Genetic And Evolutionary Feature Extraction For Autonomous Road Following

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Jayandra Pokharel

August 2014

Major Professor: Terence Soule, Ph.D.

Authorization to Submit Thesis

This thesis of Jayandra Pokharel, submitted for the degree of Master of Science with a major in Computer Science and titled "Genetic And Evolutionary Feature Extraction For Autonomous Road Following", has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor _____Date_____
Terence Soule, Ph.D.

Committee Members _____Date_____
Robert Heckendorn, Ph.D.

_____Date_____
Gregory Donohoe, Ph.D.

Department Administrator _____Date_____
Gregory Donohoe, Ph.D.

Discipline's College Dean _____Date_____
Larry Stauffer, Ph.D.

Final Approval and Acceptance by the College of Graduate Studies

_____Date_____
Jie Chen, Ph.D.

Abstract

Image Processing and computer vision has always been a major component in the field of robotics and machine learning. The information extracted from image processing varies widely depending upon the nature of tasks to be performed by the robot. The goal of this research is to enable a robot to find a path for autonomous trail following. In the research an evolutionary approach has been used to determine the minimal regions in input images, which are discriminating enough for the robot to distinguish between path and non-path. Image processing is computationally intensive in nature, requiring significant processing time and main memory. So, any reduction in area of the image to be processed for decision making is always advantageous. In this research two sets of experiments are done to compare the effectiveness of evolved regions for image processing against processing the whole image. In the first set the robot processed the entire frame from the camera to make decisions for driving on a trail by avoiding non-trail. In the second set image processing is done on only the evolved region of the frame to make the decision. It was found that image processing over the evolved region covering less than 30% of the captured image can give results comparable to processing the whole image.

Acknowledgements

I would like to express special thanks to my major professor Dr. Terrance Soule for all the guidance and support he has provided me to successfully complete the thesis. It is he who has helped me pick the topic for this thesis, and has always guided me from the very beginning. This thesis would not have been possible without his keen judgments and proper guidance. I would also like to express my special thanks to Dr. Robert Heckendorn for all the support he has provided. He has always provided the research with new ideas that can be incorporated, such that the research would become more practical. I would also like to thank Dr. Gregory Donohoe for serving as my committee member despite his business, and providing guidance for various improvements.

I would like to thank my fellow labmates Juan Marulanda, Nathaniel Ebel and Travis DeVault from "Laboratory For Artificial And Intelligent Research" (LAIR) for all their support and co-operation. They have always helped me in every steps of the experiment. Whenever I needed a helping hand during data collection, or wanted to do a code review, they would always be there for me.

Last but not the least, I would like to thank my family and friends for all their motivations and support they have continually provided in every stages of my life. I would like to dedicate and share this accomplishment with all of you.

Table of Contents

Authorization to Submit Thesis . . . . . **ii**

Abstract . . . . . **iii**

Acknowledgements . . . . . **iv**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

1 Introduction **1**

2 Background **5**

2.1 Image Processing 5

2.2 Machine Learning 6

2.2.1 Unsupervised Learning 7

2.2.2 Reinforcement Learning 7

2.2.3 Supervised Learning 8

2.3 COTS Bots 10

2.4 Evolutionary Computation 11

2.5 GEFE and Evolving regions 13

2.6 Local Binary Pattern 15

2.7 Edge and Contour Detection 17

3 Goal **18**

4 Experiment **20**

4.1 Experimental Setup 20

4.2 Static 32 regions 23

4.2.1 Vision Training Step 23

4.2.2 Training and Evolution Step 24

4.2.3 Test 28

4.2.4 Result 30

4.3 Evolving regions using GEFE 32
4.3.1 Vision Training Step 32
4.3.2 Training and Evolution Step 32
4.3.3 Test for number of regions 36
4.3.4 Test for nature of regions 42
4.3.5 Test for learning by backpropagation 44
4.3.6 Results 46
5 Summary and Conclusions **49**
6 Future Work **51**
References **52**

List of Tables

3.1 Goals of the experiment 19

4.1 Summary of the evolutionary algorithm parameters 28

4.2 Evolving neural network controller weights using inputs from 32 static regions 30

4.3 Summary of the evolutionary algorithm parameters 35

4.4 Evolving 2 fixed size regions and weights of neural network controller by running evolution for 200 generations 37

4.5 Evolving 3 fixed size regions and weights of neural network controller by running evolution for 200 generations 38

4.6 Evolving 5 fixed size regions and weights of neural network controller by running evolution for 200 generations 38

4.7 Evolving 10 fixed size regions and weights of neural network controller by running evolution for 200 generations 39

4.8 Evolving 10 fixed size regions and weights of neural network controller by running evolution for 300 generations 39

4.9 Evolving 10 variable size regions and weights of neural network controller by running evolution for 300 generations 43

List of Figures

1.1 Robot Platform over artificial track made of felt 2

2.1 GEFE with best 3 and 10 regions 14

2.2 Local Binary Pattern 16

4.1 Training and Testing tracks 21

4.2 Different steps of the experiment 22

4.3 Demonstrating the region of image assumed to be safe road surface . . . 23

4.4 Neural Network Structure 26

4.5 Neural Network Structure Taking Input From Evolved Region 34

4.6 GEFE with 3 regions 40

4.7 Evolved regions of all 5 training cases for 3 regions (left) and 5 regions (right). 40

4.8 10 regions GEFE with fixed size regions (left) and variable size regions (right) 44

Chapter 1: Introduction

With advances in computing people have been automating more and more things. Tellers in bank have been replaced by ATM machines, concession stands in public places have been replaced by vending machines, and cashiers in supermarkets have been replaced by self-checkout machines. This desire for automation has affected many different aspects of our life. In the past few years people have been working hard on automating the vehicles we drive. Private organizations and government agencies have supported many research projects and invested millions of dollars in the field. The Defense Advanced Research Projects Agency grand challenge for autonomous vehicle driving conducted by DARPA (Defense Advanced Research Projects Agency), an agency of the US DOD (Department Of Defense), is a vivid example of such an investment [1]. In 2005 and 2007 teams from Stanford University and Carnegie Mellon University were each awarded $2 million cash prizes for winning the competition [5] [6].

A major challenge for autonomous driving is the huge variation in types of road surfaces and their similarity with surrounding environment. The color, texture, and other visual cues for paved road, graveled road, asphalt road and muddy trail are very different from each other. Using only color detection will almost certainly be insufficient to solve the problem of differentiating road from non-road. Moreover, color detection approaches are very susceptible to lighting conditions. The same road surface might appear to be colored differently, generating different color profiles under different lighting conditions. For this reason, it is desirable to have multiple image processing techniques in addition to color detection (e.g. texture, edge, Hough Lines, etc.) applied to the image, before any decision regarding road surface is made. As image processing is computationally intensive in nature, it requires significant processing time and main memory. So, any reduction in area of the image to be processed for decision making is advantageous. Moreover, for many purposes, the objects of interest lie in certain regions of the image, rather than covering the whole image. For example, in using image processing for identifying road

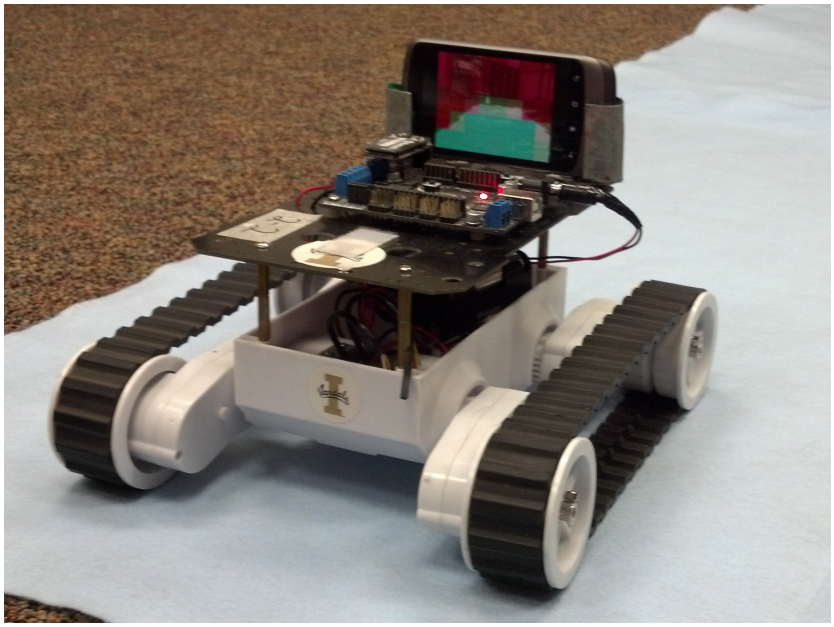

Figure 1.1: Robot Platform over artificial track made of felt
The robot platform built at UI using commodity off the shelf (COTS) principle comprises of an android smart-phone, an arduino microcontroller, and a rover 5 tank chassis. The felt track is sky-blue in color and is 18 inches wide.

surfaces, the upper half of the image might not be as important as the lower half and might even be completely ignored. So, if we could focus only on those specific areas of interest, and do image processing only over them, it would save a lot of processing time. This research focuses on identifying the regions of an image that are discriminating enough for color based image processing techniques to differentiate a road surface from non-road surface.

The research implements a genetic based feature extraction method [16, 17, 18] for evolving regions of interest in the images captured from a camera. An individual in the population is represented by a tuple $< X_1, Y_1, X_2, Y_2 \ldots X_n, Y_n >$, where a pair of X_i and Y_i represent a point in the X-Y plane. Two of these points collectively represent a region. Here, a generational approach (Section 2.4) for evolution has been used for evolving individuals. A tournament of size three is used for selecting parents. Two of the best individuals from the tournament (the parents) exchange their genetic materials using uniform crossover to produce a child. This child undergoes mutation, and is placed

in a separate child population. When the child population is the same size as the parent population, the evolutionary process is switched from the parent population to the child population. This process of creating a child population and switching the evolutionary process to it, is continued for some predetermined number of generations. By the end of the evolutionary process, we will have determined an useful individual that represents good set of regions. Image processing is then done only on those regions and the values obtained are used for making a decision regarding a surface being road or non-road, for driving the robot.

The research uses a neural network to control the robot (Figure 4.4, 4.5). The implemented neural network has three layers. The number of nodes in the input layer corresponds to the number of regions that have been evolved. The hidden and output layers have three nodes each. In this research the number of regions evolved (hence the inputs to neural network) varies from two to ten. So, having higher number of nodes in hidden layer would over learn the training data set, whereas having fewer of them would experience information loss. Having three nodes in hidden layer seemed to be a good balance between these two, so the number was chosen. Similarly, the reason behind having three nodes in output layer is that the robot uses three discrete drive commands (forward, left, and right) to drive itself on the track. Output from each of the nodes correspond to one of these drive commands. The largest value among these three output nodes is chosen to be the direction for autonomous driving. All of the nodes in each layer are connected to all of the nodes in the next layer. The hidden and output layer nodes, in addition to connections from the previous layer's node, have a separate bias input. So for a neural network with 5 inputs, the neural network has 30 weights (5*3+3*3+3+3) that represent the network. The activation function used in all of these nodes is sigmoid.

The tests in the experiment are all done by using a small tracked robot, controlled by a smart-phone mounted on its top (Figure 1.1). The tests were conducted indoors by creating an artificial track made of felt. It was found that by doing image processing in

less than 30% area of input image, the robot could drive autonomously on the testing track. The accuracy of driving decisions made while doing image processing on evolved regions only was similar to the driving decisions made while doing image processing in the whole input image. This saving of CPU resources by removing image processing in unwanted regions of input image can be used for doing other useful tasks.

The thesis is divided as follows. Chapter 2 presents a brief overview and related work in the field of image processing, machine learning, commodity off the shelf (COTS) bots, evolutionary computation, genetic and evolutionary feature extraction, local binary pattern and edge and contour detection. Chapter 3 describes the goals of this research. Chapter 4 presents a detailed explanation of the experimental setup, methods implemented, data collection process, analysis of those collected data, and the results that could be drawn from them. Chapter 5 presents a brief summary and conclusion of the research. Finally, Chapter 6 concludes the thesis with a discussion of possible future works.

Chapter 2: Background

2.1 Image Processing

Image Processing has always been a major component and favored technique for autonomous vehicle driving [7, 22]. To make autonomous navigation more accurate, image processing is often accompanied by different sensor modalities viz. infra-red, RADAR, LIDAR, etc. This combination of image processing with other navigation techniques have shown promising results in autonomous vehicle driving on roads with proper lane markings [21]. However, achieving a similar result in off-road environment remains a huge challenge. Lack of proper lane markings and distinctive road-shoulder boundaries are the major hurdles to overcome. Finding a technique to differentiate between the roadway and its surrounding, when both exhibit very similar color and texture, is the biggest challenge.

Much of the previous research on off-road autonomous vehicle driving involved identifying unique color cues on the road surface and exploiting them. These approaches are often found to be accompanied by different image processing techniques such as segmentation [11] or edge detection [9]. However, all of these approaches exhibit a major drawback. They are not able to adapt to the changes in road surfaces and its surrounding environment. Even a slight change in lighting conditions might confuse the system and lead the vehicle off trail. The system's identifiers then need to be manually re-tuned or re-trained to accommodate to these changes.

Color classification and learning by constructing and using multiple models for road and background has shown successful results in autonomous road following [22]. The approach used in [22] assumes the surface immediately in front of the vehicle to be road, and uses color classification and machine learning to construct multiple road and background models. The input images from camera mounted at front of the vehicle is compared with these road and background models to find a probability that a pixel in the

input image belongs to a particular road model or background model. Temporal fusion is then used to stabilize the results, and generate a final value. To make the vehicle adjust itself to changing conditions in road, these models are regularly updated such that any changes in road surface get incorporated in the model library. When the road model library is full, and a new model has to be added, the oldest one gets removed to make room for the new one. With the implementation of this approach of real time color learning by maintaining a library of road and background model, the authors in [22] were able to successfully deal with unstructured, non-homogeneous, and complex road shapes with varying lighting conditions.

The experiments conducted in this research implement a similar approach. In the research a 50 by 50 pixel square region at the lower center of the image is used to generate models of road surface during the vision training step (refer to Figure 4.3). These models are then used in determining the probability of whether a region in input image is road surface or not. As image processing is computationally intensive in nature, generating a histogram of the input image colors and comparing them with the road histogram models in the image library consumes significant amount of processing time and memory. This research aims at evolving the regions of interest in an image such that performing image processing on those regions only, instead of the whole image, will produce a similar result. This saves an enormous amount of computational resources that can be utilized in performing other important tasks.

2.2 Machine Learning

Machine learning is a branch of artificial intelligence, where an agent learns on its own from the provided training data set, rather than only following explicitly programmed instructions [12, 2]. It is strongly related to the concepts in statistics, optimization and is often conflicted with data mining and pattern recognition. Machine learning can be

divided into three broad categories: unsupervised learning, reinforcement learning and supervised learning.

2.2.1 Unsupervised Learning

Unsupervised learning is a form of machine learning that deals with finding the hidden structures in a series of unlabeled data. Since the data are unlabeled and there is no error or reward signal to evaluate a potential solution, the learning agent tries to cluster the data to find hidden structures. Common approaches of unsupervised learning include hidden Markov models and principle component analysis. In evolutionary robotics an unsupervised learning system has the requirement of being able to judge its performance on its own. This might be easy in simulation, but requires additional hardware or software to be implemented in a physical robot. For example a robot learning to follow a road on its own would require many more sensors than just a camera for computer vision. It might require gyroscope for measuring orientation, LASER or RADAR for measuring distance, tactile sensors for feeling the surface to drive on, bump sensors to find if it has bumped into anything or not, plus other different types of sensors for measuring different parameters. It is only after analyzing the information from this array of sensors, the robot might be able to learn and refine its behaviors for following a road on its own.

2.2.2 Reinforcement Learning

Reinforcement learning is a form of machine learning where the learning strategy includes guiding an agent towards some desired behaviors. The desired positive behaviors are reinforced and unwanted negative behaviors penalized in some way, such that the chances of demonstrating positive behaviors by the agent again in future gets increased. Under this approach an agent performs various actions randomly or based upon the policies learned so far, and is given a feedback for its immediate action. Based upon the

information of surrounding environment and feedback received, the agent builds up its policies. One of the simplest approaches to Reinforcement Learning is "Clicker Training" [10]. Clicker training is a learning approach which is similar to the clicker training used to train animals. Under this approach, the agent performs various actions in response to the provided input or environment, and is given a positive or negative reward for those actions. Based upon the input, action performed and the corresponding reward received, the agent builds up its policies such that it can get more and more rewards.

2.2.3 Supervised Learning

Supervised learning is a form of machine learning where the agent is given a series of labeled data, and has to generalize a function mapping the inputs to outputs. It involves telling an agent what it should do when given a set of inputs, and then using that information to generalize its behaviors. One of the major categories of supervised learning is "Learning from Demonstration". It has been explained in detail below.

2.2.3.1 Learning From Demonstration

Learning from demonstration is a form of supervised learning where the agent builds up its policies by observing and imitating the actions being performed by some trainer. The trainer is usually a human, but it could be another robot or even a simulated demonstrator. This mapping of real world states to actions (called policies) lies at the heart of many robotics applications [3]. The actual method of learning varies, and can include learning policies, mimicking trajectories, model building, and matching movement primitives [15, 3]. Furthermore, learning a desired behavior is complicated by whether or not output behaviors are discrete or continuous within the problem space. Learning from demonstration has been used to train robots to play soccer [20] and to train a large crusher robot to drive through varied, outdoor terrain [19]. In [20] the authors have used

learning from demonstration called "Hierarchical Training of Agent Behaviors" to train humanoid robots to play soccer in RoboCup competition. Similarly, in [19] the authors have used an expert's examples of proper navigation behavior, readings from onboard camera and LIDAR and satellite imagery of terrain to plan an optimal route from point A to point B and drive a huge crusher vehicle along the route autonomously.

An approach very similar to the one described in this thesis has been presented in [14]. That system combines learning from demonstration with back propagation on a neural network to train a controller capable of driving a modified Chevy van autonomously on roads. The controller used was a neural network with 960 input nodes, 5 hidden nodes and 30 output nodes. The controller was able to learn to drive the van autonomously at speeds of around 20 miles per hour by observing the reactions of a human driver, with less than 5 minutes of training. Using learning from demonstrations, the parameter tuning problem, which is an important component in any system involving complex perception system and planning system, gets automated through demonstration instead of experts intervention and manually tuning the parameters. It demonstrates the successful application of imitation learning in improving robustness of autonomous navigation systems, while still minimizing human expert‘s interaction.

Based upon the manner of execution, learning from demonstration can be broadly divided into two categories viz. "Demonstration" and "Imitation" [3]. In the first category "Demonstration", the demonstration is performed on the actual robot or a physically equivalent platform. Based upon how recording is done, "Demonstration" can be subdivided into "Teleoperation" and "Shadowing". In teleoperation, the robot is operated by the teacher itself, such that the robot can use its own sensor data for learning. In shadowing the teacher demonstrates the motions externally, and robot mimics those motions while using its own sensor data for learning. The second category of learning from demonstration is "Imitation", where there is an embodied mapping and demonstration is not done on the robot itself or a physically equivalent platform. Based upon how record-

ing is done, "Imitation" can be sub-divided into "Sensors on Teacher" and "External Observation". In sensors on teacher, data obtained from sensors placed on the teacher while he/she is teaching the behavior are used for learning. In external observation the data from sensors external to the teacher while he/she is teaching the behavior are used for learning.

Using the terminology of [3], our learning from demonstration can be classified as having a teleoperation demonstrator, with a mapping policy derivation scheme, and discrete output actions. The ability of the robot to imitate human behavior using sensor data mapping makes this technique a good candidate for both neurocontroller control and for evolutionary learning.

2.3 COTS Bots

This research is conducted using a robot designed and built at University of Idaho using Commodity Off The Shelf (COTS) components (Figure 1.1). The robot consists of three parts viz. Android smart-phone, Arduino micro-controller, and Rover 5 tank chassis. [a] The smart-phone behaves as the eye and brain of the robot. Images from the camera are processed, fed to the robot controller, and then a decision regarding the direction to be driven (forward, left, right) is made. In this research, both regions for feature extraction and weights of the neural network controller are evolved by using a generational evolutionary algorithm. The individual obtained at the end of the evolutionary process comprises both the co-ordinates of the regions for feature extraction from the input image, along with the weights of the neural network controller (Section 4.2.2, 4.3.2). Image processing is done only on the regions defined by the co-ordinates, and the obtained values are passed as inputs to the neural network. The neural network uses these input values, weights, and an activation function to generate the driving commands. The

[a] Rover 5 Tank chassis http://www.dfrobot.com

driving commands are transmitted to the Arduino micro-controller via Bluetooth. The microcontroller then rotates the motors of the chassis making the robot move. This research uses an additional android smart-phone which acts as a remote control, and is used by the trainer to drive the robot during the training phase. The remote phone communicates with the main smart-phone mounted over the top of the robot via Bluetooth and transmits the driving directions (forward, left, right) given by the trainer. All of the image processing in the research is done using the OpenCV library for android. [b]

2.4 Evolutionary Computation

Evolutionary computation is a branch of artificial intelligence that mimics natural evolution as an approach for solving problems [8, 4]. The process uses mutation, crossover and selection operators in a fashion similar to nature. The purpose of these operators is to maintain diversity in the population, and preserve the best parts of the best individuals to recombine into novel solutions. The fitness operator used in the process differentiates good individuals from bad ones, and the selection operator ensures that those good individuals get transferred from one generation to another. The evolutionary process starts with random possible solutions, which are then continuously optimized through mutation, crossover and selection (both individually or in combination with other individuals) until the desired solution is found.

To better demonstrate evolutionary computation, let's try to solve the knapsack problem by using an evolutionary approach. Knapsack problem is a problem related to combinatorial optimization where we are given a set of items; each with their own mass and value [13]. The goal is to determine whether an item is to be included in a collection, such that the total weight is less than or equal to a given limit, and the total value is as large as possible. The problem derives its name from the dilemma faced by anyone who

[b] http://opencv.org/

is constrained by a fixed-size knapsack, and has to fill it with as much valuable items as possible. In Evolutionary computation, every individual in the population represents a possible solution to the problem. So, if we tried to solve this problem using evolutionary approach, an individual could be represented by an array of 0's and 1's with length equal to the number of items. The value "0" in the array would indicate that the corresponding item is not included, whereas a "1" would indicate that it is included. A fitness function is defined that measures the fitness of each individual. In the above knapsack problem, the fitness could be a measure of total weight of items placed in the bag. If an individual indicates total weight of items carried is over the maximum weight limit, that individual is assigned a fitness penalty and eventually gets discarded through the evolutionary computation process. The individuals then undergo crossover and mutation operations to produce new individuals. Under the crossover operation, portions of an individual (parent 1) get swapped with portions of another individual (parent 2) to generate one or more children. This process is similar to the crossover of genes seen in nature. In the mutation operation, a portion of the individual is changed by a some small amount to generate a new individual. This process is also similar to the mutation of genes seen in nature. The purpose of the crossover operation is to ensure that good genetic material get transferred from parents to their children. On the other hand, the purpose of mutation operation is to introduce new genetic material into the population and maintain its diversity. The fitness of new individuals (children) after crossover/mutation operations is then computed.

The incorporation of children into the population can be done using a steady state or generational approach. In a steady state approach, the children replaces the worst in the population, thus maintaining the population size throughout the process. In a generational approach, the children are stored in a new location, thus creating a separate population of children. When the size of the population of children equals the current population size, the old population is discarded and the evolutionary process switches

to the child population. No matter which approach is chosen, with the passage of every generation, individuals in the population get optimized to find a better solution. This process of generating new individuals by crossover and mutation operations, computing their fitness, incorporating them into the population is continued for some predefined number of generations or until some desired fitness level is achieved.

2.5 GEFE and Evolving regions

Genetic and Evolutionary Feature extraction (GEFE) is an implementation of Evolutionary Computation where a set of feature extractors are evolved such that maximum recognition accuracy can be achieved with minimum number of extractors. GEFE applied to facial images for biometric recognition of faces has shown very good results [17, 16]. In [17], the authors evolved uniform, overlapping, unevenly distributed regions, which did not cover the entire image. The results obtained with these evolved regions were better than the use of standard local binary pattern (LBP) method over the whole image (Section 2.6). By allowing evolving regions to focus on any part of the image, the regions could evolve to focus more on the areas of image that were most discriminating for feature extraction. In the case of [17] most of the regions were found to be focused around the ocular region, suggesting that these regions hold maximum texture information for LBP to differentiate individuals from each other. Use of a similar approach of evolving regions for mitigating iris based replay attacks was also found to be successful [18].

In Genetic and Evolutionary Feature Extraction (GEFE) the population consists of individuals each representing a feature extractor. An individual in the population usually takes the form of six tuples $< X_i, Y_i, W_i, H_i, M_i, f_i >$. The variables X_i and Y_i represent x and y co-ordinates of the center of a patch. The variables W_i and H_i represent the width and height of the patch. The variable M_i represents a boolean that determines whether the patch will be used in computing the fitness of the individual or not. If the

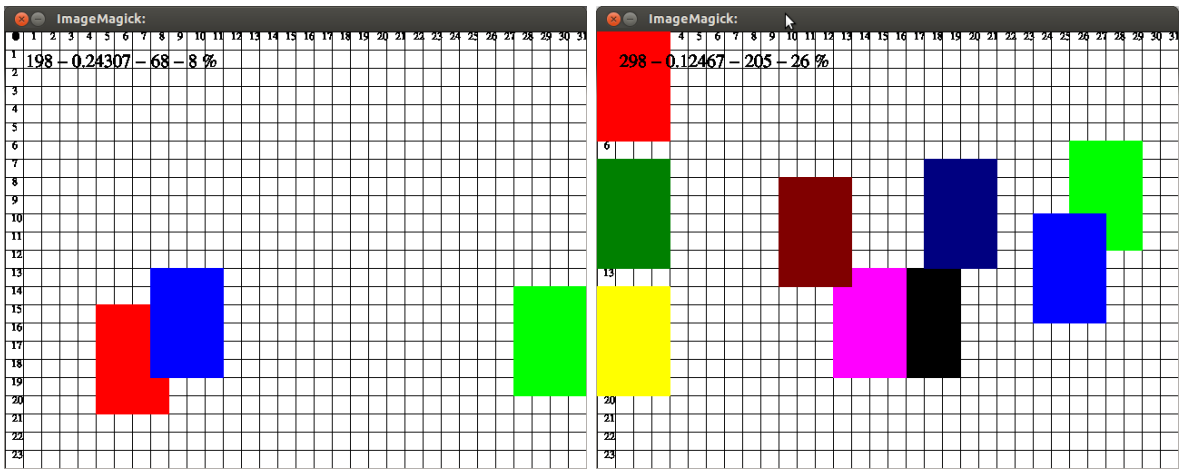

Figure 2.1: GEFE with best 3 (left) and 10 (right) regions
The three regions cover 8% area of the whole image and by doing image processing on these regions only the robot could successfully navigate 83.33% of the test track. The ten regions cover 26% area of the whole image and by doing image processing on these regions only the robot could successfully navigate 91.67% of the test track, which is the same accuracy that was obtained when processing the whole image.

value is 1, the patch contributes to the fitness computation, else it is ignored. Finally the variable f_i represents the fitness of the individual. The individuals in the population then go through an evolutionary process (Section 2.4) to find the best individual in the population.

In this research, an approach similar to GEFE has been used for evolving regions. An individual in the population is represented by tuple $< X_1,\ Y_1,\ X_2,\ Y_2\ \ldots X_n,\ Y_n >$, where a pair of X_i and Y_i represent a point in the X-Y plane. Two of these points represent a region. So an individual with n x and y pairs would represent n/2 regions. These regions are then evolved using evolutionary algorithm (Section 2.4) to find the best regions. An individual's fitness measure is the accuracy of driving decisions made when image processing is done only on the regions selected by the individual. Once the best regions have been determined, image processing is done only on those regions when making driving decisions. Figure 2.1 shows sample evolved regions with 3 and 10 regions respectively, which could drive the robot on the test track fairly successfully. It can be seen that in both cases the processed part covers less than 30% of the whole

image. In the best case where three regions are evolved, the regions cover only 8% area of the whole image. By doing image processing on these regions only the robot could successfully navigate the test track with failures in 4 corners out of 24 corners in clockwise and counter clockwise direction (Figure 4.1). Similarly, in the best case where ten regions are evolved the regions cover 26% area of the whole image. By doing image processing on these regions only, the robot could successfully navigate the test track with failures on 2 corners out of 24. This best result for evolving 10 regions is equal to the autonomous driving accuracy that was obtained while processing the whole image (failures in 2 corners out of 24, 91.67% successful navigation). As image processing is computationally intensive in nature requiring significant processing time and main memory, reduction in area of the image to be processed makes the computation much faster. Moreover, the freed resources can be used for doing other important tasks.

2.6 Local Binary Pattern

The authors in [17] have implemented local binary pattern for facial recognition. They have used both fixed size regions and varying size regions for feature extraction. Use of fixed size regions had given an accuracy of 100% for face recognition. It was found that the majority of evolved regions were focused around the ocular region, indicating that this area holds textures that are enough to differentiate one person from another. Similar to the approach in [17], in this research we have used fixed size regions of dimensions 120 pixels by 80 pixels and variable size regions for genetic and evolutionary feature extraction (Section 2.5). However, the local binary pattern has been replaced by backprojection. A brief introduction to local binary pattern has been given in the following paragraph for reference purposes.

Local Binary Pattern (LBP) is a feature extraction technique used in computer vision. It is used to capture the texture information from an image. The advantage of LBP over

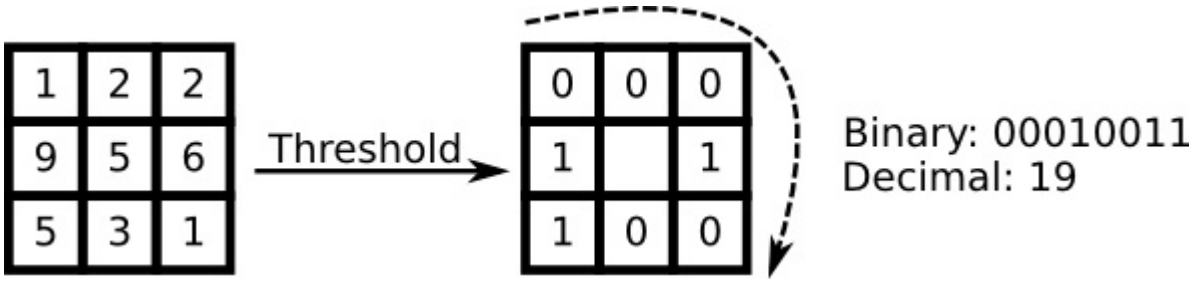

Figure 2.2: Local Binary Pattern
Computation of LBP for the pixel with intensity level 5 (left figure), based upon the intensity level of its neighbors. The value '0' for a neighbor (right figure) indicates that the neighbor has its intensity level less that the pixel of interest, and vice versa.

other texture operators is that it is fast to compute and resilient to variations in lighting conditions. In LBP, the value of any pixel in the image is computed based upon the relative values of its neighbors, so the variation in lighting condition gets canceled out. If the image is bright, the pixel of interest, along with all its neighbors, will probably have high intensity level, and if the image is dark, the pixel of interest, along with all its neighbors, will probably have low intensity level. Thus, the results of relative intensity level computation will be same in either case. LBP computation is usually done with radius 1 or 2 or 3. For simplicity let us consider a LBP computation with radius 1 and thus 8 neighbors (Figure 2.2). For any pixel in the image whose LBP value has to be computed, its intensity level is compared with the intensity level of its surrounding neighbors. If the neighbors have intensity level greater than the pixel of interest, the neighbor would get a value of 1. Otherwise, it gets a value of 0. The LBP value for the pixel of interest would now be the equivalent binary number when starting from any corner (top-left corner has been chosen in Figure 2.2) and writing the 8 neighbors binary values. To better illustrate the computation of LBP, let us consider a pixel with intensity level 5 in Figure 2.2. The relative value of the pixel with respect to its neighbors is shown in the second rectangle. The final LBP value for the pixel is then the binary number which we get by starting at top-left neighbor, and going in the clockwise direction. In Figure 2.2 (page 16), the value is calculated to be 00010011 or 19.

2.7 Edge and Contour Detection

Edge detection and contour detection are two other very useful approaches in image processing. Though the experiment does not implement edge detection and contour detection, implementing them can be a next step into the research. Both of these techniques have been described below for reference purposes.

Edge detection is defined as the use of mathematical tools for identifying discontinuities in a digital image. The tools detect sudden changes in brightness and/or color and use it to determine if there is an edge between two surfaces. It is one of the many feature detection methods that are used in image processing. Some of the other feature detection methods that resemble edge detection include Harris corner detection, Hough circle detection, and Hough line detection. The most popular, and commonly used method for edge detection is Canny edge detection. It is based upon Canny algorithm developed by Jonh F. Canny in 1986. The method uses multistage approach for detecting a wide range of edges in an image. Some of the stages include noise reduction, finding of intensity gradient and hysteresis thresholding. A lot of image processing libraries support Canny edge detection (including the openCV library for Android used in this project).

Contour detection is the approach of joining neighboring edges such that they form a closed curve. This method is often used in image processing to identify the objects present in an image. As contour detection involves forming of closed curves, objects that are partially present in an image often get left out. To detect them as well, four lines have to be draws along the edges of the image such that these lines close the curve enabling the partially present object to be detected. OpenCV library for android supports contour detection as well. The method “findContours()” can be used for this purpose.

Chapter 3: Goal

The experiments conducted in this research evolve regions in an image with an attempt of finding the minimum number of regions that can effectively capture sufficient information from the image, for the task of autonomous path following. The regions are evolved using Genetic and Evolutionary Feature Extraction (GEFE) method (Section 2.5) in the robot phone, mounted at the top of COTS Bots platform (Section 2.3). The task chosen for measuring performance is the ability of robot to navigate the test track autonomously, when image processing is done on the evolved regions only. The robot is first trained on a training track, and then put on an autonomous mode on a testing track. In the autonomous mode, only the portion of image covered by evolved regions are used for image processing and making driving decisions. The training and testing track are deliberately chosen to be different such that the generalization ability of neural network controllers used in this research could also be tested.

The first goal of the experiments is to see if GEFE can be successfully implemented for autonomous trail following without hampering the robot's performance, in comparison to case where entire image is processed. For this, two sets of experiments are performed. In the first set the whole input image is processed, whereas in the second set GEFE is used to evolve regions for image processing. The trade-off between the area of the image that has to be processed with the performance of the robot in driving autonomously over the testing track is measured. The second goal of the experiments is to determine if allowing the evolving regions to have different sizes leads to any performance improvement over evolving the regions by restricting them to a fixed size. For this a set of experiments are done by evolving same number of regions, where in the first set the size of regions are fixed, and in the other set the regions are allowed to have varying size. The third goal of the experiments is to determine if the use of an evolutionary approach or backpropgation is better for evolving weights of a neural network controller, while using GEFE for evolving regions. For this a set of experiments are conducted, where in the first set of experiments

the weights of the neural network are evolved by using evolutionary approach, and in the second set of experiments backpropagation is used to train the values of the weights. A summary of all the goals in this research has been listed in table 3.1

Goal	Details
1	Determine feasibility of GEFE in autonomous road following
2	Determine effect of nature of regions (fixed vs. variable) in GEFE process
3	Compare the efficiency of backpropagation vs evolution to train neural network while using GEFE

Table 3.1: Goals of the experiment

It was found that by doing image processing in less than 30% area of input image, the robot could drive autonomously in testing track (Table 4.2, 4.8 and Figure 2.1). The accuracy of driving decisions made while doing image processing on evolved regions only were similar to the driving decisions made while doing image processing in the whole input image. This saving of CPU resources by getting rid of image processing in less important regions of the input image can be used for doing other useful tasks. Moreover, evolving regions by fixing their size was found to produce better results than evolving regions with varying size. In evolving regions with varying size, the regions were found to cover a larger percentage of the input image for image processing, and the performance of robot in autonomous mode was also not very good (Table 4.8 and Table 4.9). Furthermore, while using GEFE for evolving regions, evolving weights of neural network controller by using evolutionary approach was found to be faster and more successful in driving the robot during autonomous mode, than using backpropataion to train its weights (Section 4.3.5).

Chapter 4: Experiment

4.1 Experimental Setup

The experiments are done indoors using a COTS robot (Section 2.3) and artificial tracks made of felt (Figure 4.1). The android smart-phone in the robot used for capturing and processing images was a Samsung Galaxy Nexus. The phone has a dual core, 1200 MHz, ARM Cortex-A9 processor, 1 GB main memory and a 5 Mega Pixel rear facing camera. The remote control phone used was a Samsung Galaxy Y. The remote control phone is used only for transmitting the driving directions (forward, left, right) via Bluetooth to the robot during the training phase. No processing is done on it, so it's hardware specification has no impact on the experiment. The training and testing tracks are shown in Figure 4.1. The training track consists of 3 stretches of paths with angles of 45 degrees between them. On the other hand, the testing track comprises of a series of straight paths and angular turns. The numbering of corners on the testing track, along with the lengths of the outer edges for the tracks are given in Figure 4.1. As the training and testing track are different from each other, with differences in both the lengths of the straight stretches and of the angles between them, any success of the robot while drive autonomously on testing track implicitly demonstrates the generalization ability of neural networks.

The experiment conducted can be broadly divided into two sub-experiments. In the first sub-experiment, the input image (640 pixels wide by 480 pixels high) is divided into 32 static regions (80 pixels wide by 120 pixels high) covering the whole image, and image processing is done on all 32 regions. In the second sub-experiment, the regions for feature extraction in input image are evolved. Based upon the different goals mentioned in Chapter 3 (Table 3.1), the second sub-experiment includes experiments with fixed size regions, varying size regions, and training of weights by the evolutionary algorithm versus training by backpropagation.

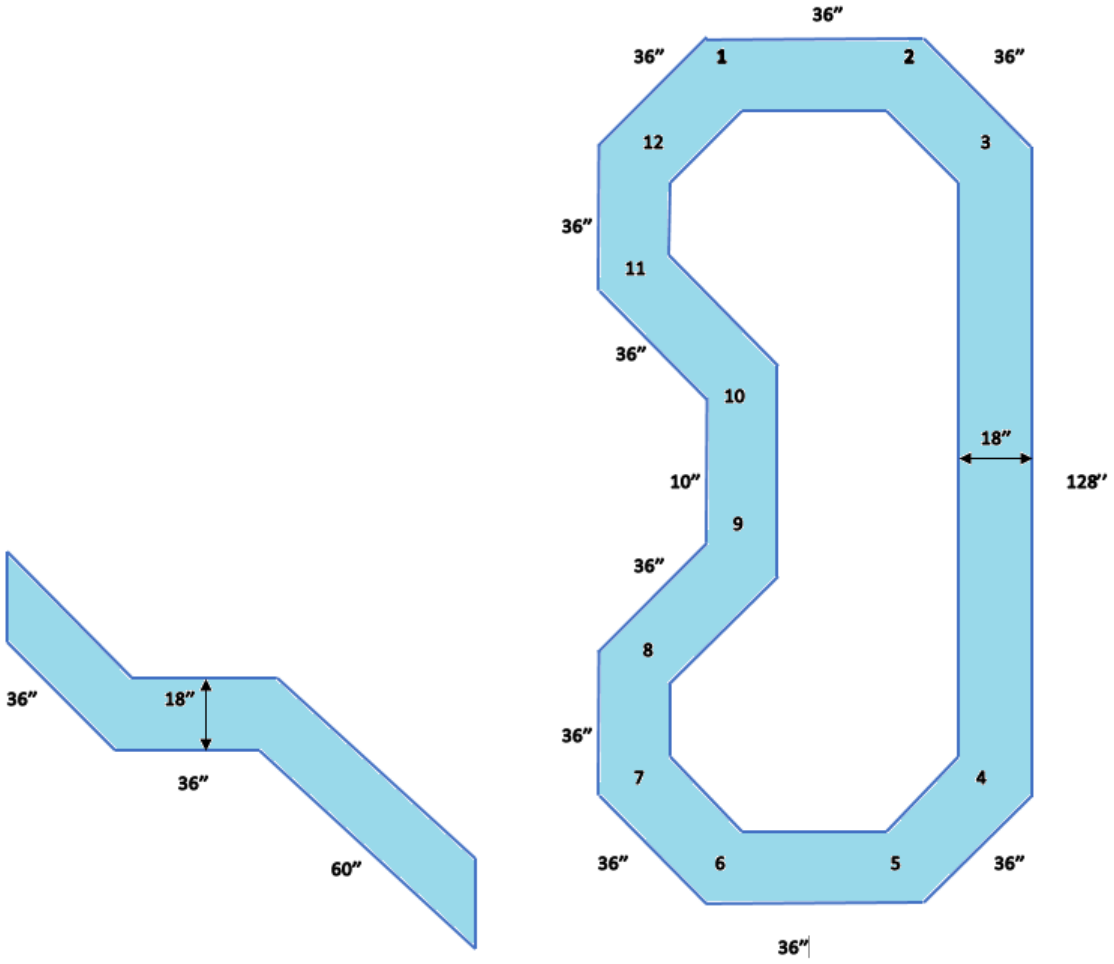

Figure 4.1: Training (left) and Testing (right) tracks
The training track comprises of 3 stretches of paths with angles of 45 degree between them. The testing track comprises of 12 stretches of paths with varying lengths and angle in between them. During autonomous mode, the robot is placed in between corners 3 and 4, and run autonomously in both clockwise and counter clockwise directions.

Each of these sub-experiments comprises of 3 steps (Figure 4.2). The first step is "Vision Training" step, followed by "Training and Evolution" step and "Test" step. In vision training step, human operator drives the robot on the training track by placing it in "Train Vision" mode, where some samples of road surfaces get collected for building a histogram library of safe road surfaces. This library is later used during image processing for finding road probabilities in input image. In the evolution step the human trainer drives the robot on the training track by placing it in "Training" mode. In this mode

Figure 4.2: Different steps of the experiment
The experiment comprises three different steps. In the vision training step a histogram library of safe road surfaces is build. In the evolution step a training library set comprising of images and corresponding driving directions is build by driving the robot on training track. This library is used for evolving regions and weights of neural network. In test step the robot is run on test track and the corners where it fails to successfully navigate is recorded.

the robot phone builds a training library set comprising of images captured from the camera, and driving direction given by the trainer for that image. This training set is used for fitness measurement while evolving regions and evolving weights for the neural network controller. In the test step the robot is placed in "Autonomous" mode and run on the testing track. The best neural network obtained from the training phase acts as the robot controller during testing, and makes its driving decision by processing input images from robot phone's camera. If the regions for image processing have been evolved, image processing is done only on those regions and passed as input to the neural network. Otherwise the whole image is processed by dividing it into 32 regions, each of which is passed as an input to the neural network. In either test, the robot is initially placed midway between corners 3 and 4 of the testing track, and run autonomously in both clockwise and counter clockwise directions. So, for each training case, two sets of test data are collected. A detailed explanation of the phases of the experiments and the experiments conducted are given in following sections.

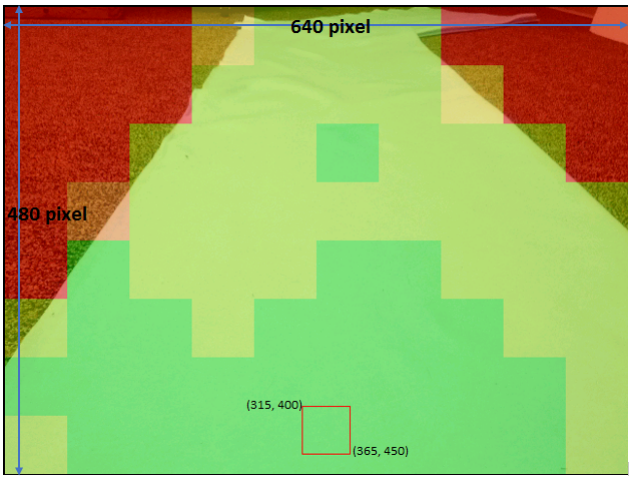

Figure 4.3: Demonstrating the region of image assumed to be safe road surface
The 50 by 50 pixel square region at the bottom of input image is assumed to be safe road surface. A color histogram library of these safe road surface is used during backprojection calculation to find the probability of a pixel in input image belonging to a road surface.

4.2 Static 32 regions

In this sub-experiment, the input image (640 pixels wide by 480 pixels high) is divided into 32 static regions (80 pixels wide by 120 pixels high) covering the whole image. The probability of each region being road surface calculated by using backprojection method is then passed as an input to the neural network controller for making autonomous driving decisions. Thus the neural network controller has 32 input nodes. A detailed explanation of each phases of the sub-experiment is given below.

4.2.1 Vision Training Step

This is the first step of the sub-experiment, where a library of histograms of safe road surfaces is prepared. It is done by giving the robot thirty drive commands on the training track. With each command, a sample image is taken from the bottom center region which is assumed to be safe road surface (Refer Figure 4.3). The histogram values of those images are used to generate a road color histogram library. This library is later used during the learning/training and autonomous/testing phases to calculate road

probabilities. The road probability values are calculated for each regions in the input images by using the BackProjection method in the OpenCV library running on robot phone. Backprojection is a technique used for finding objects of interest in an image. The operation creates an output image of the same size as input, where each pixel corresponds to a probability that it belongs to the object of interest. In our case the object of interest is road surface, so the output probability values from the method indicates the probability that a pixel in the input image is road surface. Then we compute the probability values for regions by using backprojection and the histogram library of safe road surfaces. To find the probability value of a region, the individual probability values of pixels within that region are averaged. Average probability values are computed for all the histograms in the library. The largest probability value among them is chosen to be the probability value for that region. Pseudo-code representing the probability value computation for regions in an input image is shown in Algorithm 1.

Algorithm 1 : Computation of road probability for regions in an image

```
outputProbabilities [ number-of-regions ] {initialize array to 0}
for each histograms in library do
   for each regions in image (indexed as r) do
      avgProbability = Sum of probabilities of every pixel in region / number of pixels
      if avgProbability > outputProbabilities [ r ] then
         outputProbabilities[ r ] = avgProbability
      end if
   end for
end for
return outputProbabilities
```

4.2.2 Training and Evolution Step

In the second step of the sub-experiment a library of training cases is generated and the weights of neural network are trained. Each of these training cases is comprised of an image and a driving direction (forward, left, right) pair associated with that image. To generate the library, a human trainer switches the remote phone to “training” mode, and drives the robot on training track. When the user gives a drive command (forward, left

or right) the robot phone captures the frame at that moment from its camera, and stores the pair (image and drive command) as a training case. Multiple cases are combined to create a library of training cases, which is then used to train the weights of the neural network controller. The size of training cases library was fixed to be 45. This library contained fifteen training cases for each of the three drive commands (forward, left, and right).

The neural network implemented in both sub-experiments has 3 layers (Figure 4.4). In this sub-experiment, the input layer consists of 32 nodes. The input to these nodes are the probability values of 32 regions from the images of the training set computed using backprojection and the histogram library from the vision training step. The number of nodes in both hidden and output layers are fixed to be three. Three nodes are used in the hidden layer because the number of regions evolved varies from two to ten. Having higher number of nodes in hidden layer would over learn the training data set, whereas having fewer of them would experience information loss. Having three nodes in hidden layer seemed to be a good balance between these two, so the number was chosen. Similarly, the reason behind having three nodes in output layer is that the robot uses three discrete drive commands (forward, left and right) to drive itself on the track. Output from each of the nodes correspond to one of these drive commands. The largest value among these three output nodes is chosen to be the direction for autonomous driving. The output from all of the nodes in a layer are passed as input to all of the nodes in next layer. Each node in the hidden and output layer has an additional bias input. So, in the sub-experiment every neural network has a total of 111 (32 * 3 + 3 * 3 + 3 + 3) weights. An array of 111 floating point values is used to represent the neural network. An evolutionary algorithm is used to train the weights of the neural networks. By the end of the evolutionary process, the network should have its weights trained such that it can successfully generate correct driving direction for input values of road-probability. The best trained neural network is then used for driving the robot in autonomous mode for testing.

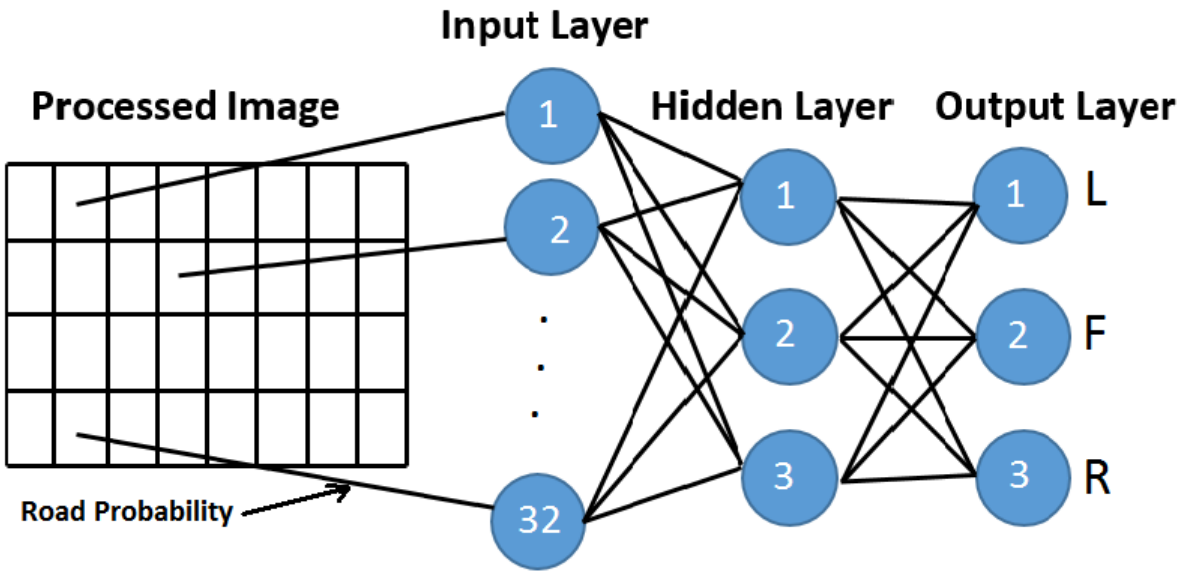

Figure 4.4: Neural Network Structure
The neural network consists of three layers and is fully connected. The number of nodes in input layer correspond to the static 32 regions covering the whole image. The hidden and output layers consist of three nodes each with an additional bias.

The evolutionary algorithm used for the evolving weights of neural network is a standard generational evolutionary algorithm (Section 2.4). An individual in the population is represented by an array of floats of size 111. Each element in the array corresponds to one of the weights in the neural network. The array is initialized with random values in range of -1.0 to 1.0. Selection of individuals for evolution is based upon a tournament of size three. The two best parents selected in the tournament undergo mutation and crossover to produce one offspring. This offspring is then added to the next generation. For crossover, uniform crossover was used. So, every gene (a floating point number) from the parents chromosomes have a 50% chance of being exchanged in generating the child chromosome. The mutation rate is 0.2. So, every weight of the neural network has a 1/5 probability of being mutated. The mutation added a uniformly distributed random value in the range [-0.1, 0.1] to the original weight. Fitness of an individual is then calculated. The library of training images with associated driving direction collected during

the training step is used for this purpose. The neural network represented by an array of weights (an individual in the population) is provided the road probability values from the 32 regions of an image in the training set, and uses that value to calculate the output driving direction. The output driving direction computed by the network is compared with the actual driving direction associated with that image. If the network generated the same action as the action stored in the training set, fitness is improved (here decremented because we are minimizing the error, refer Algorithm 2). To ensure that the child population always contains good individuals, elitism has been implemented. The two best individuals from parent population are always copied to the child generation.

The process of passing the road probability values to the neural network and comparing the resulting driving direction is done for all the training cases in the training library, thereby determining the fitness value of a single individual in the population. The fitness values of all individuals are initially 0.0, and are only calculated during the construction of the next generation. So, in the case of first generation, a random selection of individuals get transferred to the second generation. The training cases are divided into categories based upon expected actions (left, right, and forward). An individual can earn an equal amount of fitness for getting all the cases in a category correct, regardless of the number of cases within the category. Algorithm 2 describes the fitness function which assigns a better fitness to an individual if it's output of the correct action is higher than the output at its other actions. Full fitness is given if the difference between the highest activated output node and the sum of the remaining nodes is above a threshold value of 0.4. No testing was done on this threshold value; it was chosen because it was a mid range value, and both high and low values were expected to work poorly in the fitness function. If the network chooses the correct action, it always gets a positive fitness for that training case, but the fitness bonus in larger if the difference is at least equal to the threshold.

Algorithm 2 : Genetic Algorithm Fitness Function

```
for each network in population do
    fitness = 0
    for each training case do
        actualResults = network result from running training case
        expectedResults = results from training case
        winningValue = actualResults at the position of the expectedResults winner
        difference1 = winningValue - actualResults at first position other than winner
        difference2 = winningValue - actualResults at second position other than winner
        for each difference do
            if difference ≥ threshold (0.4) then
                fitness += 0.5 / number of training cases in that category
            else
                fitness += 0.5 * (difference / threshold) / number of training cases in that category
            end if
        end for
    end for
    fitness = 1 - fitness / 3
end for
```

A summary of the evolutionary algorithm is given in Table 4.1. The table presents the evolutionary parameters used during the evolution process and their corresponding values.

Population Size	75
Generations	200
Mutation Rate	0.2
Crossover type	Uniform
Selection Method	Tournament (size 3)
Elitism	2 best individuals form parent generation gets copied to children generation

Table 4.1: Summary of the evolutionary algorithm parameters
The evolutionary algorithm used is a standard generational algorithm. To ensure that the children population always contains good individuals, elitism has been implemented.

4.2.3 Test

As mentioned in Section 4.1, the experiment conducted comprises 3 different steps viz. train vision, evolution/training, and testing. The experiment begins by placing the robot phone in train vision mode and using the remote phone to issue the driving commands. Once the robot is in the train vision mode, it is driven on the training track (Figure

4.1) where it generates a color histogram library of safe road surfaces (Subsection 4.2.1). A total of thirty sample road images are generated during this phase, so the histogram library comprises thirty road surface histograms. The robot is then placed in evolution/training mode by issuing a command from remote phone. In this step, the robot is driven on the training track where it associates each user provided drive command with the frame from camera at that moment, to prepare a library of training cases. The size of training cases library is fixed to be 45. The library contains 15 training cases for each of the three drive commands (forward, left, right). When the training set comprises of 10 cases of forward, left and right commands each, evolution of the neural network weights is triggered. The number 10 for training cases was chosen such that from the very beginning of the training/evolution process there are enough cases to train the weights. As mentioned in section 4.2.2, the evolution process is continued for 200 generations.

When the evolution process is complete, the robot phone is placed in "Autonomous" mode and driven on the testing track (Figure 4.1) to collect test data. The robot is placed midway between corners 3 and 4 and run in both clockwise and counter clockwise directions. During each test run, the corner in which robot fails to navigate successfully is recorded. Here, failure means the cases where the robot goes off track, or cannot not make a decision to move forward, or left and right in more than five repetitions. In the case of any failure, the robot is picked up and placed on the track following the failed corner. For example, when the robot fails to navigate corner 7, it is picked up and placed in the stretch between corners 7 and 8 or 6 and 7 depending upon the clockwise or counter clockwise direction it is driving, thus skipping the failed corner. The data collection process then continues normally until the robot navigates the whole track. While placing the robot on a stretch, it is always placed at a position equidistant from both edges of the 18" wide test track.

	Clockwise		Counter Clockwise			
Training Set	Run Time	Failed Corners	Run Time	Failed Corners	Total Success	Success (in %)
1	12:56 min	7	11:32 min	12	22	91.67 %
2	9:36 min	6,2,3	11:45 min	1,5	19	79.17 %
3	6:50 min	6,10	9:45 min	1,12,4	19	79.17 %
4	7:48 min	7,12	7:41 min	no failures	22	91.67 %
5	10:57 min	success	9:08 min	12,7	22	91.67 %
Total					104	86.67 %

Table 4.2: Evolving neural network controller weights using inputs from 32 static regions The columns indicate training set, runtime and corners failed in both clockwise and counter clockwise directions, number of corners successfully navigated (out of 24 corners) and the percentage of successful navigation.

The data that was obtained during the experiment is given in Table 4.2. It comprises of a list of corners for the clockwise and counter clockwise runs, that the robot failed to successfully navigate.

4.2.4 Result

Analyzing the data in table 4.2 it can be seen that when the whole image is used the robot is able to successfully navigate 104 corners out of total 120 corners (12*2*5) for the five training cases on the testing track (an accuracy of 86.66%). The reason behind failures in 16 corners is likely to be due to a combination of different factors. The factors include flaws in learning of weights by neural network controller, training library set being insufficient, flaws in image processing, image auto-correction in robot phone, vision training library being insufficient, blurring of images, etc. The learning of weights by neural network controller might not have been effective because of the evolutionary parameters chosen in the sub-experiment. The number of generations for evolution (200 in this

sub-experiment) could have been insufficient to train the weights of the neural network controller, or the fitness function chosen might be missing some important components. Neural networks normally require a large data-set to be properly trained. So, a total of 45 training images and their associated driving direction could have been insufficient to fully train the network. The backprojection method of image processing is sensitive to lighting conditions. Though the experiments are all done indoors and the lighting conditions are constant, the flaw could have been introduced due to image auto-correction feature of robot phone. Whenever the robot moves and a new frame is captured, auto-correction of brightness for that frame by the phone software could be observed. The library of road samples collected during vision training phase (30 in this sub-experiment) might have been insufficient to accurately represent the entire track. The portion of track directly under the fluorescent lights on the ceiling are brightly lit than the portions around it. The 30 samples of road library collected might not have been able to cover all the variations in lighting over the test track. As the images for processing are collected from a moving robot, the frame captured by the robot phone's camera can be blur. So image processing on the blurred image is likely to produce bad road probability values, which then gets used by the neural network controller for making the robot's driving decision. Considering all of these factors, the 87% accuracy in autonomous navigation when processing the whole image, is taken as the base case for all the further experiments in this research. As these factors remain same in all the experiments, to prove the feasibility of GEFE in autonomous road following the experiments implementing GEFE for feature extraction should produce an equivalent or better accuracy level.

4.3 Evolving regions using GEFE

In this sub-experiment in addition to evolving the weights of neural network controller, the regions for image processing are also evolved with the goal of capturing sufficient information using the minimum area of the input image. As the evolving regions can place in themselves in any area of input image, the evolutionary process can focus on areas of image which are more discriminating for road following. Image processing is then done on those regions only and passed as input to the neural network controller for making autonomous driving decisions. The result of this sub-experiment is compared with the results of sub-experiment comprising of static 32 regions covering the whole image (Section 4.2) to determine the feasibility of GEFE in autonomous road following. The results obtained within this sub-experiment are also used to determine the effect of nature of regions (fixed versus variable) in GEFE process, and determine the effect of backpropagation versus evolutionary algorithm to train neural network weights while using GEFEA for evolving regions (Table 3.1). A detailed explanation of each phase of this sub-experiment has been given below.

4.3.1 Vision Training Step

This is the first step of the sub-experiment where a histogram of safe road surfaces is prepared. This step is identical to vision training step of the sub-experiment comprising of 32 static regions. Please refer to subsection 4.2.1 for more details.

4.3.2 Training and Evolution Step

This is the second step of the sub-experiment where a library of training cases is generated and regions for image processing are evolved along with evolving the weights of the neural network controller. In the test where backpropagation has been used for training weights,

only the regions are evolved. The process of generating a library of training cases is identical to the process mentioned in subsection 4.2.2.

The neural network implemented in the sub-experiment is similar to the one mentioned in subsection 4.2.2, and comprises three layers. However, unlike the previous sub-experiment, the number of input nodes in this sub-experiment is not fixed. The number of nodes in the input layer of this sub-experiment corresponds to the number of regions that are evolved. Similar to the previous sub-experiment, the number of nodes in both hidden and output layers are fixed to be three. Output from all the nodes in a layer are passed as input to the next layer. The nodes in hidden and output layers have an additional bias input. A pictorial representation of the neural network structure is shown in Figure 4.5. For example if 5 regions are evolved, then the number of weights in the neural network will be 30 (5 * 3 + 3 * 3 + 3 + 3). The regions and weights are collectively represented by a tuple, and the evolutionary algorithm is used for evolving regions and training weights of neural network. If backpropagation is used for training the weights, the tuple comprises only regions. By the end of the evolution process, the regions for image processing will have evolved and the weights of the neural network will be trained. Image processing is then done on those regions only, and passed as input to the best neural network.

The evolution process in the sub-experiment is also similar to the one described in subsection 4.2.2. Similar to subsection 4.2.2, the training library set is used to train the weights of neural network and evolve the regions for image processing. A major difference is the representation of an individual in the population. Instead of an array representation comprising of 32 regions, an individual in the population is represented by a tuple. The tuple encapsulates both the regions of the image to be evolved and the weights of neural network to be trained. A tuple representing an individual in the population is of form $< W_1, W_1 \ldots W_m, X_1, Y_1, X_2, Y_2 \ldots X_n, Y_n >$. The part $< W_1, W_1 \ldots W_m >$ represents the weights of neural network to be trained. The number of

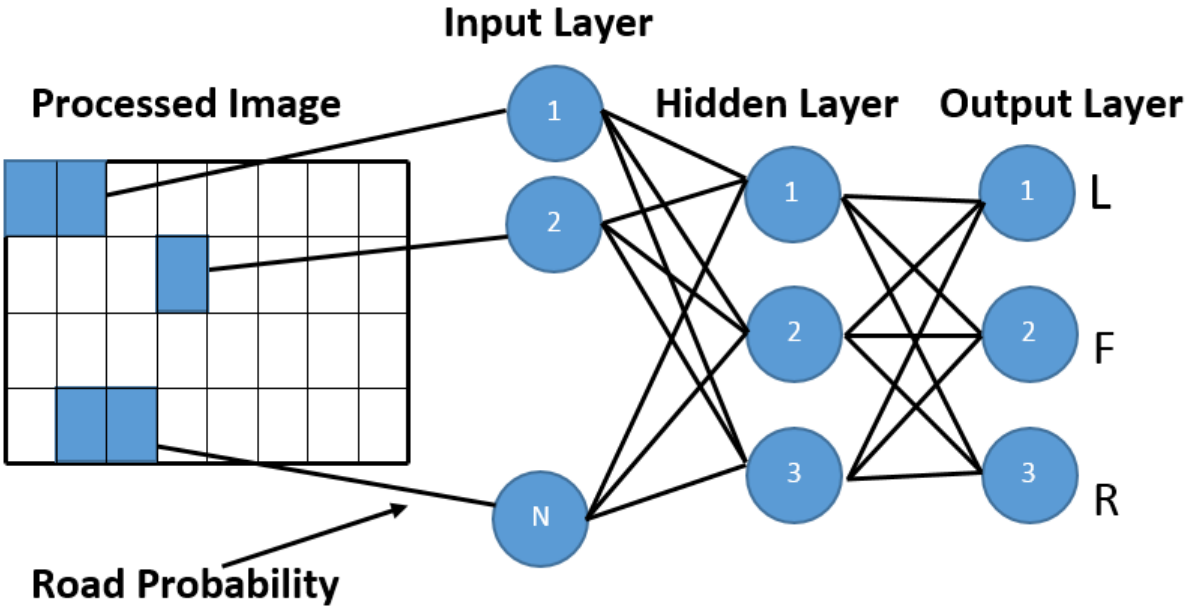

Figure 4.5: Neural Network Structure Taking Input From Evolved Region
The neural network consists of three layers and is fully connected. The number of nodes in input layer correspond to the total number of regions evolved in input image. The hidden and output layers consist of three nodes each with an additional bias.

weights in the tuple correspond to number of weights in the neural network. Unlike the sub-experiment with static regions, where the number of weights is always fixed to be 111, the number of weights is this sub-experiment varies and depends upon the number of regions being evolved. There is a one to one mapping between number of regions being evolved and the number of nodes in input layer. So, as the number of regions increases, the number of nodes in input layer also increases. In the case where backpropagation is used to train the weights of the neural network, this part is completely absent in the tuple because backpropagation rather than evolutionary algorithm, is used to train the weights. The remaining part of the tuple $< X_1, Y_1, X_2, Y_2 \ldots X_n, Y_n >$ represent regions to be evolved. A pair of X_i and Y_i in the tuple represents a point in the X-Y plane. The co-ordinate system for X-Y plane in the experiment comprises of a grid of squares with 20 * 20 pixel dimensions. The input image from the camera, which is 640 pixel by 480 pixel is divided into 32 * 24 small squares, and the co-ordinate system is applied to

this grid. So, the value of X_n in the tuple is bound to the range of [0, 31], and the value of Y_n in the tuple is bound to the range of [0, 23]. While computing the fitness of an individual, image processing is done only on the evolved regions and the resulting probability values passed as inputs to the neural network. The fitness of an individual is increased if the network generated same action for an input, as the action stored in the training set. The crossover, mutation and selection policy are similar to those of the evolution for static 32 regions (Section 4.2.2). A summary of the evolutionary algorithm has been given in Table 4.3.

The regions that are evolved in this sub-experiment are of fixed size or variable size. A pair of x and y co-ordiantes in the $< X_1, Y_1, X_2, Y_2 \ldots X_n, Y_n >$ part of the tuple represents a point in the 32 * 24 grid system mentioned in the above paragraph. If the regions evolved are of variable size, two of these consecutive points collectively represent a region. So a tuple with n x and y pairs would represent n/2 regions in the image. On the other hand, if the regions evolved are of fixed size, only first half of the tuple representing regions get used, and the second half gets discarded. Similar to the case of variable size regions, a pair of x and y co-ordinates in the tuple represents the top-left corner for the fixed size region. The co-ordinates for bottom-right corner are then computed by using the fixed size region's dimensions. In this research the dimensions of the fixed size region are set to be 120 pixels wide by 80 pixels high.

Population Size	75
Generations	200
Mutation Rate	0.2
Crossover type	Uniform
Selection Method	Tournament (size 3)
Elitism	2 best individuals form parent generation gets copied to children generation

Table 4.3: Summary of the evolutionary algorithm parameters
The evolutionary algorithm used is a standard generational algorithm. To ensure that the children population always contains good individuals, elitism has been implemented. The number of inputs to the neural network controller is equal to the number of regions that have been evolved.

4.3.3 Test for number of regions

The procedure followed for identifying effectiveness of different number of regions in GEFE process, is very similar to the procedure followed for sub-experiment involving static 32 regions (Section 4.2.3). The test can be divided into three different phases viz. train vision, evolution/training and test. The process begins by placing the robot in train vision phase. During this phase thirty road samples are collected, which are used to generate a color histogram library of safe road surfaces (Section 4.2.1). The robot is then placed in evolution/training phase. During this phase the robot is driven on the training track where it associates user provided drive commands with the frame from camera at that moment, to prepare a library of training cases. A total of 45 training cases (15 each of forward, right and left) are used in the experiment. When the library comprises at least 10 of each (forward, left, and right) training cases, evolution gets triggered. The evolutionary process evolves the regions for image processing, and trains the weights of neural networks. The evolution process is continued for 200 generations (Table 4.3). By the end of evolutionary process, the best regions in the input image for image processing, and the best neural network controller to drive robot autonomously are identified. The robot is then placed in autonomous mode and driven on the test track (Figure 4.1) for data collection. Data collection is done by placing the robot midway in between corners 3 and 4, and running autonomously in both clockwise and counter clockwise directions. In the case of any failure, the robot is picked up and placed on the next stretch following that corner. The major difference in this experiment with the procedure mentioned in subsection 4.2.3 is that instead of having 32 static regions for image processing, image processing is done on the evolved regions only. As the experiment is focused on analyzing the effect of number of regions on GEFE, the size of the evolved regions is fixed. Similar to the experiment comprising of 32 regions, each evolved region is 120 pixel wide and 80 pixel high. The resulting road probability values computed by using backprojection on

	Clockwise		Counter Clockwise			
Training Set	Run Time	Failed Corners	Run Time	Failed Corners	Total Success	Success (in %)
1	11:07 min	4,5,6,7,12,2,3	15:32 min	12,11,10,9,8	12	50 %
2	5:07 min	4,5,7,8,11,3	7:30 min	2,12,11,7,5,4	12	50 %
3	10:10 min	5,7,8,9,11,1	8:15 min	9	17	70.83 %
4	8:01 min	4,5,6,7,11,12,2,3	9:16 min	9	15	62.5 %
5	6:08 min	7,9,10	8:49 min	3,2,1,11,8,7,6,5	13	54.17 %
Total					69	57.5 %

Table 4.4: Evolving 2 fixed size regions and weights of neural network controller by running evolution for 200 generations

those regions are passed as inputs to the neural network for making autonomous driving decisions.

To test the effect of numbers of regions in autonomous road following using GEFE, four different sets of experiments were conducted. The four sets comprise evolving 2, 3, 5 or 10 regions and running the robot autonomously on the testing track. Except for the number of regions being evolved, all the other factors of the experiment have been kept the same in all these experiments. The effect of number of regions during evolution is then analyzed by observing the performance of the robot for all these cases in autonomous mode. The data obtained from these sets of experiments have been given in tables from 4.4 to 4.8.

From the data in tables 4.4 to 4.7 it can be seen that in the test track comprised of 120 corners (12*2*5), the robot could successfully navigate 57.5%, 78.33% , 70.83% , 77.5% of it when 2, 3, 5, 10 regions are evolved. As all the experimental parameters except the number of regions being evolved are identical in these experiments, the results indicate the optimal number of evolving regions required for the chosen test case of autonomous navigation. The results suggest that evolving only 3 regions is sufficient to make accurate

	Clockwise		Counter Clockwise			
Training Set	Run Time	Failed Corners	Run Time	Failed Corners	Total Success	Success (in %)
1	8:54 min	10	11:09 min	3,12,8,5	19	79.17 %
2	7:54 min	4,5,2,3	7:18 min	no failures	20	83.33 %
3	7:35 min	10	10:34 min	1,11,10,5	19	79.17 %
4	8:14 min	7	9:57 min	1,10,8,4	19	79.17 %
5	9:50 min	4,6,7,8,12,1	10:09 min	9	17	70.84 %
Total					94	78.33 %

Table 4.5: Evolving 3 fixed size regions and weights of neural network controller by running evolution for 200 generations

	Clockwise		Counter Clockwise			
Training Set	Run Time	Failed Corners	Run Time	Failed Corners	Total Success	Success (in %)
1	13:20 min	6,7,11	13:48 min	12,11	19	79.17 %
2	6:45 min	6,7,11,12	6:30 min	12	19	79.17 %
3	8:33 min	6,10,11,12,2	7:15 min	11	18	75 %
4	9:20 min	4,5,6,7,10,11,12,2,3	7:34 min	no failures	15	62.5 %
5	8:46 min	4,5,7,8,11,12,1,2	7:30 min	10,8	14	58.33 %
Total					85	70.83 %

Table 4.6: Evolving 5 fixed size regions and weights of neural network controller by running evolution for 200 generations

	Clockwise		Counter Clockwise			
Training Set	Run Time	Failed Corners	Run Time	Failed Corners	Total Success	Success (in %)
1	12:57 min	4,6,11,3	10:12 min	no failures	20	83.33 %
2	8:00 min	6,7,8,10	9:10 min	8	19	79.17 %
3	7:53 min	4,5,7,12,3	10:30 min	8,7,5	16	66.67 %
4	15:01 min	7,12	13:10 min	12,10,7	19	79.17 %
5	9:10 min	7	13:41 min	12,11,7,4	19	79.17 %
Total					93	77.5 %

Table 4.7: Evolving 10 fixed size regions and weights of neural network controller by running evolution for 200 generations

	Clockwise		Counter Clockwise			
Training Set	Run Time	Failed Corners	Run Time	Failed Corners	Total Success	Success (in %)
1	12:42 min	7	12:54 min	10	22	91.67 %
2	8:45 min	7,2,3	20:20 min	1,2,3,9	17	70.83 %
3	9:28 min	7,10	12:10 min	4	21	87.5 %
4	6:29 min	no failures	6:56 min	12,11	22	91.67 %
5	15:49 min	4,5,10,12,1,3	15:25 min	12,8	16	66.67 %
Total					98	81.67 %

Table 4.8: Evolving 10 fixed size regions and weights of neural network controller by running evolution for 300 generations

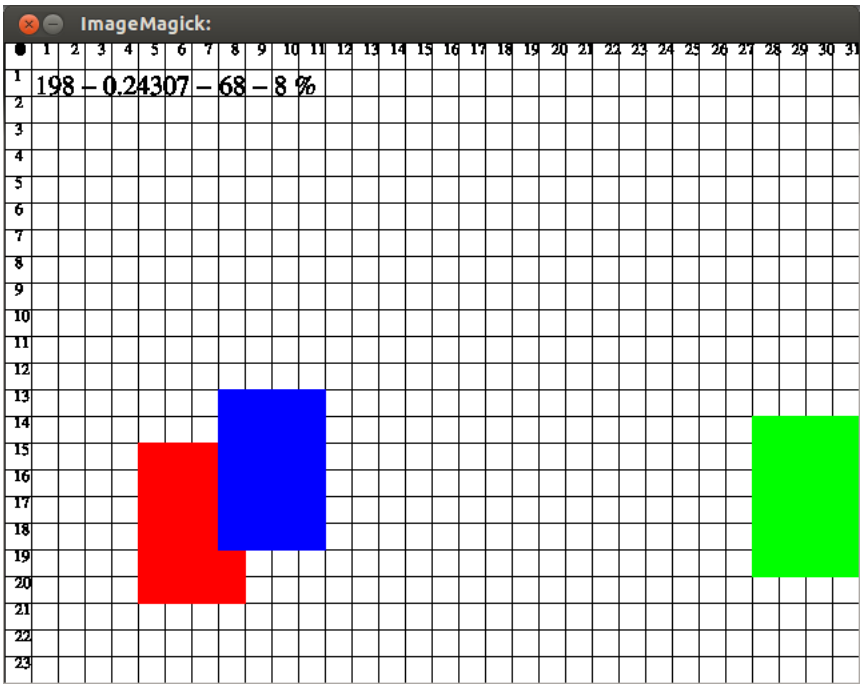

Figure 4.6: GEFE with 3 regions
The three regions evolved using training set 2 cover 8% area of the whole image, and by doing image processing on these regions only the robot could successfully navigate the test track in 20 corners out of total 24 corners (83.33% accuracy)

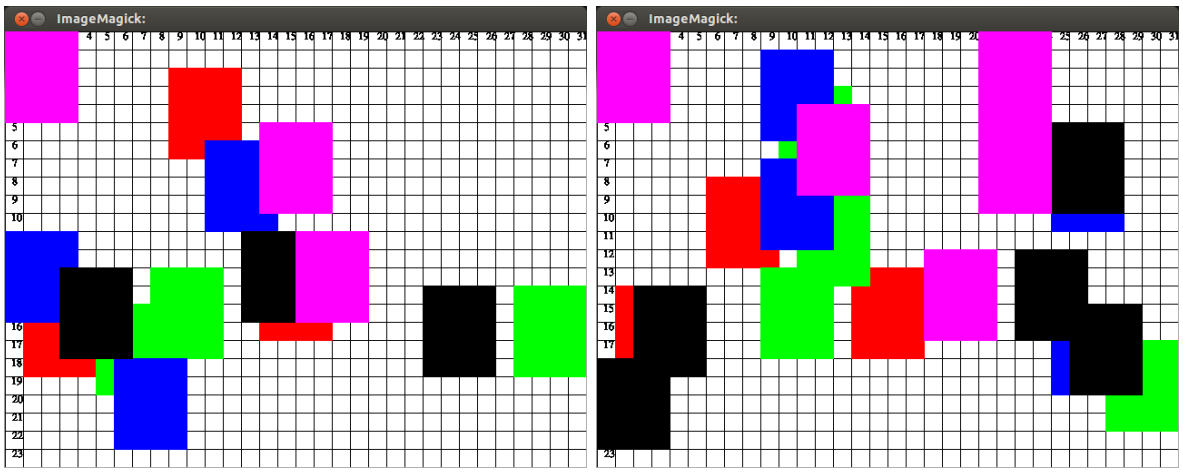

Figure 4.7: Evolved regions of all 5 training cases for 3 regions (left) and 5 regions (right). The summary of evolved regions for all 5 training cases show the regions to be more focused in central one-third portion of input image. The top and lower one-third portions are found to be less significant for image processing to identify a road from non-road.

driving decisions. The GEFE process would spread out the evolving regions in center, left and right portions of input image, such that even by doing image processing on these regions only the neural network controller can make accurate driving decisions for the robot (refer figure 4.6). In figure 4.6 it can be seen that the evolved regions cover only 8% area of the input image and can still make 83.33% accurate driving decisions. This result is comparable to the 86.66% accuracy level that was obtained when processing the whole image (Section 4.2.4).

The summary of evolved regions for all five training cases when observed for three and five regions show the regions to be focused more on central one-third portion of input image (Figure 4.7). The lower distribution of evolved regions in upper one-third and lower one-third portions of input image suggested these portions to be less significant for image processing to distinguish a road surface from non-road surface. Doing image processing on the median one-third strip was found to be sufficient to identify directions for making autonomous driving decisions. The lower one-third strip might not have made sense because most portion of it would always contain road surface in it. The upper one-third strip might not have made sense because road surface would be sparsely distributed in it. It is the median strip which would have a good variation of both road-surface and non-road surfaces. So, doing image processing in the middle one-third strip of input image, and learning from the variations in road/non-road surface in it for making autonomous driving decisions does make sense.

To test the effect of number of generations in evolutionary process the experiment comprising 10 evolving regions run for 200 generations (Table 4.7) was re-run by pushing up the number of generations to 300 (Table 4.8). It can be seen that pushing up the number of generations has a positive effect in the evolutionary process. In Table 4.7 it can be seen that running the evolution for 200 generations the robot could successfully navigate 93 corners out of 120 (77.5% accuracy). By pushing up the number of generating to 300 (Table 4.8), the robot could now successfully navigate 98 corners out of 120 (81.66%

accuracy).

4.3.4 Test for nature of regions

The procedure followed for identifying the effectiveness of variable size areas (instead of fixed one) in GEFE process, is very similar to the procedures mentioned in subsection 4.2.3 and subsection 4.3.3. The experiment can be divided into three distinct steps viz. "vision training", "evolution/training" and "test". During vision training step thirty road samples are collected, which are used to generate a color histogram library of safe road surfaces. During evolution/training step a library of training cases is generated. As in previous sub-experiments, a total of 45 training cases (15 each of forward, right and left) are used in this sub-expriment. These training cases are used for evolving the regions for image processing, and training the weights of neural network controller. The difference with the evolutionary process described in subsection 4.3.3 is that in the case of subsection 4.3.3 the evolving regions are fixed to have a size of 120 pixel width by 80 pixel height. Whereas, in this sub-experiment the regions are allowed to take any rectangular shape. An individual in the population is represented by a tuple of form $< W_1, W_1 \ldots W_m, X_1, Y_1, X_2, Y_2 \ldots X_n, Y_n >$. The part $< W_1, W_1 \ldots W_m >$ represents the weights of neural network to be trained and the part $< Y_1, X_2, Y_2 \ldots X_n, Y_n >$ represents the regions to be evolved. The number of weights in the tuple correspond to number of weights in the neural network. The sub-experiment comprising of tests for number of regions (subsection 4.3.3) gave best result for evolving 10 regions (table 4.8), hence the number of regions evolved in this sub-experiment has been chosen to be 10. So, the total number of weights in an individual tuple is 45 (10*3+3*3+3+3). Every evolving region is represented by two points and each point comprises of x and y coordinates. So, to represent 10 regions evolved, the part $< Y_1, X_2, Y_2 \ldots X_n, Y_n >$ comprises of 40 coordinates (10 regions = 20 points = 40 coordinates). The evolution process gets

	Clockwise		Counter Clockwise			
Training Set	Run Time	Failed Corners	Run Time	Failed Corners	Total Success	Success (in %)
1	7:55 min	10	8:09 min	1,8,4	20	83.3 %
2	6:41 min	7,8,10,1	7:08 min	5,4	18	75 %
3	12:22 min	5,7,11,12,2,4	14:18 min	12,11,8	15	62.5 %
4	9:34 min	5,1	7:20 min	5	21	87.5 %
5	8:19 min	4,5,6,11,2,3	9:33 min	9,8	16	66.67 %
Total					90	75 %

Table 4.9: Evolving 10 variable size regions and weights of neural network controller by running evolution for 300 generations

triggered when training library loads 10 of each training cases. Evolution is continued for 300 generation, and image processing is done on the evolved regions regions only for computing the fitness of an individual. By the end of evolutionary process, the best regions for image processing and the best neural network controller to drive autonomously are identified. The robot is then placed in autonomous mode and drives around the test track in both clockwise and counter clockwise direction for data collection. The data obtained has been given in Table 4.9.

To test the effect of allowing regions to have different size during GEFE process, the result of subsection 4.3.3 comprising 10 fixed size regions and evolution run for 300 generations (Table 4.8) is compared to the result of this sub-experiment (Table 4.9) where the 10 regions are allowed to take any size and the evolution has been run for 300 generations. Besides the regions being fixed size versus allowing them to have variable size, all the other factors between these two experiments have been kept the same. The effect of allowing/disallowing regions to have variable size during GEFE process is then analyzed by observing the performance of robot for both cases in autonomous mode. It can be seen that by allowing regions to have variable size, the evolutionary process has a

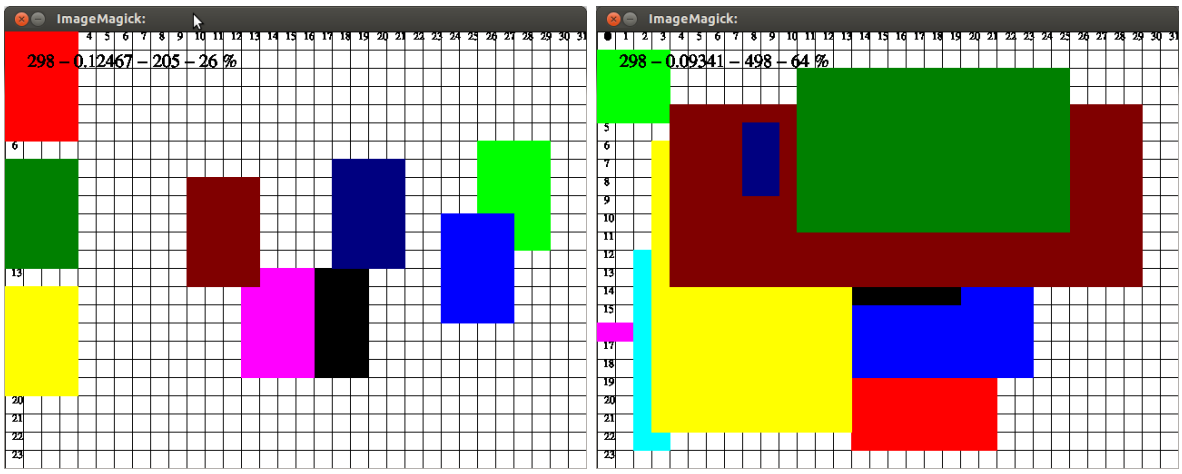

Figure 4.8: 10 regions GEFE with fixed size regions (left) and variable size regions (right) GEFE with 10 fixed size regions covered 26% of the whole image area and was able to autonomously navigate 91.67% of the test track. GEFE with 10 variable size regions covered 64% of the whole image area and was able to autonomously navigate 87.5% of the test track.

tendency to cover larger area of input image for getting better performance. Figure 4.8 shows the evolved regions that gave the best performance during autonomous run when 10 fixed size regions were evolved (refer table 4.8), along with evolved regions that gave the best performance during autonomous run when 10 variable size regions were evolved (refer table 4.9). It can be seen that evolving 10 fixed size regions covered 26% of the input image and could successfully navigate 91.67% of the test track. Whereas, evolving 10 variable size regions covered 64% of the input image but could only navigate 87.5% of the test track. This result favoring fixed size region is similar to the result obtained by authors in [17]. In [17] the authors were able to obtain 100% face recognition accuracy by processing 35.82% of uniform patches evolved, against 99.84% of face recognition accuracy when processing 36.90% of non-uniform patches were evolved.

4.3.5 Test for learning by backpropagation

The goal of this sub-experiment is to determine whether use of an evolutionary approach or backpropgation is better for evolving the weights of the neural network controller, while using GEFE for evolving regions. The experimental procedure followed is very

similar to the procedures followed in previous experiments. This sub-experiment can also be divided into three different steps viz. vision training step, evolution/training step and testing step. The vision training step and testing step are identical to the steps described in sections 4.2 and 4.3. The evolution/training step is however different. Unlike subsections 4.2.2 and 4.3.2 where evolutionary algorithm has been used for evolving the weights of neural network, this sub-experiment uses backpropagation to train the weights of its neural network controller. The procedures for generating a library of training cases and evolving regions for image processing are identical to subsections 4.2.2 and 4.3.2

Backpropagation is a common method for training the weights of a neural network. It is an abbreviation for "backward propagation of errors" and uses the errors at the output nodes of a neural network to update its weights, such that the errors get minimized. As, the process involves computing error, the desired outputs for a given set of inputs must be known before hand. With every iteration of backpropagation, the weights in the neural network get updated such that for any given input, correct output gets generated.

In this sub-experiment, 10 regions have been evolved by running evolutionary algorithm for 75 generations, and backpropagation has been run for 40 iterations to train the weights of a neural network controller. The number of generations is chosen to be 75 because for every generation, every individual in the population goes through backpropagtion to train its weights. So running evolutionary algorithm for larger number of generations would take a long time. Image processing is done on training set images at coordinates indicated by the evolved regions, and the computed road probability values are passed as inputs to the neural network. The output driving direction generated by the network is compared against the stored driving direction for that training set image to compute the error. This error is used by backpropagation to update the weights of the neural network. The best neural network and evolved regions are used as the neural network controller during the testing step and data collection. During the testing step, it was found that the neural network controller was totally unsuccessful in driving the

robot autonomoulsy on test track. The robot either moved in circles or moved in the forward direction taking it off the track.

To determine the effectiveness of evolutionary approach versus backpropagation for training the weights of a neural network controller while using GEFE for evolving regions, the results obtained in this sub-experiment is compared with the results in table 4.8. In table 4.8, evolutionary approach has been used to train the weights of neural network controller while GEFE has been used to evolve 10 fixed size regions. The results show 81.67% success rate in driving the robot autonomously on test track, in comparison to 0% success rate for this sub-experiment. So, it can be concluded that while using GEFE to evolve the regions for image processing, use of an evolutionary algorithm is better than use of backpropagation to train the weights of neural network controller. The reason behind this could be the efficient generalization nature of neural network and bacpropagation. As backpropagation is good at mapping the neural network‘s inputs to outputs, no matter which regions are chosen for image processing, it adjusts the neural network's weights to cope to those regions. So, every combination of regions produced similar result, thus severely hampering the GEFE process of evolving regions for image processing.

4.3.6 Results

Analyzing the results from subsections 4.3.3, 4.3.4 and 4.3.5 it can be seen that GEFE can be successfully applied to the task of autonomous road following. The best case obtained when GEFE is used for evolving regions for image processing is 91.67% accuracy in successfully driving the robot on test track in autonomous mode (training set 1 and 4 in table 4.8). This result is equal to the 91.67% best accuracy level that was obtained when image processing is done on the whole input image (training set 1, 4 and 5 in table 4.2). The best “average success rate” using GEFE was obtained when 10 fixed

size regions were evolved for 300 generations. This average success rate (81.67% in table 4.8) is comparable to the average success rate which was obtained when the whole input image was proceed for making autonomous driving decisions (86.67% in table 4.2).

In the sub-experiment comprising of tests checking the effect of number of regions, it could be seen that number of regions evolved, and successful navigation are not linearly related. The robot in an autonomous mode performed poorly for 2 regions (57.5% success), but well for 3 regions (78.33% success). The result for 5 regions was also poor (70.83%), whereas the result for 10 regions was similar to that of 3 regions (77.5%). Increasing the number of generations for evolution however, had a positive effect on the performance of robot during autonomous mode. When 10 regions were evolved for 200 generations, 77.5% successful navigation was observed. But by increasing the number of generations from 200 to 300, the percentage of successful navigation increased from 77.5% to 81.67%. In the summary image of evolved regions (Figure 4.7), it could be seen that majority of the regions were concentrated in the median one-third portion of the input image. I speculate this occurred because, a major percentage of lower one-third portion always consisted of road surface in it, whereas the upper one-third portion occasionally consisted of road surface in it. So, it is the median one-third strip that contained maximum variations in road/non-road surfaces, and the GEFE approach could successfully place the evolved regions in this strip.

In the sub-experiment testing the nature of evolving regions for GEFE, it was found that fixed size regions performed better than variable sized regions. When 10 regions were evolved using both fixed sized regions (120 pixel width by 80 pixel height) and variable sized regions for 300 generations, it was found that in the sub-experiment with fixed sized regions the robot could successfully navigate 81.67% of the test track (table 4.8). In the sub-experiment comprising of variable sized regions only 75% success in autonomous navigation could be achieved (table 4.9). Moreover, variable sized regions were found to have a tendency of covering larger area of input image to achieve better

fitness (Figure 4.8). In the sub-experiment with 10 regions the best training set for variable regions covered 64% of input image while successfully navigating 87.5% of the test track (table 4.9). On the other-hand the best training set for 10 fixed size regions covered only 26% area of the input image while successfully navigating 91.67% of the test track (table 4.8).

In the sub-experiment testing the effectiveness of evolutionary approach versus backpropagation for training the weights of neural network controller while using GEFE for evolving regions, it was found that evolutionary approach outperformed backpropagation (as implemented in subsection 4.3.5). Evolutionary approach was found to be both fast and successful in driving the robot autonomously. Using evolutionary approach for 300 generations to train the weights of neural network with 10 inputs, the neural network could successfully navigate 81.67% of the test track. On the other-hand use of backpropagation to train the same network was totally unsuccessful in driving the robot in test track. The reason behind this could be the generalization nature of neural networks and backpropagation. No matter which reasons are used for image processing, backpropagation adjusts the weights of neural network to cope with those reasons, thus producing similar fitness values and hampering the GEFE process.

Chapter 5: Summary and Conclusions

The goal of the research was to test the feasibility of GEFE in autonomous road following. As image processing is computationally intensive in nature, requiring significant processing time and main memory, any reduction in area of the image to be processed for decision making is advantageous. In this research, GEFE has been used to evolve regions for image processing in an input image. Performance of the robot in autonomous mode when image processing is done only using the evolved regions, has been analyzed to determine the success or failure of GEFE approach in autonomous road following.

Dividing the input image into static 32 regions covering the whole image, and doing image processing on those regions for making driving decisions allowed the robot to successfully navigate 86.67% of the testing track. The reasons behind the robot failing to successfully navigate the whole test track could be due to flaws in image processing, flaws in learning the weights of the neural network, the training set being insufficient, vision training library being insufficient, image auto-correction by the robot phone, or blurring of images (subsection 3.3.4 discusses these potential issues in more detail). Similar levels of accuracy were obtained when GEFE was used to evolve 10 regions for 300 generations. Using 10 evolved regions the robot was able to successfully navigate 81.67% of the test track. In the best case, the 10 evolved regions covered only 26% of the whole image while still successfully driving the robot on 91.67% of the test track (refer table 4.8 training set 1).

The relationship between number of regions for GEFE, and successful autonomous navigation was found to be non-linear. In the sub-experiments where 2, 3, 5 and 10 regions were evolved, the rate of success in autonomous navigation increased when going from two to three regions, then decreased for five regions, and again increased for ten regions. The corresponding success for two, three, five, and ten regions were 57.5%, 78.33%, 70.83% and 77.5% respectively (table 4.4 to table 4.7). Increasing the number of generation had a positive impact on the performance of the robot in autonomous mode.

Pushing the number of generations for evolving 10 regions from 200 to 300 increased the success in autonomous navigation from 77.5 % to 81.67% (table 4.7 and table 4.8). It was found that for autonomous road following, the GEFE process aligned evolved regions in median one-third horizontal strip of input image (figure 4.7). I speculate this occurred because maximum variations in road and non-road surfaces can be observed in this strip (subsection 4.3.3). Evolving fixed sized regions was found to perform better than variable sized regions (Section 4.3.4). It was also found that while using GEFE to evolve the regions for image processing, use of an evolutionary algorithm to train the weights of neural network controller outperforms use of backpropagation for training the weights (Section 4.3.5).

Based upon the data collected and results analyzed, it can be concluded that GEFE can be successfully applied to the task of autonomous road following in a controlled environment. When fixed sized regions are evolved for significant number of generations, GEFE is found to place them at positions such that sufficient information for autonomous navigation can be obtained by processing image on those regions only. This information can then be used in making decisions for autonomous road following. While GEFE has been used to evolve the regions for image processing, an evolutionary algorithm has to be used to train the weights of the neural network controller. Use of the evolutionary approach is found to be both fast and successful in training the weights of the neural network. When the final regions for image processing have been identified using GEFE, and the evolutionary algorithm has trained the weights of the neural network controller, image processing can then be done on those regions only for autonomous navigation, instead of the whole input image. This saves a significant amount of CPU time and main memory, which can be used for doing other important tasks.

Chapter 6: Future Work

The research presented in this thesis presents support for several important conclusions, but the presented approach has room for improvements and suggests a number of additional research projects.

The backprojection approach of image processing used in this research is very susceptible to changes in lighting conditions. Slight variation in lighting condition can give a different result. Implementation and testing of image processing approaches which are less susceptible to changes in lighting conditions (e.g. Local Binary Pattern, HSV, etc.) can be a good extension to this research.

The research currently uses backprojection to calculate the road probability values in an input image. Decisions for autonomous navigation are solely based upon these calculated road probability values. Incorporating other image processing techniques in addition to probability values from backprojection in the decision making process, might make the driving decision more accurate. Some of the other promising image processing approaches that can be incorporated include edge detection and texture detection.

The machine learning representation used in this research is a neural network. Though neural networks have been shown to be effective in learning the test problem of autonomous road following in this research, various other approaches for machine learning can be implemented. Some of the other promising techniques that are wroth testing include decisions trees and fuzzy logic.

The experiments done in this research are all done indoors. As the image processing for identifying road from non-road becomes more robust, a natural extension to this research can be doing the experiment outdoors.

Bibliography

[1] Defence Advanced Research Project Agency. Darpa. http://www.darpa.mil/.

[2] John Robert Anderson, Ryszard Stanisław Michalski, Jaime Guillermo Carbonell, and Tom Michael Mitchell. *Machine learning: An artificial intelligence approach*, volume 2. Morgan Kaufmann, 1986.

[3] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[4] Thomas Back, David B Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[5] Defence Advanced Research Project Agency (DARPA). Darpa grand challenge. http://archive.darpa.mil/grandchallenge05/index.html, 2005.

[6] Defence Advanced Research Project Agency (DARPA). Darpa urban challenge. http://archive.darpa.mil/grandchallenge/overview.html, 2007.

[7] G.N. DeSouza and A.C. Kak. Vision for mobile robot navigation: a survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(2):237–267, Feb 2002.

[8] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. springer, 2003.

[9] Yinghua He, Hong Wang, and Bo Zhang. Color-based road detection in urban traffic scenes. *Intelligent Transportation Systems, IEEE Transactions on*, 5(4):309–318, Dec 2004.

[10] Frédéric Kaplan, Pierre-Yves Oudeyer, Enikö Kubinyi, and Adám Miklósi. Robotic clicker training. *Robotics and Autonomous Systems*, 38(3):197–206, 2002.

[11] Lei-Jian Liu, Mingwu Ren, Yulong Cao, and Jingyu Yang. Color road segmentation for alv using pyramid architecture. volume 2028, pages 396–404, 1993.

[12] Stephen Marsland. *Machine learning: an algorithmic perspective*. CRC Press, 2011.

[13] Silvano Martello and Paolo Toth. Knapsack problems.

[14] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[15] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003.

[16] Joseph Shelton, Gerry Dozier, Kelvin Bryant, Joshua Adams, Khary Popplewell, Tamirat Abegaz, Kamilah Purrington, Damon L Woodard, and Karl Ricanek. Genetic based lbp feature extraction and selection for facial recognition. In *Proceedings of the 49th Annual Southeast Regional Conference*, pages 197–200. ACM, 2011.

[17] Joseph Shelton, Gerry V Dozier, Kelvin S Bryant, Lasanio Smalls, Joshua Adams, Khary Popplewell, Tamirat Abegaz, Damon L Woodard, and Karl Ricanek. Comparison of genetic-based feature extraction methods for facial recognition. In *MAICS*, pages 216–220. Citeseer, 2011.

[18] Joseph Shelton, Kaushik Roy, Brian OConnor, and Gerry V Dozier. Mitigating iris-based replay attacks. *International Journal of Machine Learning & Computing*, 4(3), 2014.

[19] David Silver, J Andrew Bagnell, and Anthony Stentz. Applied imitation learning for autonomous navigation in complex natural terrain. In *Field and Service Robotics*, pages 249–259. Springer, 2010.

[20] Keith Sullivan and Sean Luke. Real-time training of team soccer behaviors. In *RoboCup 2012: Robot Soccer World Cup XVI*, pages 356–367. Springer, 2013.

[21] A. Takahashi, Y. Ninomiya, M. Ohta, and Koichi Tange. A robust lane detection using real-time voting processor. In *Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on*, pages 577–580, 1999.

[22] C. Tan, Tsai Hong, T. Chang, and M. Shneier. Color model-based real-time learning for road following. In *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*, pages 939–944, Sept 2006.