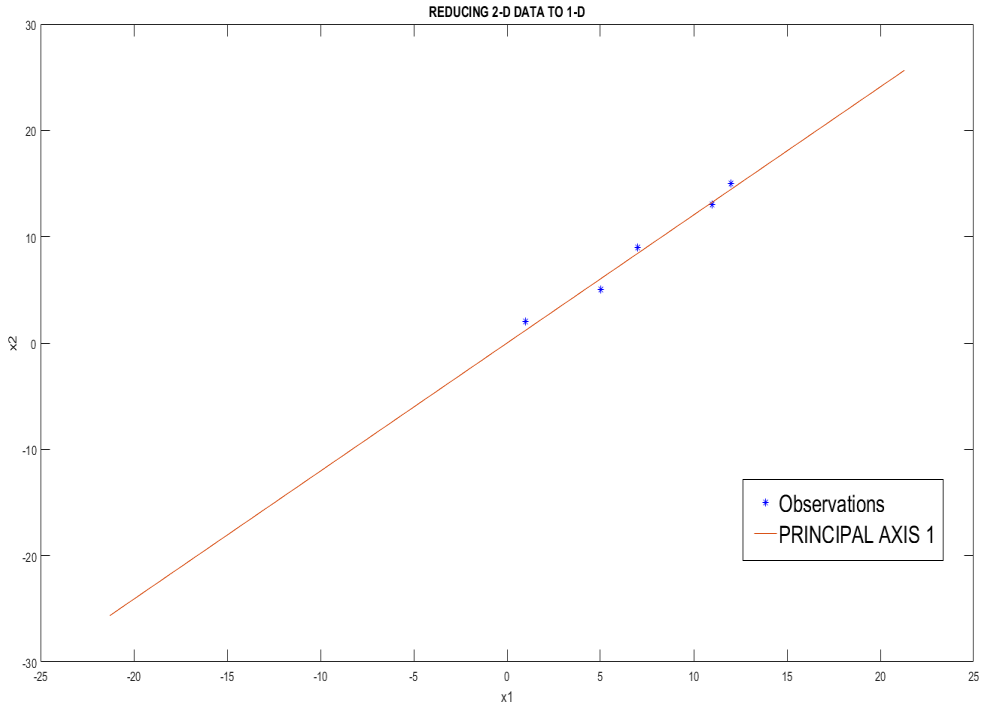

Figure 24: Plot of principal axis 1 vector obtained from ‘U’ matrix

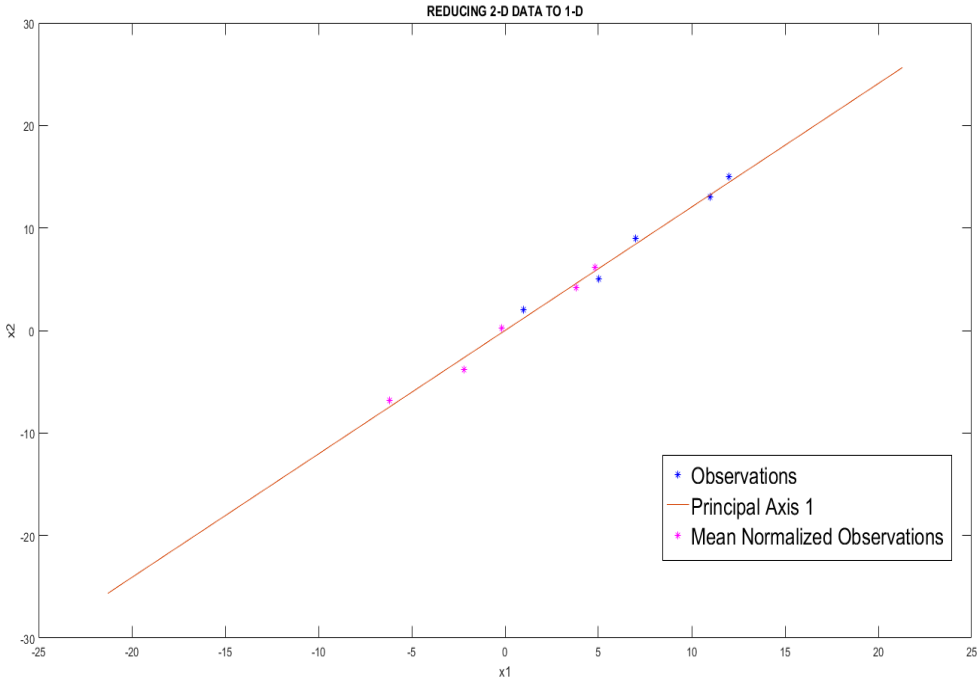

Figure 25 : Mean normalized observations and original observations

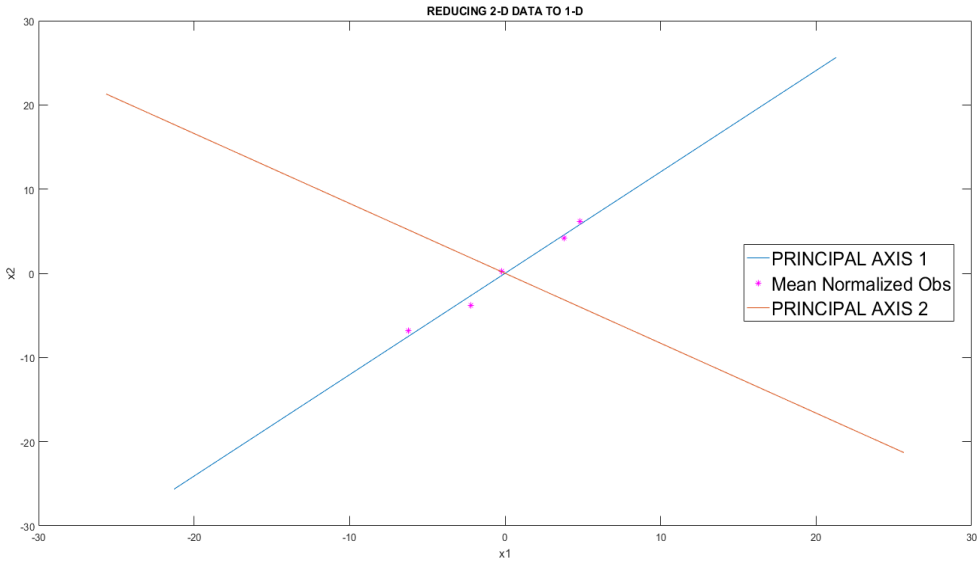

Figure 26 : Principal axis 1 and principal axis 2 vectors

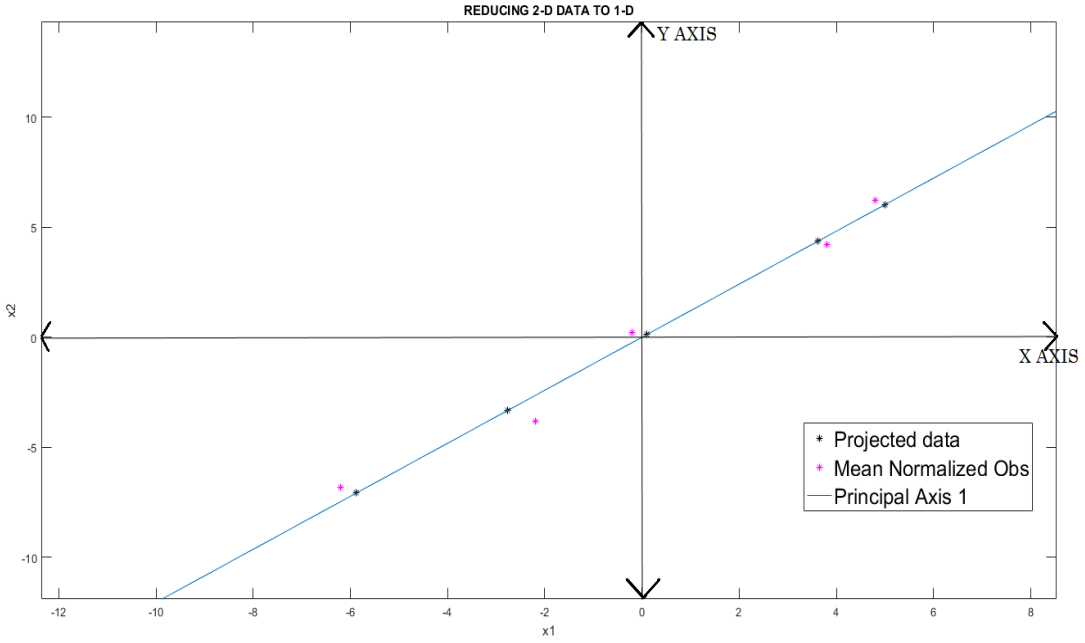

Figure 27: Projecting data onto principal axis 1

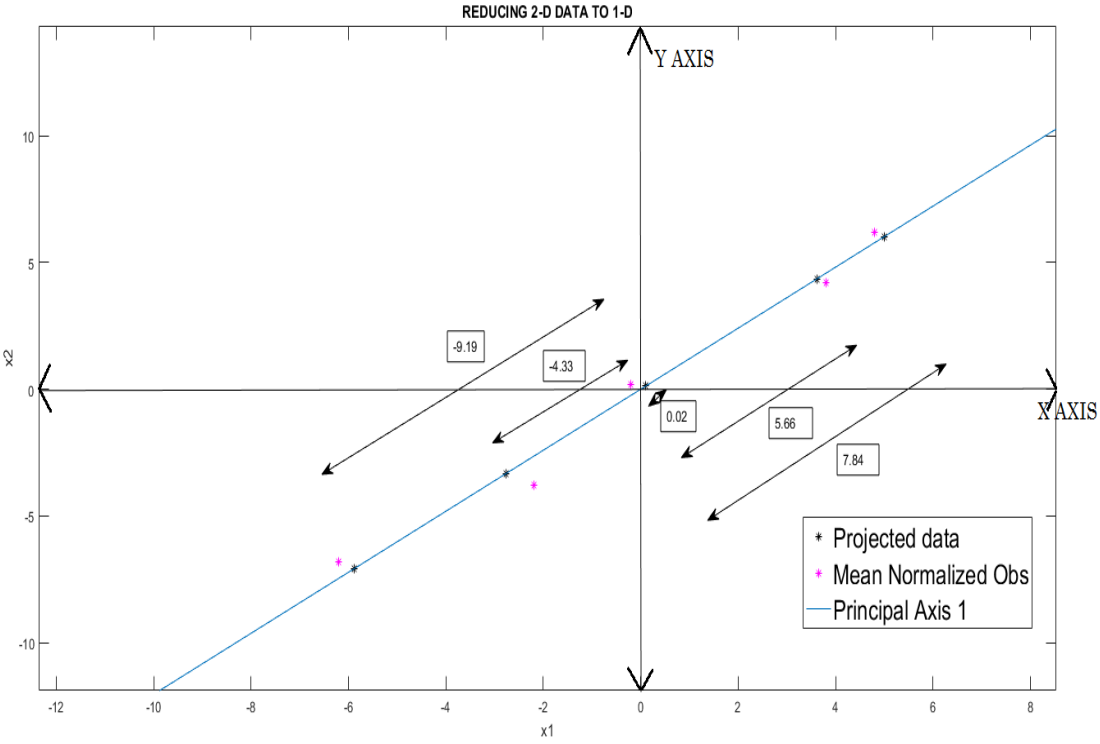

Figure 28: The distance from origin to projected points is the projection matrix

As observed from Figure 28, the distances from origin are the same as first column of projection matrix obtained above.

Thus, the X_new ϵ R^{5x1} is $\begin{bmatrix} -9.19 \\ -4.33 \\ 0.02 \\ 5.66 \\ 7.84 \end{bmatrix}$

4.16 EXPERIMENTAL RESULTS

Our input data X to PCA is the training data consisting of crack and non-crack images.
We obtained two videos of real time underground pipe inspection performed by Western Water Group, Melbourne, Australia.

By taking snapshots, we then separated the images for training the logistic regression and testing as follows –

	Number of images with crack	Number of images without crack
For Training	161	76
For Prediction	33	25

A sample of crack images used for training are as shown below:

Figure 29 : Training set for crack images

Each image is of 720x576 pixels, 161 image with crack and 76 images without crack were used to train the logistic regression.
Thus, X ϵ $R^{237x414720}$

As we can observe the huge number of features (i.e., 414720) per image make the computations very slow.

We used the MATLAB command,
[U,projection,l,t,explained] = pca(X) and obtained:

$U \epsilon R^{414720x236}$ which are the 236 principal axis found,

projection $\epsilon R^{237x236}$ are the projections of the 237 images on each of the 236 principal axis obtained.

We choose different values of 'k' and use
X_new = projection (:,1:k) as our new training data and train the parameters of logistic regression and then predict the test data.
Now, with this 'k' above we also need to reduce the dimensions of the test images by using:

X_testnew = X_test * U(:,1:k)

Where X_test is the original test image data $\epsilon R^{58x414720}$ as 58 images were predicted and,
X_testnew is the reduced test data ϵR^{58xk}

The prediction results for different values of 'k' used are as shown below:

k	Lambda	PREDICTION ACCURACY	TRAINING SET ACCURACY
10	1	72.4	79.7
20	1	87.9	89.4
30	1	84.5	96.2
40	1	62.0	100
50	1	62.0	100
60	1	63.8	100
70	1	84.5	100
80	1	89.6	100
90	1	87.9	100
100	1	81.03	100

110	**1**	**89.6**	**100**
120 TO 230	1	87.9	100

We chose k = 110 because sum(explained(1:110)) = 99.26 %

Thus for k = 110, 32/33 images with crack and 20/25 images without crack were predicted successfully.

4.17 TESTING ON BEAGLEBONE

After obtaining the matrix 'U' ϵ $R^{414720x236}$, the X_test is reduced to X_testnew as described in chapter 4.16. This X_testnew is used for prediction of crack as described in chapter 4.9.
The code used is shown in Appendix B.

The time taken by Beaglebone to perform these operations and recognize one image of 720x576 pixels for presence of crack was found out to be 24.8 seconds.

The same operation took 0.3 seconds on a PC with an Intel i7 CPU with processor speed 3.4Ghz and 8GB RAM.

4.18 SUMMARY

Logistic regression requires all the training and prediction images to be of the same size. Thus, two methods of running logistic Regression were investigated – First, with a training set with images of various sizes and second with a training set of images obtained from Western Water Group, Australia from their underwater pipe inspection.

Initially, a method of cropping images of various sizes into same crop sizes and regressing them was presented along with its merits and de-merits. This method gave a low accuracy of 67% on predictions of test images.

Thereafter, an improved algorithm based on principal component analysis was presented which improved the accuracy to 89.6%.

It was also observed that the image recognition performance of Beaglebone is slow and could be improved by using a CPU with higher processing power in the AUV.

The next chapter presents a method to detect cracks on images of any size using OpenCV to overcome the drawback of collecting images of same size in training and test sets for using logistic Regression.

CHAPTER 5

CRACK DETECTION USING COMPUTER VISION

5.1 INTRODUCTION

Computer vision is concerned with automatic extraction and analysis of data in the form of an image or a collection of images to deduce useful information. The image data can take the form of an image or a video sequence or data from a medical scanner.

The project uses programming functions from OpenCV(Open Source Computer Vision). OpenCV is a library available online , that contains functions for real-time computer vision. It was originally developed by Intel.

Areas of Application of OpenCV include facial recognition , pattern recognition , motion tracking , augmented reality etc. And to achieve the above mentioned applications , OpenCV includes statistical machine learning libraries that contains numerous machine learning algorithms like Boosting , Naive Bayes Classifier , Support vector Machine(SVM) to name a few.

Machine learning problems that involve classification can be solved using OpenCV classifier functions that applies one or a combinations of the above mentioned algorithms for classification problems. Classifiers are algorithms that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

5.2 PROCEDURE FOR TRAINING CLASSIFIER

A classifier (namely a *cascade of boosted classifiers working with haar-like features*) is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (*stages*) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different ***boosting*** techniques (weighted voting). Currently, Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported.

5.3 BOOSTING AND ADABOOST ALGORITHMS

Boosting is a general ensemble method that creates a strong classifier from a number of weak classifiers.
This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first

model. Models are added until the training set is predicted perfectly or a maximum number of models are added.

AdaBoost was the first really successful boosting algorithm developed for binary classification. It is the best starting point for understanding boosting.

5.4 HOW TO TRAIN ONE MODEL USING ADABOOST

A weak classifier (decision stump) is prepared on the training data using the weighted samples. Only binary (two-class) classification problems are supported, so each decision stump makes one decision on one input variable and outputs a +1.0 or -1.0 value for the first or second class value.

The misclassification rate is calculated for the trained model. Traditionally, this is calculated as:

$$\text{error} = (\text{correct} - N) / N$$

Where error is the misclassification rate, correct are the number of training instance predicted correctly by the model and N is the total number of training instances. For example, if the model predicted 78 of 100 training instances correctly the error or misclassification rate would be (78-100)/100 or 0.22.

This is modified to use the weighting of the training instances:

$$\text{error} = \text{sum}(w(i) * t_{\mathbf{error}}(i)) / \text{sum}(w)$$

Which is the weighted sum of the misclassification rate, where w is the weight for training instance i and $t_{\mathbf{error}}$ is the prediction error for training instance i which is 1 if misclassified and 0 if correctly classified.

For example, if we had 3 training instances with the weights 0.01, 0.5 and 0.2. The predicted values were -1, -1 and -1, and the actual output

variables in the instances were -1, 1 and -1, then the t_{error} s would be 0, 1, and 0. The misclassification rate would be calculated as:

$$error = (0.01*0 + 0.5*1 + 0.2*0) / (0.01 + 0.5 + 0.2) \text{ or,}$$

$$error = 0.704$$

A stage value is calculated for the trained model which provides a weighting for any predictions that the model makes. The stage value for a trained model is calculated as follows:

$$stage = \ln((1\text{-}error) / error)$$

Where stage is the stage value used to weight predictions from the model, ln() is the natural logarithm and error is the misclassification error for the model. The effect of the stage weight is that more accurate models have more weight or contribution to the final prediction.

The training weights are updated giving more weight to incorrectly predicted instances, and less weight to correctly predicted instances.

For example, the weight of one training instance (w) is updated using:

$$w = w * \exp(stage * t_{error})$$

Where w is the weight for a specific training instance, exp() is the numerical constant e or Euler's number raised to a power, stage is the misclassification rate for the weak classifier and t_{error} is the error the weak classifier made predicting the output variable for the training instance, evaluated as:

$$t_{error} = 0 \text{ if}(y == p), \text{ otherwise } 1$$

Where y is the output variable for the training instance and p is the prediction from the weak learner.

This has the effect of not changing the weight if the training instance was classified correctly and making the weight slightly larger if the weak learner misclassified the instance.

5.5 STEPS INVOLVED IN TRAINING CLASSIFIER

The steps involved in training the cascade classifier are as follows:

1) A negative image is selected and 100x100 sections of the image is cropped out of it from random areas to generate the negative sample set .

2) The positive images are formed by appending the object to be detected on to the negative sample set at random positions. Here the object to be detected is the crack.

Here are some of the cracks the are used for generating positive sample set.

Figure 30: Crack image templates

The generated positive images are shown below :



Figure 31 : Image1 Image 2 Image 3

3) An information file is generated that contains the locations of the objected to be detected(cracks) on the negative background.
4) The classification is done using the OpenCV trainer that takes in the negative and the positive data along with the information file generated. The number of stages have to specified along with it.

5.6 EXPERIMENTAL RESULTS

Once training is done , a .xml file is generated by the classifier that contains parameters associated with image training.

The training was done using 10 stages. As the number of stages increase the training time also increases. But as the number of stages increase , the accuracy of detection will also increase . For the limited computation capacity that was available to us , we found satisfactory results with 10 stages. It took about 47min for 10 stages on a system with 3GHz processor and 4 GB RAM. The detected images are shown below :

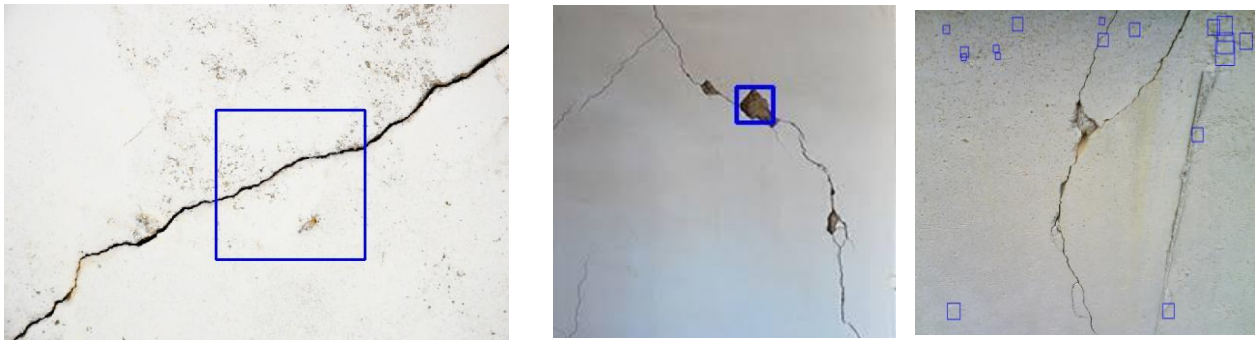

Figure 32 : Detected cracks on Images for stage 12, 10 and 8

The blue box indicates the crack location detected by the classifier after training.

5.7 SUMMARY

This chapter presents a further study into crack detection using functions from the libraries of OpenCV.
The details of the algorithm that is used by the classifier for detecting the crack is presented in this chapter as well as the steps involved in training the classifier.

Once the classifier is trained using collection of positive and negative sample set of images, an .xml file containing the parameter is generated by the function which can be then used to detect crack on any test image.

The experimental results of the trained classifier are also presented in this chapter.
Next chapter presents the conclusions of this thesis and future work on the AUV.

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

In this thesis we have investigated the challenges faced in constructing an Autonomous Underwater Vehicle (AUV). The thesis describes several methods of image recognition using machine learning. This chapter provides the summary of the main contributions and open problems for future work.

6.1 CONTRIBUTIONS OF THE THESIS

The mechanical structure is completed and the hull was water proofed so that the payloads can be protected. The motor speed was adjusted to reduce vibration on the body. The design provides three degrees of freedom movement which can be controlled using pwm signals from the microcontroller that are fed to the motors via ESCs.

Several image recognition techniques were considered among which logistic regression with principal component analysis for dimensionality reduction was chosen because of its simplicity, speed and robustness for binary classification.
The logistic regression algorithm implemented gave 89.6 % accuracy on binary classification for detecting cracks.

To overcome the limitation of logistic regression wherein all the training and prediction images need to be of same size, a study was done in the area of computer vision and results of different stages were documented.

The next section describes the extensions to the present work.

6.2 EXTENSIONS AND FUTURE WORK

i) Training of cascade classifiers as well as image recognition in the AUV requires more computational power.
Proposed solution :- Training of cascade classifiers requires high end Graphical Processing Units(GPUs) to reduce training time which enables us to test various possible configurations. The image recognition by the AUV can also be improved by using a GPU instead of microcontrollers such as Beaglebone.

ii) Location sensing at 30m depth inside a mild steel pipe surrounded by concrete.
Proposed solution :- Since the range of WiFi /GPS/ GPRS at such depths are not robust enough to capture the location of the AUV, we intend to capture images at a regular time interval with a time stamp embedded onto them to locate the AUV.

iii) The Li-Po battery has high discharge rates and hence cannot be used to power microcontrollers on board. On the other hand, using lead acid batteries is safer but increases the weight of AUV.
Proposed solution :- Battery eliminator circuits need to be used if Li-Po batteries are to be used.

iv) The Li-Po battery that is used has an endurance of only 15min at maximum load.
Proposed Solution :- We can increase the number of Li-Po batteries by connecting them in parallel along with an emergency power supply cable.

v) The pipelines have turn, gradients ,descents and dead ends. So the AUV has to navigate through such a route.
Proposed Solution :- Waterproof ultrasonic proximity sensors can be used to calculate the distances from the surface of the pipe and manoeuvre accordingly.

APPENDIX A

CALIBRATION AND SPEED CONTROL CODE

A.1 CALIBRATION CODE

```
#include <Servo.h>

#define MAX_SIGNAL 1500
#define MIN_SIGNAL 1000
#define MOTOR_PIN 2

Servo motor;

void setup() {
  Serial.begin(9600);
  Serial.println("Program begin...");
  Serial.println("This program will calibrate the ESC.");

  motor.attach(MOTOR_PIN);

  Serial.println("Now writing maximum output.");
  Serial.println("Turn on power source, then wait 2 seconds and press any
key.");
  motor.writeMicroseconds(MAX_SIGNAL);

  // Wait for input
  while (!Serial.available());
  Serial.read();

  // Send min output
  Serial.println("Sending minimum output");
  motor.writeMicroseconds(MIN_SIGNAL);

}

void loop() {
}
```

A.2 SPEED CONTROL CODE

```
uint16_t pwmSignal[3]={1050,1250,1050 };
uint16_t summationSignal=0;

bool tog=0;

#define MAX_CYCLE 20000// 20mS

#define NUMBER_OF_SIGNAL_CHANNELS 3

#define CH1 0

#define CH2 1

#define CH3 2

void setup() {

  DDRD|=0b00011100; //pin 2,pin3,pin4

  Serial.begin(115200);

  Serial.println("programStart");

  DDRB|=(1<<5); // pin 13 OP

  PORTB|=(1<<5); // switch on pin13

  delay(1000);

  PORTB&=~(1<<5);// switch off pin13

  for(uint8_t i=0;i<NUMBER_OF_SIGNAL_CHANNELS;i++)
  summationSignal+=pwmSignal[i];

}

void loop() {

while(1) // infinite loop makes code run faster
```

```
{

 PORTD|=(1<<2); // switch on pin 2
 delayMicroseconds(pwmSignal[CH1]);
 PORTD&=~(1<<2); // switch off pin 2

 PORTD|=(1<<3);// switch on pin 3
 delayMicroseconds(pwmSignal[CH2]);
 PORTD&=~(1<<3); // switch off pin 3

 PORTD|=(1<<4);// switch on pin 4
 delayMicroseconds(pwmSignal[CH3]);
 PORTD&=~(1<<4);// switch off pin 4

uint16_t diff=MAX_CYCLE-summationSignal;
delayMicroseconds(diff);

tog=!tog;

tog?PORTB|=(1<<5):PORTB&=~(1<<5); // toggle led pin 13

// end of cycle
}
```

APPENDIX B

IMAGE RECOGNITION IN BEAGLEBONE

```
from PIL import Image
from scipy import io
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime


img = Image.open('/projdata/proj/crack-
images/scene03688.png').convert('L')
img.save('greyscale3.jpg')

#conversion to an array
arr = np.asarray(img)
arr =np.reshape(arr , (1,414720))

#load the theta obtained after training


theta = io.loadmat('/projdata/proj/all_theta')
img_arr = []
j = 0
rad = np.zeros(110)
startTime = datetime.now()
for i in range(5):
    a = io.loadmat('/projdata/proj/U'+str(i)+'.mat')
    a = a['tst']
    partial = np.matmul(arr , a)
    rad[j:j+20] = partial
    j+=20

#    rad = np.concatenate(rad , partial)
```

```
a = io.loadmat('/projdata/proj/U5.mat')
a = a['tst']
partial = np.matmul(arr , a)
rad[100:110] = partial

theta = theta['all_theta']
theta_trans = np.transpose(theta)


rad = np.append(1, rad)
out = np.matmul(rad , theta_trans)

sigmoid = 1/(1 + np.exp(-1*(out)))
if (sigmoid[0] > sigmoid[1]) :
    y = 0;
else:
   y =1;
print(y)
print(datetime.now() - startTime)
```

REFERENCES

[1] https://en.wikipedia.org/wiki/Autonomous_underwater_vehicle

[2] http://robots.dacloughb.com/project-2/esc-calibration-programming/

[3] http://www.t5eiitm.org/2014/02/iitms-underwater-journey/

[4] https://www.coursera.org/learn/machinelearning/home/week/3

[5] http://www.holehouse.org/mlclass/06_Logistic_Regression.html

[6] http://www.holehouse.org/mlclass/07_Regularization.html
https://stats.stackexchange.com/questions/69602/what-is-a-classifier

[7] http://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/

[8] http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html