OBJECT DETECTION AND LOCALIZATION USING APPROXIMATE NEAREST NEIGHBOR SEARCH: RANDOM TREE IMPLEMENTATION

By

Esubalew Tamirat Bekele

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2009

Nashville, Tennessee

Approved:

Professor Richard Alan Peters II

Professor Don Mitch Wilkes

To my beloved mother, Atale Belay

To my understanding sister, Wossenie Tsegaye

and

To my wonderful brother, Embiale Zemedie

for investing in me to be where I am right now

ACKNOWLEDGEMENT

I would like to take this chance to thank Dr. Peters II, Richard Alan for his continued effort in this thesis in particular and in my progress towards this degree in general. His confidence in me, his encouragement and passion that he has for his work are among the various things that need to thank him for and am grateful for. I would like to thank him for letting me try his code for epipolar geometry stereopsis.

I am also very thankful to Dr. Mitch Wilkes for his permission and support to help me use their idea of random tree implementation for performance analysis. He was such a great help in the successful completion of this thesis.

I thank also Jonathan Hunter for his kind and continuous support to give me a lot of ideas whenever I want them. He was there whenever I wanted to talk to him.

I forward my heartfelt appreciation and thanks to Flo. I am so grateful to her enduring support in material and anything I want. She was so encouraging and helpful. I am so happy to get to know her.

Finally, I would like to thank all other colleagues, friends (Especially Abiy Tekola, His wife Abigya and Birhan Woldegiorgis, my esteemed friends, for their encouragement and being there whenever I wanted them) for their support and encouragement.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENT iii

LIST OF TABLES vii

LIST OF FIGURES viii

LIST OF EQUATIONS xi

LIST OF ACRONYMS xii

CHAPTER

I. INTRODUCTION 1

1.1 Overview 1

1.2 Objective of the Thesis 4

1.3 The Proposed system 5

1.4 Structure of the Thesis 6

II. PERCEPT ACQUISITION AND PRE-PROCESSING 7

2.1 The test bed (ISAC) 7

2.1.1 The cameras 7

2.1.2 The Lenses 11

2.1.3 The Camera Mounts 11

2.1.4 The Frame Grabber 12

2.2 Pre Processing 14

2.2.1 Digital Image Processing using OpenCV 14

2.2.2 Noise Filtering 15

2.2.3 Background Subtraction 18
III. FEATURE EXTRACTION 20
3.1 Biological Inspiration 20
3.2 Background of the features selected 24
3.3 Rationale For the use of High Dimensional Features 26
3.4 The perceptual features used 28
3.4.1 The color probability density function (pdf) 29
3.4.2 The texture component 31
3.5 Extraction of the features from the images 32
IV. PERCEPTUAL LEARNING AND SEGMENTATION 36
4.1 Perceptual Learning 36
4.1.1 The Pure Nearest Neighbor Algorithm 36
4.1.2 The Distance (Similarity) Measure 41
4.1.3 The Approximate Nearest Neighbor Algorithm (The Random Tree) 42
4.2 Segmentation 50
4.3 Performance Measures 52
V. STEREOPSIS 54
5.1 Camera Calibration 55
5.2 Co-planar Steriopsis 58
VI. PROPOSED EXPERIMENTS 62
6.1 Experiment I 64
6.2 Experiment II 65
6.3 Experiment III 66
6.4 Experiment IV 67

VII. EXPERIMENTAL RESULTS AND DISCUSSIONS 68
7.1 Training Results (Experiment I Results) 68
7.2 Testing Time Results (Experiment II Results) 74
7.3 Testing Accuracy Results (Experiment III Results) 79
7.4 Stereopsis Results (Experiment IV Results) 84
VIII. CONCLUSIONS AND FUTURE WORK 86
8.1 Conclusions 86
8.2 Future Work 87
REFERENCES 89
APPENDIX 94
SELECTED SEGMENTED IMAGES 94
KNN 94
Left 94
Right 107
RANDOM TREE 121
Left 121
Right 134

LIST OF TABLES

Table Page

Table 1: the basic random tree data structure. 44

Table 2: List of the experimental objects that are used in the project 62

Table 3: Platform Specification of the test beds for experiment III 66

Table 4: Number of training images used and training vectors extracted out of them for the old set of training images 69

Table 5:Number of training images used and training vectors extracted out of them for the new set of training images 70

Table 6: Summary of the comparison of the pure nearest neighbor algorithm with approximate random tree implementation 83

Table 7: Theoretically computed 3D coordinate ranges 84

Table 8: Experimentally found 3D coordinate ranges for both pure and random tree implementations 85

LIST OF FIGURES

Figure Page

Figure 1: General Structure of the proposed system..... 5

Figure 2: The two camera modules, the lenses and the mounts constitute ISAC's eyes.... 8

Figure 3: The Sony XC-999 Color Video Camera Module..... 9

Figure 4: The inner parts of the camera. 9

Figure 5: Single camera Horizontal and Vertical FOVs..... 10

Figure 6: Working Field provided by the stereoscopic cameras of ISAC 10

Figure 7: The Sony fixed focus 6mm lens..... 11

Figure 8: The vertical tilt of -71^0 and a horizontal pan of 0^0 with the world frame of reference for the vision system (not drawn to scale) 12

Figure 9: The original image and the smoothed image (not in their original size)..... 17

Figure 10: Original smoothed image (Left) and its background subtracted version (right) 19

Figure 11: The human receptor fields with in a tissue sample (right) and a typical CCD arrays of a camera are shown here. 21

Figure 12: Parts of the human eye (13)..... 21

Figure 13: distribution of rods and cones around in the human retina. Boxes at the top illustrate the appearance of cross sections through the outer segments of the photoreceptors at different eccentricities. 22

Figure 14: The Visual pathway from eyes to primary visual cortex, of a human brain. .. 23

Figure 15: A Conical Model of the HSV color space. The hue is the angle that varies from 0^0 to 360^0. The saturation and the value both range from 0 to 1. 30

Figure 16: A 15 x 15 ROI of an image from which a single feature vector is calculated out (Note: the figure is magnified from its original small size to see the details. 33

Figure 17: A typical set of training feature vectors. The spikes at 10,001 are the texture measures and the rest are the pdf of the histograms. 34

Figure 18: Plot of a feature vector that shows how sparse the vector is. It has dominant non-zero values near the maxima of the pdf of the objects color. 35

Figure 19: A segmented image that is before the re-labeling with the recognized objects and their corresponding centroids. (Centroids are superimposed for completeness. However, they are calculated after re-labeling.)..... 51

Figure 20: A segmented image that is after the re-labeling with the recognized objects and their corresponding centroids. 52

Figure 21: Contingency table or Confusion matrix of the four possible outcomes (52). . 53

Figure 22: The 30 images that are used in the calibration process 55

Figure 23: Example corners extracted from example image of the check board..... 56

Figure 24: Re-projection errors in the computation of the camera parameters 58

Figure 25: Geometrical configuration of the coplanar image planes, the respective frames of reference and object projection to the image planes..... 59

Figure 26: Experimental Setup of center of the head of ISAC and a table in front of him (not drawn to scale)..... 61

Figure 27: The Experimental objects that are used by this project..... 63

Figure 28: plot of the training times of the old (36 training images) and new image set (54 training images) versus number of training vectors..... 71

Figure 29: training times in seconds versus minimum number of training vectors per impure leaf node 72

Figure 30: Plots of testing time of the pure nearest neighbor algorithm with number of testing vectors for the right and left images (top) and with number of training vectors (bottom) TrIm represents Training Image and TsIm represents Testing Image in the legends 74

Figure 31: Plots of testing time of the random tree for both left and right images with number of testing vectors (above) and with number of training vectors (below) TrIm represents Training Image and TsIm represents Testing Image in the legends..... 75

Figure 32: Plots of testing time of the pure algorithm with number of testing vectors and with number of training vectors. This is the result of program running on Sally..... 77

Figure 33: Plots of testing time with number of testing vectors and with number of training vectors that run on Sally. 78

Figure 34: Testing accuracy plots and ROC curves for individual test objects of the pure nearest neighbor search method. 79

Figure 35: Accuracy of individual test objects with number of training images and ROC curves. 81

Figure 36: Accuracy plots of the random tree implementation with the minimum number of training vectors in an impure node and the corresponding ROC curves for individual test objects 82

Figure 37: 3D plots of the coordinates of the 352 centroids of the detected experimental objects (dimensions are in mm) 84

LIST OF EQUATIONS

Equation Page

Equation 1: Camera Color Model 25

Equation 2: Hue, Saturation and Value computation 30

Equation 3: Laplacian 31

Equation 4: Laplacian kernel 31

Equation 5: Texture measure 32

Equation 6: approximation function 38

Equation 7: Algorithm of pure nearest neighbor search 39

Equation 8: General Euclidean distance 39

Equation 9: General Minkowski Metric Formula 41

Equation 10: Euclidean distance for the approximate method 42

Equation 11: Training algorithm for the random search tree 46

Equation 12: Search algorithm for the random search tree 48

Equation 13: ROC calculation equations 53

Equation 14: Coplanar Stereopsis Equations 60

LIST OF ACRONYMS

Acronym Page

2D: Two Dimensional 14

3D: Three Dimensional 14

AI: Artificial Intelligence 1

ANN: Approximate Nearest Neighbor 84

BSD: Berkley Software Distribution Licence 13

Cal Tech: California Institute of Technology 11

CCD: Charge Coupled Devices 15

CIS: Center for Intelligent Systems 61

CRL: Cognitive Robotics Laboratory 61

CV: Computer Vision 13

CVAUX: Computer Vision Auxiliary Library 13

CVCAM: Computer Vision Camera Library 13

CXCORE: Core Datastructures Library 13

DLL: Dynamically Linked Libraries 12

fMRI: functional Magnetic Resonance Imaging 24

FOV: Field Of View 9

HIGHGUI: High Graphical User Interface Library 13

HSI: Hue, Saturation, Intensity 26

HSL: Hue, Saturation, Lightness 26

HSV: Hue, Saturation, Value 26

IplImage: Image Processing Library Image Data Structure 12

ISAC: Intelligent Soft Arm Control 4

JPEG: Joint Photographic Experts Group 12

KNN: K-Nearest Neighbors, K accounts for the K nearest neighbors to a query point ...94

LGN: Lateral Geniculate Nucleus 23

ML: Machine Learning Library 14

MT: Middle Temporal 24

OpenCV: Open Computer Vision Library 12

RGB: Red Green Blue 25

ROC: Receiver Operating Characteristics 51

ROI: Region Of Interest 32

CHAPTER I

INTRODUCTION

1.1 Overview

Much work has, and continues to be done toward making robots to think, to behave, and to act like human beings. There are two branches of artificial intelligence concerning this: Strong AI and Weak AI (1). The Weak AI group says that machines have to think and act rationally, where as the Strong AI camp says that machines must think and act humanly. Both camps have their rationale. The first raises the current inability of machines to represent or to process the large amounts of data that is involved. The second camp raises future possibilities of efficient algorithms and representations and most importantly the discovery of how the human brain works. Humanoid robotics researchers tend to be among the second group.

Humanoid robotics research focuses on perception, learning and control where in each part; it tries to mimic the human counterparts. There have been many research projects on robots (2) (3) that are based on neurobiological research results of humans. Current research clearly demonstrates that it is the human brain that represents (stores) and processes the bulk of information related to percepts, learning and control. (There is some evidence that a muscle can store some information related to movements. Hence

not only the brain is responsible for storage and processing.) Humans learn from experience (3). A child learns much from his or her parents. With respect to visual learning, the parents might show the child an object, say the object's name, what it does etc. The child's brain stores the object's features, and their relations to the object's other affordances (though, the representation for the storage is not fully understood). This is called *supervised learning*. Sometimes, a child learns about objects or situations by himself. It will try various actions on the object and based on its reaction the child will modify his actions. That is partially *supervised learning*. On some occasions, the child might learn without any feedback at all and that is *unsupervised learning*. Of the neurobiological sciences, cognitive sciences are of particular interest to humanoid robotics.

The perceptual learning process, especially visual learning involves a lot of data and hence requires high performance machines or efficient representation and processing. It also requires focusing on what is relevant to the current task.

Ideally, robots would perceive (visually) environments at the level of the human visual system so as to emulate human-like sophisticated reasoning and control of actions. However, this is a big problem that is not solved to date. Among the reasons for this failure are the limited processing and storage of current machines, the lack of efficient representations (features), lack of very efficient algorithms, and the infinitely large possibilities in complex environments. Perhaps most importantly, the lack of sufficient neurological knowledge due to the complexity of the human visual system makes engineering approaches to the problem somewhat *ad hoc*. This is what is called the visual

discrepancy problem (2). It is not possible to encode all possibilities in the robot and have the robot perceive as humans perceive in a dynamic and complex environment.

To operate in complex and dynamic environments, therefore, it makes sense intuitively to have the robots form percepts on their own by learning in supervised, semi-supervised, and completely unsupervised ways, in contrast to the preprogramming. Based on those self-formed percepts, the robot should be able to classify future perceptions. The robot should first learn the objects' features, store them in some form, and then use them to segment future images and recall the learned objects on its own.

To this end, a typical robotic vision system should meet the following criteria. It should learn, form, and store percepts from its past experience. It should also adapt to dynamic environments since, for example, a robot might observe the same object in different lighting conditions, at different places, or in different poses (position and orientation with respect to the robot). It should focus on the most relevant part of the visual percept that is linked to the current task at hand. This work will incorporate two of the three criteria.

The proposed system is able to learn colors and textures from its visual experience using the approximate nearest neighborhood algorithm using a random search tree for fast response. The system also is able to adapt to dynamic situations. This is achieved by learning a large sample for individual objects in different conditions so as to help the classifier perform better in different conditions.

1.2 Objective of the Thesis

The main objective of this thesis is to build a perceptual system that is capable of recognizing objects whose features have been learned previously and capable of distinguishing a new object. The system was developed for a humanoid robot that acts in a complex and dynamic environment like ISAC, the Vanderbilt Humanoid.

1.3 The Proposed system

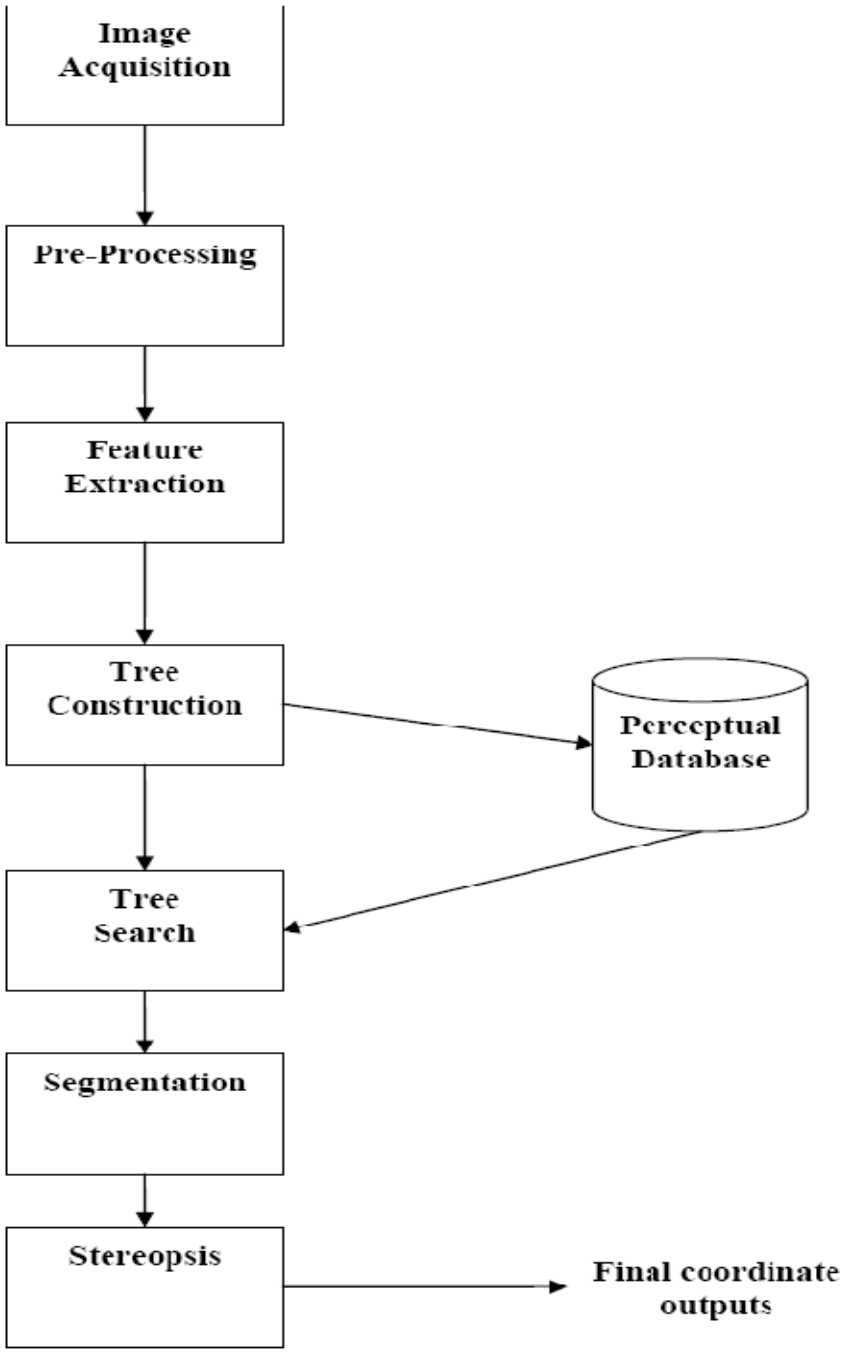


Figure 1: General Structure of the proposed system

1.4 Structure of the Thesis

The current chapter was overview of the system that was implemented in this work, the criteria that were to be met, and the general goals of the thesis.

Chapter 2 is a discussion of the percept acquisition and preprocessing steps. The process of image capturing together with the details of capturing hardware and software are presented. The preprocessing noise removal and background subtraction methods that were used in this work are also discussed.

Chapter 3 is a discussion of the feature extraction step. The very high dimensional feature vector representation and how it is extracted from the images is dealt in a greater depth. The chapter also includes the rationale to use a very high dimensional feature space.

Chapter 4 is devoted to the description of the approximate nearest neighbor algorithm used in this work and its theoretical comparison with the pure method. The chapter introduces the new random tree implementation discusses how the search trees are constructed and how the search trees are used to find the nearest neighbor.

Chapter 5 is the presentation of the approximate coplanar stereopsis technique that was used. It explains the assumptions made to use this approximation and shows how stereopsis was implemented.

Chapter 6 describes the four experiments that were performed and their setups. Chapter 7 presents the experimental results along with a discussion of the results. Chapter 8 concludes the thesis.

CHAPTER II

PERCEPT ACQUISITION AND PRE-PROCESSING

2.1 The test bed (ISAC)

The test bed for this project is the Vanderbilt humanoid robot, ISAC, an acronym for Intelligent Soft Arm Control. ISAC is composed of many parts from perception to cognition and control both in hardware and software domains. The main focus of this project is perception and part of cognition. The robot is equipped with two stereoscopic cameras with mounts that are capable of moving the cameras both horizontally and vertically. ISAC is also equipped with two soft arms that are actuated by muscle like air pressured rubbers (McKibben artificial muscles).

2.1.1 The cameras

The two “eyes” of ISAC are Sony XC-999 CCD Color Video Camera Modules (4). The lenses are two VCL-06S12XM fixed focus lenses (5). The combination of the camera module and the lenses together with the moving mount constitute ISAC’s two eyes as shown in the figure below.

Figure 2: The two camera modules, the lenses and the mounts constitute ISAC's eyes

The cameras has a native 768 (H) x 494 (V) elements with a sensing area of 6.4 x 4.8 mm. In this project, however, a 620 x 470 resolution is used, due to the original configuration on the ISAC system. This is partly important for the reduction of the number of pixels (from 379392 to 291400, which is 23.2% reduction in data), and hence the data to be processed, without harshly affecting the picture quality. The camera tube and the inner parts of the camera are shown below.

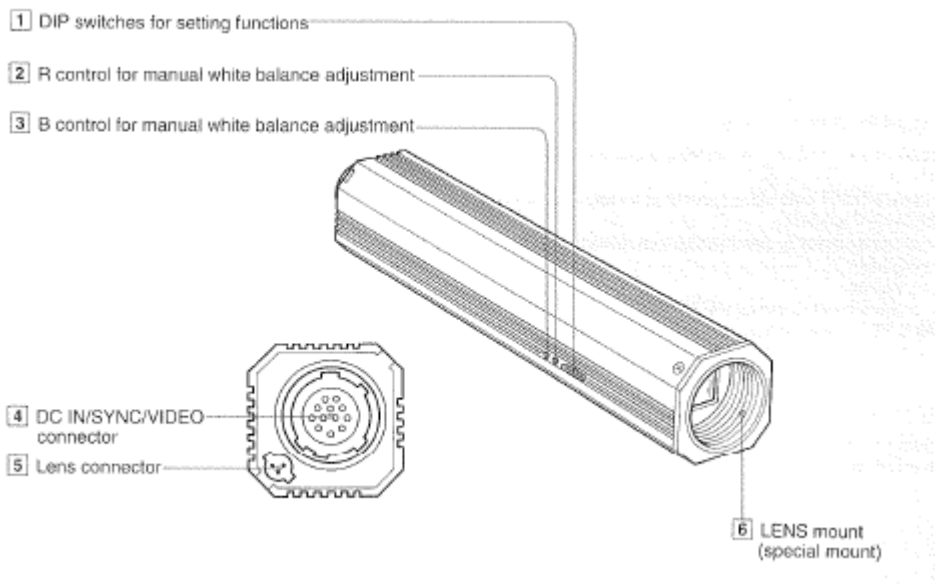


Figure 3: The Sony XC-999 Color Video Camera Module (4)

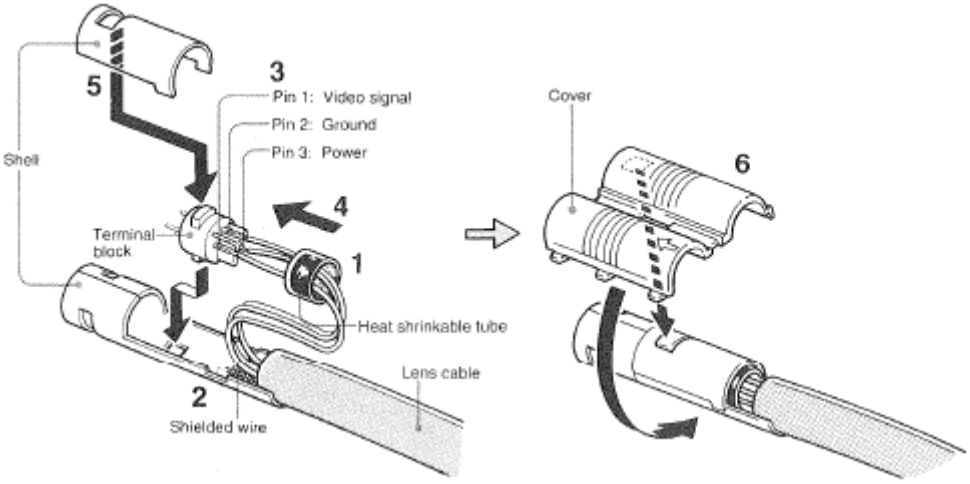


Figure 4: The inner parts of the camera. (4)

As shown in the figure above, most of the space is occupied by the cables and their shield. Each camera has horizontal field of focus (FOV) of approximately 55.77^0 and vertical FOV of 42.78^0 as shown in Figure 5 (6).

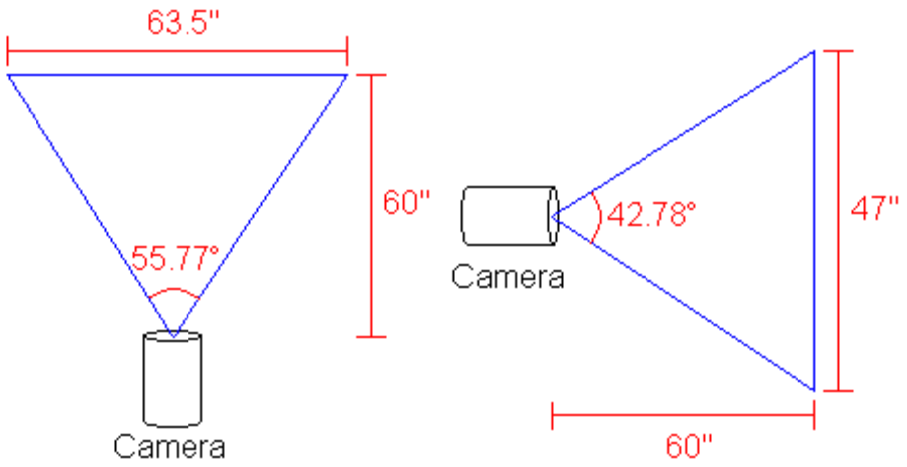


Figure 5: Single camera Horizontal and Vertical FOVs (6)

The objects that are used in this project must be in the range provided by the limitations in the horizontal and vertical FOVs. The two cameras are located at a head span of 11 inches. This provides a visible area of 47 inches high and 51 inches wide in front of them as shown in the figure below.

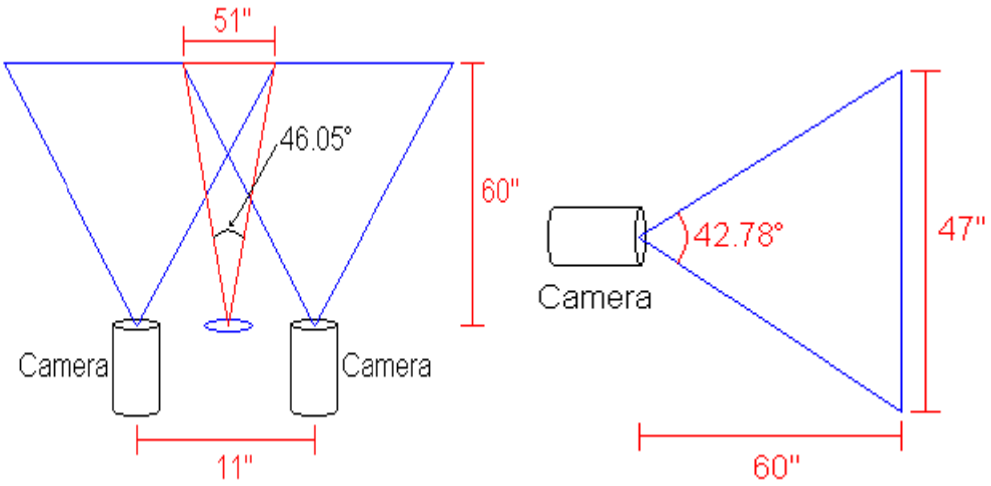


Figure 6: Working Field provided by the stereoscopic cameras of ISAC (6)

The above distances are computed for a distance of 5 inches from ISAC's eyes. The actual stereopsis calculations that are used for this project are presented in the Stereopsis chapter of this thesis.

2.1.2 The Lenses

The camera can be operated with either 6mm or 12mm focal length lenses. The lenses that are actually in use are the 6mm fixed focus lenses. The focal length has been verified (6.38 mm) with best accuracy using the California Tech (Cal Tech) Matlab calibration toolbox (7). The lens is shown in the figure below together with its specifications.

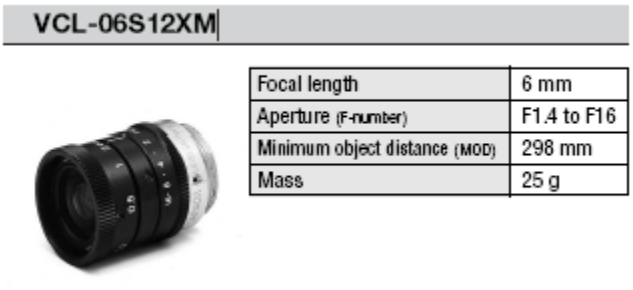


Figure 7: The Sony fixed focus 6mm lens (5)

2.1.3 The Camera Mounts

The cameras are mounted on Directed Perception Model PTU-D46-17.5 miniature Pan-Tilt units (8). These pan-tilt control mounts are capable of fast tracking speed of up

to 300^0/second, a resolution of 0.003^0. For this research, a tilt of -71^0 measured below a horizontal line along the axis of the cameras and a 0^0 horizontal pan were used for both cameras. The images were taken by presenting the objects on a table in front of ISAC in its working space provided by the horizontal and vertical FOVs. This enabled accurate results using an approximate co-planar Stereopsis.

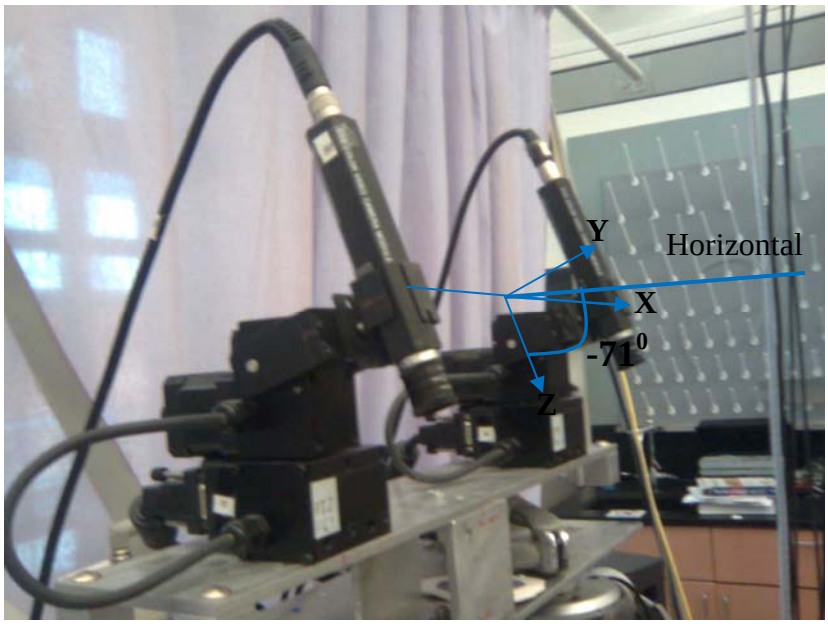


Figure 8: The vertical tilt of -71^0 and a horizontal pan of 0^0 with the world frame of reference for the vision system (not drawn to scale)

2.1.4 The Frame Grabber

The input video signal that is converted to voltages by the sensor array is then transferred to a video grabber that works with the cameras to convert the input analog signal in to a digital form and store the images. The frame grabber that is used in this project is the imagination vision systems PXC200 Precision Color Frame Grabber (9). This grabber has a high color resolution and can connect up to four camera inputs. It has

three image capture modes: software-initiated grab, triggered grab, and continuous acquire mode. The first mode is used in this project. The grabber comes with C DLLs that are used to initialize the grabber by setting the height, width, resolutions, and brightness and are used to allocate the buffers for the incoming digital image data. Each frame is then copied from the buffers to an IplImage data structure that is provided by the OpenCV library (10). The image frames are then stored to disk as JPEG (Joint Photographic Experts Group) image files.

For this project, two sets of images which contain 74 and 90 images were used for the experiments (they are called the old set and the new set from now on). The old set was the primary set that is used for most of the experiments. The world coordinates of the setup were not recorded at the time of collection of the old set. That is one of the reasons why the new set was used. The other compelling reason to take a new set was to see the performance of the proposed system beyond 6 set of training images (which was the limit of the old set) to 9 set of training images for each of the test objects. The detail of this is in the experiments, and discussion and results section of this thesis.

2.2 Pre Processing

2.2.1 Digital Image Processing using OpenCV

OpenCV is an acronym that stands for the Open Computer Vision open source library which is primarily supported by Willow Garage. The library was originally developed by Intel and is free under a BSD (Berkley Software Distribution license) (11). It is a cross-platform library that runs on Windows, Mac OS X, Linux, VCRT (Real-Time OS for smart camera) and others.

The library is composed of libraries like CV, CVCAM, HIGHGUI, CXCORE, and the outdated CVAUX. The main image processing functions are contained in the CV library. CVCAM is a library that contains functions related to camera operations like initializing cameras, image capture from camera, saving to file, etc. The HIGHGUI library is a library of helpful functions that are used to create windows graphical user interfaces very easily and quickly. The main supporting library to the CV library is the CXCORE library. It contains core data structures and algorithms that are used in processing images using the CV library. The OpenCV library also adds a machine learning library called MLL recently. While most of the OpenCV implementations are C functions the MLL library is a C++ implementation of most of the common machine learning algorithms that are useful to digital image processing. The CVAUX library is an outdated library that is a collection of auxiliary image processing functions. The summer 2009 release is purported to be a major improvement on the current versions of the

libraries. It is to provide a C++ interface to most of the functions in the five major libraries, a better python interface than the current version, better 2D and 3D descriptors and features, major changes to the MLL library to include new procedures, optimization and support for threading, and other things (12).

Part of this project was implemented with the OpenCV libraries. Copying the images from the buffers of the frame grabber, saving the images, retrieving back the images, noise reduction, background subtraction for training, feature extraction, and saving the feature vector sets were done using this versatile library.

2.2.2 Noise Filtering

Digital images like other digital signals are corrupted by noise. The type and source of the noise might vary depending on the environment of the recording, the camera used, and the other tools involved in the capture and storage of the digital images. The noise types may be distributed according to known distributions like the Gaussian or normal. The source of noise might also vary from simple thermal and sound disturbance to electromagnetic interference. For instance, when the first set of images were taken for this project, the electronic valves of ISAC were turned on and the electromagnetic disturbance from the valves created a set of horizontal lines on the images. Since such kinds of noise are difficult to remove, because of their correlation and significant illumination as the objects, the valves were turned off and another set of images were taken. The most typical and dominant noise in the digital images were salt and pepper

noise (sparse light and dark disturbances) and dense Gaussian noise (13). The former one might be due to aging and failure of the CCD (charge coupled devices) in the cameras and dust inside the camera. This type of noise is observed on the images that are taken using ISAC's cameras. Apparently some years ago there was an experiment that used a laser. The researchers repeatedly shined the laser around the right camera of ISAC. As a consequence the right eye is not, now, as effective as the left eye of ISAC. This became evident as the first set of images was taken. The right images were darker than those of the left images and a bit noisier than the left ones. The brightness for the right eye was therefore set to 0.2 and the left eye 0.1 in -0.5 to 0.5 range of brightness. This wasn't the only problem that was encountered due to the discrepancy between the right and left eyes. The noise level of the right eye was significantly higher than that of the left eye. Hence the noise filtering parameter that was good for the left eye performed poorly on the right one. If a good value is set for the right eye, then the left eye was over-filtered which destroyed part of the objects of interest as noise. But an optimal parameter (7x7 window size for the original image and 3x3 window size for the HSV image was used) was found after some experimenting.

Several noise filters were considered initially. After some experimental trials and information about the filters, the median filter was chosen. The types of noise filters can be grouped as linear and non-linear filters. The median filter is an example of a non-linear filter and usually works fine in the presence of the uncorrelated salt and pepper noise that was dominant in this project's images. The median filter scans the image. For each pixel, it gets the list of the neighboring pixels, sorts the list, and replaces the

intensity of the pixel with the median value. This replaces the sparse noise pixels with the median of their neighbors. The overall effect is blurry smoothening effect. If this filter is applied to with a proper window size, it preserves image details like edges, which are particularly important to distinguish the objects from their background. One of the test images is filtered using the median filter and shown below to see the effect. In the example a median filter of window size 9 was used to emphasize the smoothing effect. In the experiments, a window size of 3 was used.

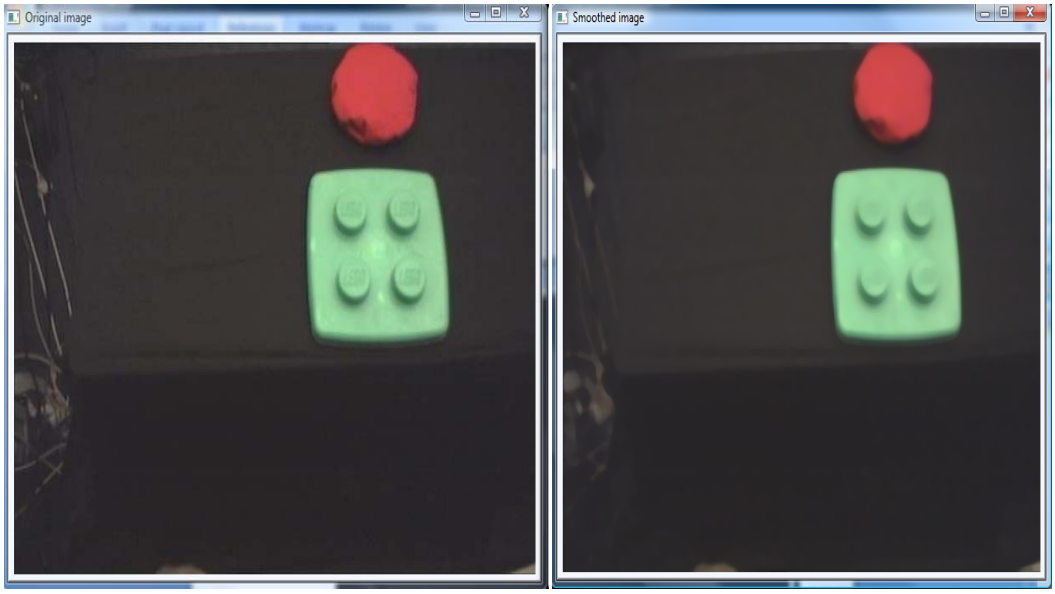


Figure 9: The original image and the smoothed image (not in their original size).

The median filtering is superior to the averaging filter in the presence of isolated outlier points. If the averaging filter were used, the outliers would significantly affect the average and the result will be unwanted. The median filter ignores the outlier points by selecting the median values always and hence removes noise which is uncorrelated to data points or other noise points (14).

2.2.3 Background Subtraction

For the purpose of this project specifically, and in computer and/or robotic vision in general, separating the objects of interest from the unwanted parts of an image is crucial. Since image data is large, effectively doing this job is critical to focus only on the important parts for processing. This project was focused on object recognition using approximate nearest neighborhood. For the recognizer to be effective in the sense of both storage and time, it is imperative that it only focuses on objects to be learned. This was achieved by background accumulation and subtraction on the training images with the result that the number of training feature vectors was tremendously reduced. Background subtraction can be applied whenever there is a steady or close to steady background. For example, we cannot apply background subtraction for each testing image to a robot that is moving around. Because as the robot is moving as the background is changing. However, for this project since the training images were taken in a steady environment, background modeling and subtraction was applied to the training images.

The background subtraction that was used in this thesis involved two steps: learning and modeling the background, followed by its subtraction from the incoming images. This is called the Averaging Background Method (14). It basically learns and stores the average and standard deviation of each pixel as frames of background images are modeled by the system. Based on these absolute frame-to-frame differences, it sets high and low thresholds. Once the background is modeled by sufficient frames, the incoming image is separated in to its color planes and each plane is checked to see if values are in ranges of the high and low thresholds. If they are in range then a single

channel black and white mask image is set and if not it is unset. The three channel results are then logically ORed, because presence of a background in any channel is sufficient. The final result is then inverted to indicate the foreground objects. The mask is passed through some morphological operations to get closer to a regular object shape mask. The original image is then masked to separate out the objects from the background. The following figure illustrates an original image and its background subtracted equivalent.

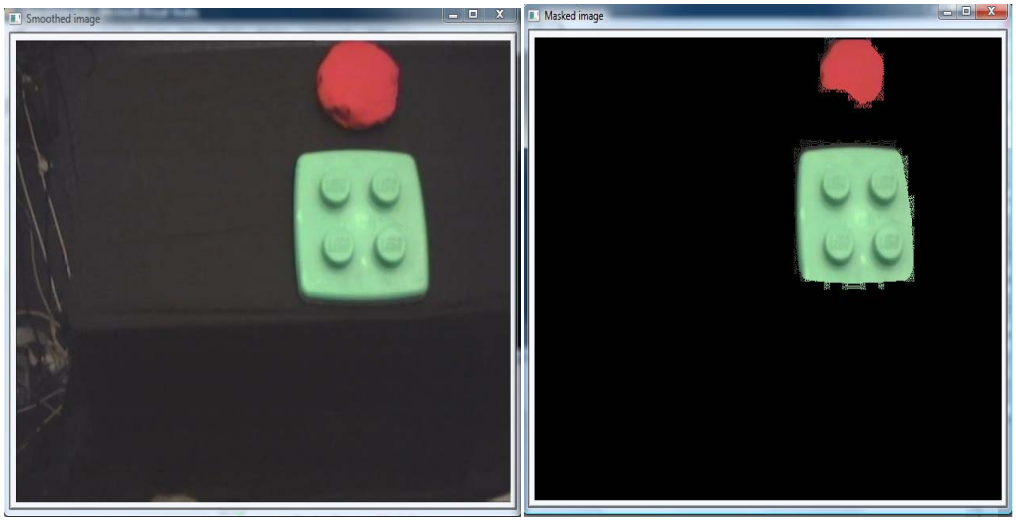


Figure 10: Original smoothed image (Left) and its background subtracted version (right)

From the images above, we can see that strong edges are preserved. This required much experimentation due to the above explained discrepancy between the right and the left cameras. Setting the thresholds and setting parameters of the morphological operations for the mask was very tricky, especially for objects that had similar averages and standard deviations as the background. The red object shown here was among those objects that were difficult to separate from its background.

CHAPTER III

FEATURE EXTRACTION

3.1 Biological Inspiration

Some robotic vision systems are biologically motivated by the human visual system (2) (3). The internal design of today's modern cameras is basically based on the human visual system. The sensory arrays are the counterparts of the rods and cones in the retinas of the human visual system. The electrical stimuli are carried through electrical wires while the chemo-electrical stimuli from the retina sensors are carried by nerve fibers to the brain. This is just a basic comparison. The dependence goes beyond that to the level that much of today's robotics research is based on neuropsychological findings about the workings of the visual system. The following figures show the similarity between the human eye receptor fields and the camera CCD arrays.

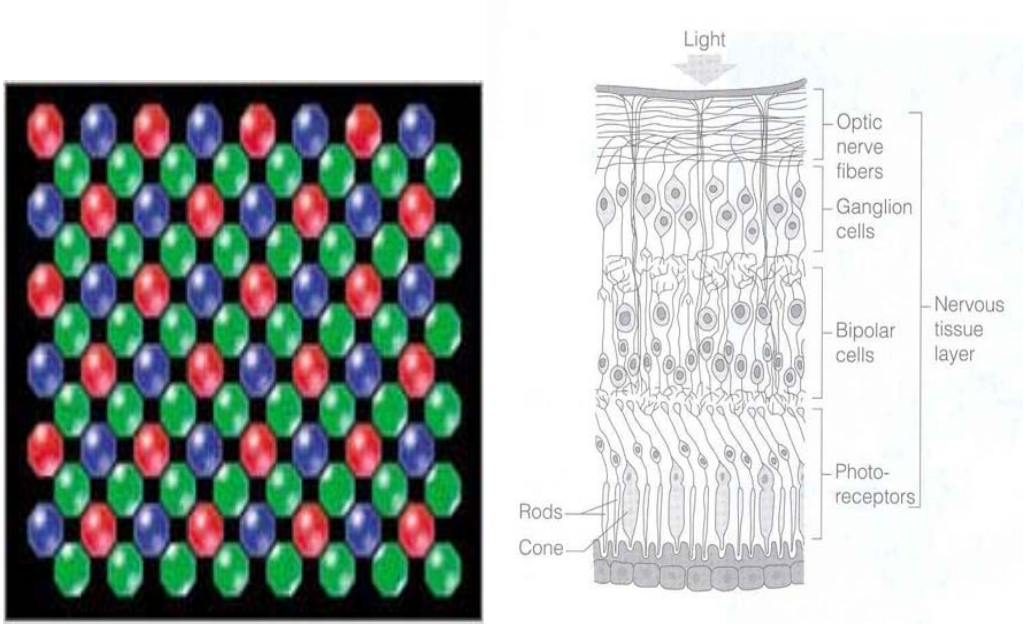


Figure 11: The human receptor fields with in a tissue sample (right) and a typical CCD arrays of a camera are shown here. (2) (15)

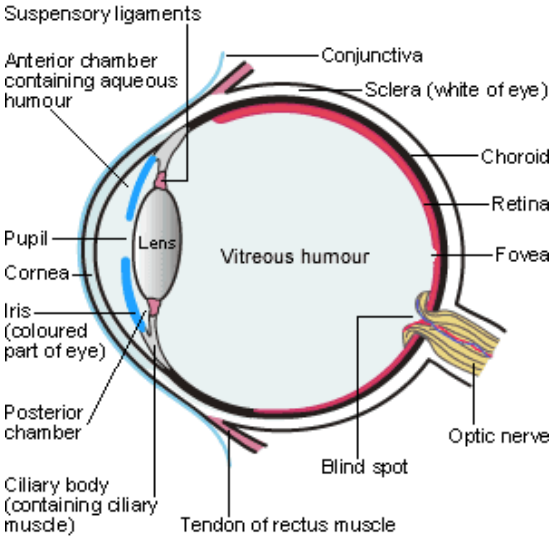


Figure 12: Parts of the human eye (2)

The main difference is the rods and cones of the human eye are not evenly distributed throughout the retina unlike the sensors in camera CCD arrays. Their sizes are not also the same in the different parts of the retina. Although daylight

vision system is cone-mediated, the number of rods (91 million) far exceeds the number of cones (approximately 4.5 million). Therefore, the density of the rods is much higher than that of the cones in most of the retina except at a highly specialized small region of the retina called the fovea. The cones are highly dense and smaller in diameter around there. This high density of cones at the fovea gives this region a high visual acuity. Consequently the human eye can only concentrate acutely on a small region at a time. A 6^0 eccentricity to the line of sight would reduce the visual accuracy by 75% (16). The distribution of cones and rods at the fovea is shown below.

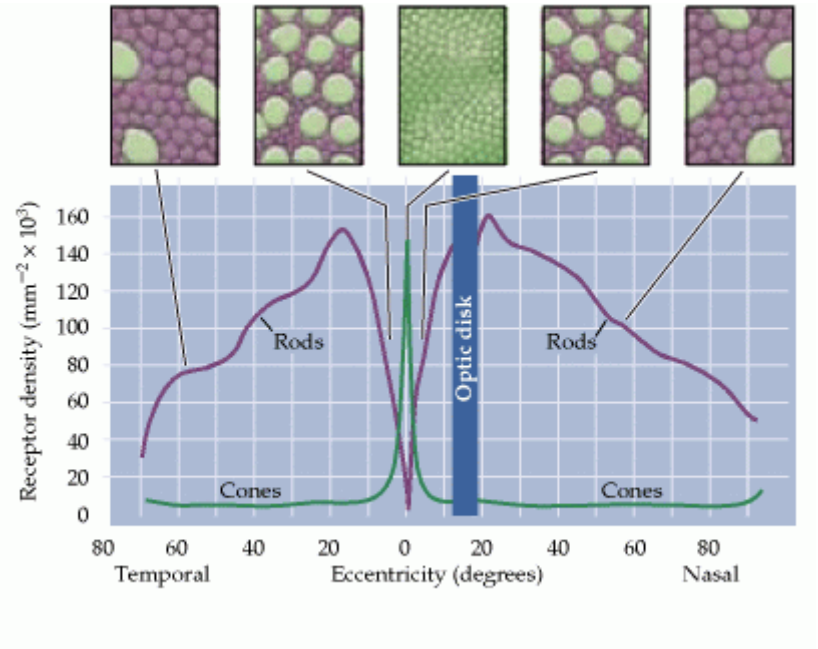


Figure 13: distribution of rods and cones around in the human retina. Boxes at the top illustrate the appearance of cross sections through the outer segments of the photoreceptors at different eccentricities. (16)

The visual stimuli that are formed in these receptors are transported to part of the brain called the visual cortex. The connections, parts involved and the visual cortex are shown in Figure 14 below (17).

Current researchers use fMRI images from human and monkey subjects in experiments to untangle the workings of this complex system. The visual cortex contains at least 5 parts starting from the primary visual cortex (V1) to the Middle Temporal (MT or V5) area. Each area is specialized in to different functions ranging from motion analysis, global localization, feature binding, attentional modulation and others.

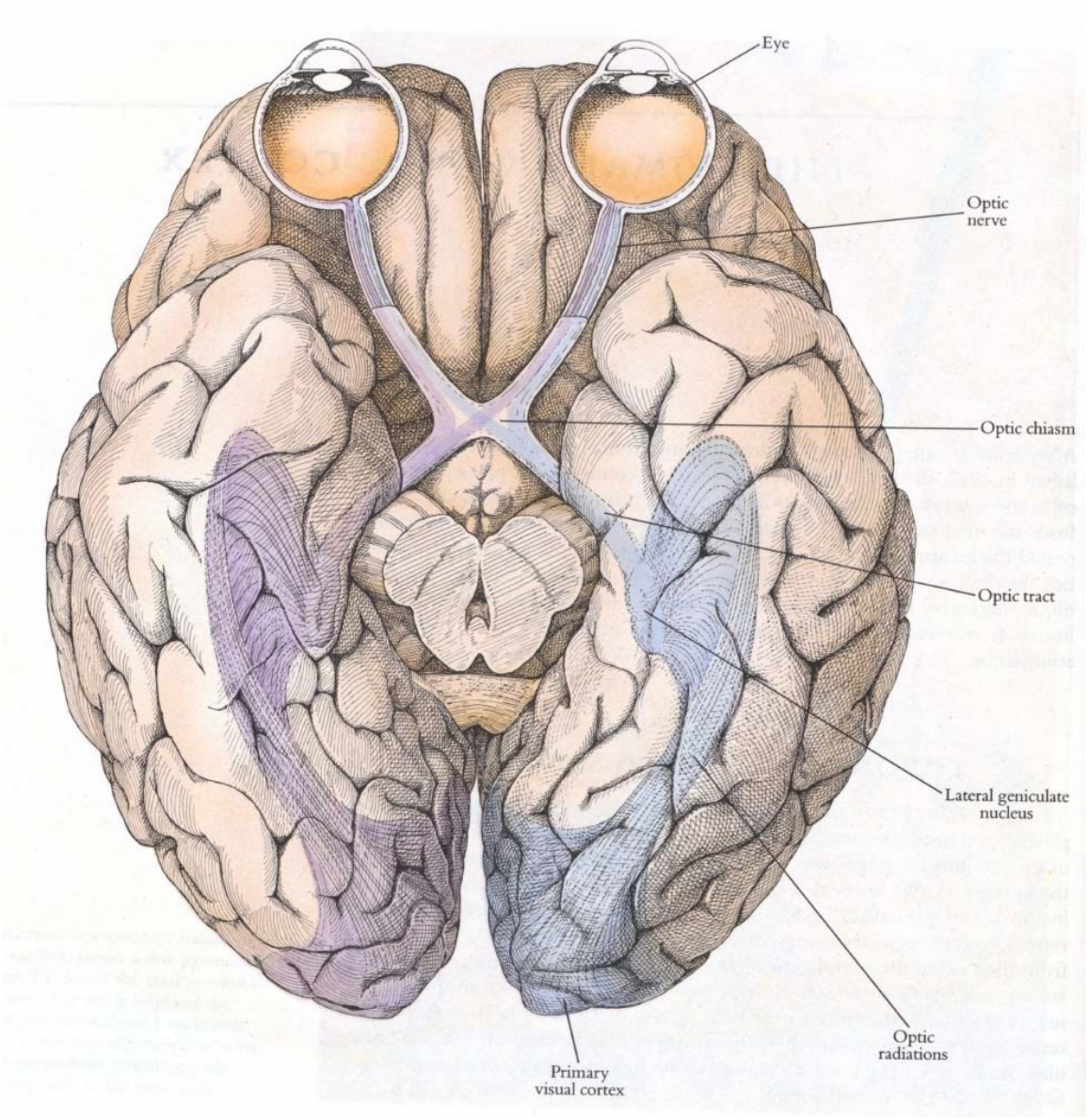


Figure 14: The Visual pathway from eyes to primary visual cortex, of a human brain. (17)

The lateral geniculate nucleus (LGN) receives the signals from the optic nerves and most of the color coding is done here. Here, there are many color sensitive cells. The partially processed visual information is transmitted to the primary visual cortex, which is also known as V1, which separates the information in to different parts and passes it to the specialized visual areas of extrastriate cortex region for further processing. There are particular classes of neurons in V1 that are unresponsive to edges, bars and lines but are responsive to texture. The next part of the visual cortex (V2) is divided in to two major parts: the dorsal and the ventral pathways (18). The ventral pathways (the "what" pathways) are responsible for object identification using features like color and shape, while the dorsal pathways (the "where" pathways) are responsible for processing of position and motion or in general object location (2). This is the inspiration that motivates this work for object detection and localization. The selection of the features is based on this biological motivation.

3.2 Background of the features selected

There are many features to be considered for segmentation of colored images and hence object detection. Color, textures, and shapes are among the most common ones. Shapes are relatively complex features that are not most frequently used for segmentation purposes alone. Shapes can be represented by using contours. An object's shape can be modeled by contours that are obtained from the object in the training time and some distance measure can be used as explained in (19) to cluster pixels which do or do not

belong to a particular shape model. Other methods use descriptors to store information that is obtained from the contours. There are a number of representations that are used for this purpose. Chain codes, polygonal approximations, signatures, boundary segments, skeletons, and others are among such representations (20). Shape features can be used together with color and texture features. Texture features and color features alone are a robust feature combination. Each of them alone may not be efficient and sometimes actually unhelpful.

Textures are useful to identify the smoothness, coarseness, and regularity of a region. Different texture descriptors are used for such measures. Some are maximum probability, element difference moment, inverse element difference moment, uniformity, and entropy (20).

Color features are often used for image segmentation for robotic applications. Human color perception is not understood fully in the psychology and neurobiology world. However there has been much research on the subject. In digital image processing colors are represented by models called color spaces. There are different color space representations of the same colors. The RGB space is the most widely used space for the representation of the information contained in digital images and storage. Each pixel in a digital image has three separate values. The collection of these values defines color space planes.

The R, G, and B are obtained by a color camera, with the general model (21):

$$C = \int_{\lambda} p(\lambda) f_c(\lambda) d\lambda$$

Equation 1: Camera Color Model

Where: $C \in (R, G, B)$ $giving\ the\ c^{th}\ sensor\ response,$

$$where\ p(\lambda) is\ the\ radiance\ spectrum\ and$$

$$f_c(\lambda)\ are\ the\ three\ color\ transmission\ functions$$

Other important color spaces include CMYK, l*u*v*, the l*a*b*, HSV, HSL or HSI, YUV, and others. RGB is used mainly for representation and storage as it is very sensitive to environment variations like lighting conditions. This is because it is a direct function of the sensor arrays of the camera. For segmentation and object recognition purposes, mostly color spaces that separate luminance from color like the HSV, HSI, and l*u*v* are used (22) (23) (24) (25) (26) (27).

3.3 Rationale For the use of High Dimensional Features

Segmentation can be performed using methods ranging from simple thresholding to using feature vectors of high dimensionality. Simple thresholding suffers from the specificity of the threshold. Choosing an optimal threshold is not an easy task and sometimes it may even be impossible. For instance, if there is a lot of dynamicity like lighting variations in the environment, the color values even for a particular object will vary in range. There might even be a range overlap between two colors. So, in such situations not only the simple thresholding, but also adaptive thresholding techniques will suffer a lot.

Generalized feature vectors are often used for object recognition. The dimensionality of a feature vector can have an effect on the accuracy and the speed of the recognizer, balancing the two and achieving the optimal dimensionality is of a great importance. The time and space complexity of an algorithm are fundamental measures of the algorithm. However, for non-perfect (in result) algorithms like those for machine learning accuracy is also important.

In general having a higher dimensional feature gives better accuracy overall and, consequently, a higher capacity to learn (3). For instance, in this project, there is a red bean bag object. The totality of colors deemed to be red will usually range over some values and might therefore require a number of elements in the feature vector. The higher the dimensionality of the feature vector, the higher the resolution of the description of red. If it is used correctly that greater resolution in representation can lead to increased accuracy in the testing phase.

However, using a high dimensional feature vector has a time and space penalty, which is similar to '*the curse of dimensionality*' problem where irrelevant attributes of a feature are included in a vector while only a small number of dimensional attributes is sufficient for classification (28). But, the bins in the feature vector used in this project are mostly unoccupied (ZERO), because the feature vectors represent very large sets of possible colors and a texture measure but are gathered from a small region of the image. Moreover, not all colors will be present in a particular scene. For example, this vectors used by this project each had a total of 10,000 color bins and a texture measure concatenated to form 10,001 dimensional (length) feature vector. But in a particular scene

in a particular region, there will only be hundreds (at most) distinct colors present. Therefore, the feature vector will be mostly zeros. This gives the advantage of using a sparse representation for the feature vectors. Only the non-zeros values together with their indices are stored and processed. This tremendously decreased the time and the space complexity of the nearest neighborhood algorithm that was applied to them. The other 'curse' of dimensionality is the need for larger training sets as the dimensionality increases. The rule of thumb suggested by (3) is to use five times the size of the feature vector, which generally implies that the size of the database increases linearly with the size of the feature vector. However, as images are naturally full of data, it is easy to take as many training images as required and get the required minimum database size. However, the size of the database should not be increased beyond the point where the accuracy curve starts to flatten out. Increasing the database number beyond adds computational cost without any significant gain in accuracy.

3.4 The perceptual features used

By taking biological evidence, time and space complexity, accuracy and simplicity in to account, the feature vectors used in this project were composed of color bins and a texture measure. The two features are described in more detail below.

3.4.1 The color probability density function (pdf)

To use color as features that represent a particular region in general and an object in particular, one of the color spaces must be selected. Due to its relative insensitivity to lightning conditions, rotation and translation, and other factors the (Hue, Saturation, Value) or HSV color space is often used for image segmentation. It is more robust than the RGB representation and is computationally inexpensive compared to other robust color representations (29). The HSV color space as formally defined by Alvy Ray Smith in 1978 is extensively used in computer graphics and other applications (29). Many interactive graphics applications use an HSV color wheel that permits a user to select from a continuous distribution of colors. Among such applications are Pixel Image Editor, Pixia, Bryce, The GIMP, Paint, Mac OS X, and Photoshop. Other well known applications like CSS3 specification, Inkscape, Macromedia studio, MS Windows, Paint Shop use HSL, a relative of the HSV space.

There are many definitions for HSV in different literature. However, this project uses the most standard definition used by the above mentioned well known applications as well as the OpenCV library (29) (30). The definition that was implemented for this project was compared to the built in function in OpenCV and found to be similar. The definition is as follows:

$$\max = \max(r, g, b), where\ r, g\ and\ b\ are\ normalized\ or\ are\ in\ range\ [0,1]$$

$$\min = \min(r, g, b)$$

$$h = \begin{cases} 0 & ,if\ \max = min \\ \left(60^0 \times \dfrac{g-b}{\max - min} + 360^0\right) mod\ 360^0 & ,if\ \max = r \\ \left(60^0 \times \dfrac{b-r}{\max - min} + 120^0\right) & ,if\ \max = g \\ \left(60^0 \times \dfrac{r-g}{\max - min} + 240^0\right) & ,if\ \max = b \end{cases}$$

$$s = \begin{cases} 0 & ,if\ \max = 0 \\ \dfrac{\max - min}{max} = 1 - \dfrac{min}{max} & ,otherwise \end{cases}$$

$$v = max$$

Equation 2: Hue, Saturation and Value computation

The HSV space has a direct graphical interpretation of colors. The hue is a measure of the visible difference between colors. The saturation is the measure of how intense a particular color is and the value is the brightness or luminance of the color. The following cone is a mathematically accurate model of the HSV space.

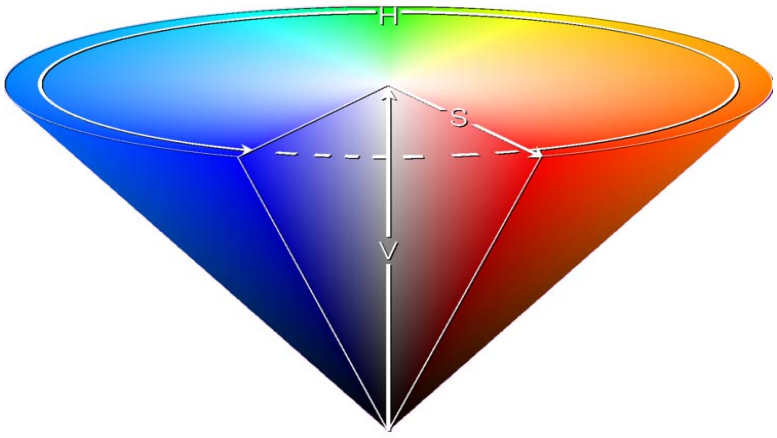


Figure 15: A Conical Model of the HSV color space. The hue is the angle that varies from 0^0 to 360^0. The saturation and the value both range from 0 to 1. (29)

3.4.2 The texture component

To differentiate between two regions that are similar in color features, a single texture value of the region was incorporated. A spatial Lapalacian operator was applied to the image region. The Laplacian is a 2D isotropic measure of the second spatial derivative of an image region (3). The Laplacian of image intensity with spatial coordinate (x, y): $I(x, y)$ is given by:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Equation 3: Laplacian

To compute the Laplacian for the discrete valued digital image, the following kernel is used.

$K_{LAP}(x, y)$ = 1/8 *

-1	-1	-1
-1	8	-1
-1	-1	-1

Equation 4: Laplacian kernel

First, the image was convolved with this Laplacian kernel. The result of this was normalized to 1 and its magnitude was further convolved with an averaging kernel. The averaging kernel (K_{AVE}) was a 15 x 15 window of ones in each cell and a dividing factor of 225 (= 15 x 15). The overall effect of the kernel is to normalize the images region's Laplacian values and then average out to get the texture measure.

The calculation of the final texture measure is then summarized as follows:

$$Texture = Conv(K_{AVE}, Abs\left(\frac{Conv(K_{LAP}, I)}{255}\right))$$

Equation 5: Texture measure

3.5 Extraction of the features from the images

This section explains how the features were extracted from the images. The input image was passed through a color space conversion routine that converted the native RGB vectors into HSV space as described in the above section. The image in HSV space is then is scanned by a moving window to extract the features for each analysis region. The size of the region of interest (ROI), which equals the window size, affects the segmentation algorithm. Too large a size will cause the number of feature vectors extracted from an image to be insufficient to represent the objects and the background. Too small a size will extract many feature vectors, which make the resolution subtle, but may force the classifier to be too specific and to lose generality as well as increase the processing time... The optimal value for the images that we are using was found to be 15 x 15 in [Tugcu 2007] (3). Therefore each ROI is obtained by moving the moving window of size 15 x 15 (225 pixels) around the entire image with a horizontal and vertical displacement of 10 pixels. The regions overlap by 5 pixels in each direction to enforce continuity on the feature vector sample. An HSV ROI taken out of an image while processing is shown below:

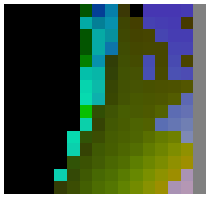


Figure 16: A 15 x 15 ROI of an image from which a single feature vector is calculated (Note: the figure is magnified from its original small size to see the details).

For each such ROI, a feature vector of size 10,001 is extracted. The first 10,000 values are obtained from the normalized color histogram (which is a probability density function – pdf) of the ROI while the last value is the texture measure of the region. The pdf is computed from a quantized histogram that partitions the HSV space in to a 10, 000 regions by dividing the H dimension into 100 evenly distributed bins and the S and the V dimensions into 10 bins. The H dimension is first normalized by dividing the positive angle in degrees by 360. Hence the bin step is effectively 0.01. For the S and V dimensions the bin step is 0.1 as their range was originally from 0 to 1. Each element of the first 10,000 in the vector for a region therefore contains the number of pixels in the region that had a color in the range represented by the corresponding bin of the histogram.

For any given region, the number of distinct colors that can is obtained is never greater than the total number of pixels contained in the region — 225. That is small compared to the size of the vector that is used to represent the region (10,000). Therefore, most of the feature vector elements are zero. In fact each feature vector will always have at least 9,775 zeros in it. Here is where an efficient way to store and process this feature is imperative. There is neuropsychological evidence that shows that primates use sparse coding (31) for feature representation in the primary visual cortex. A computer analog of

this is an efficient coding that enables fast processing and minimal storage space for a feature vector. A sparse vector is stored using two separate vectors, one for the non-zero values and one for their indices into the feature vector. Any increase in the dimensionality of the feature vector affects neither the computational time nor the storage requirements since the number of possible colors with non-zero pdf values is determined by the number of pixels in the region.

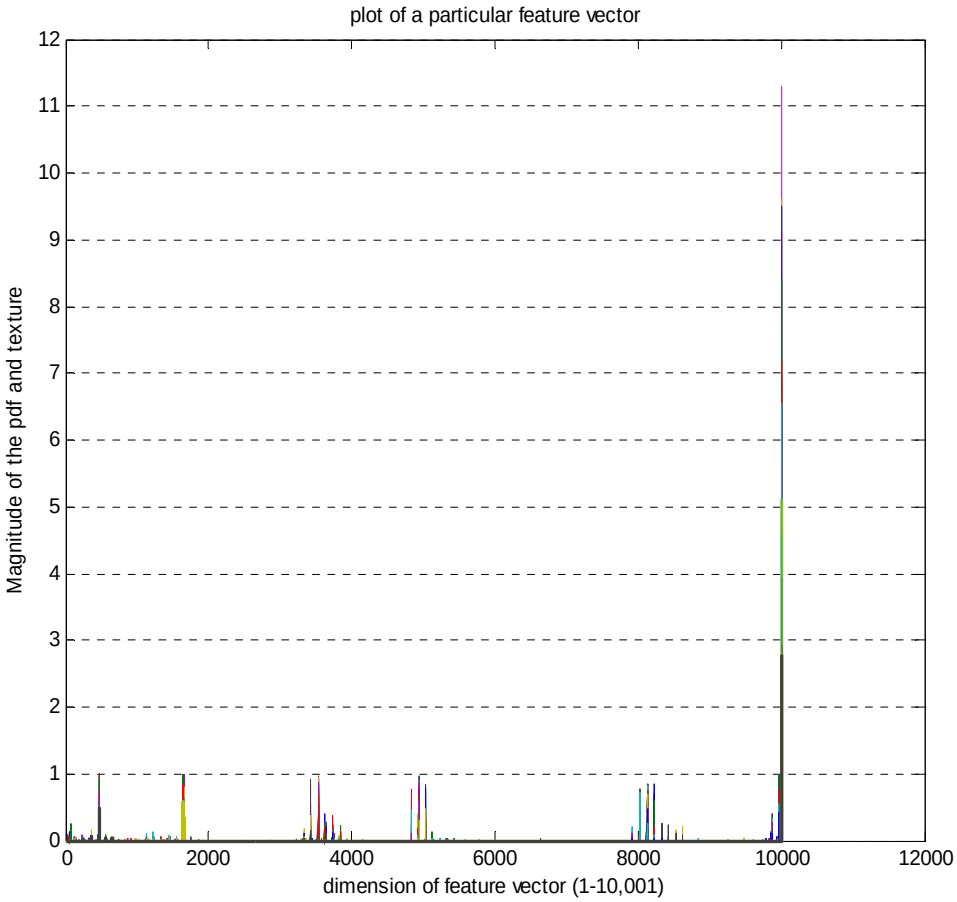


Figure 17: A typical set of training feature vectors. The spikes at 10,001 are the texture measures and the rest are the pdf of the histograms.

The above figure shows a set of training features that contain 413 feature vectors. The feature vectors are obtained from 6 images that contain each of the six experimental

objects together with the background. We can see that there are 6 clusters other than the texture spike. These correspond to the six objects. From this, we can see that the feature vector is easily separable and mostly concentrated around the objects' maxima.

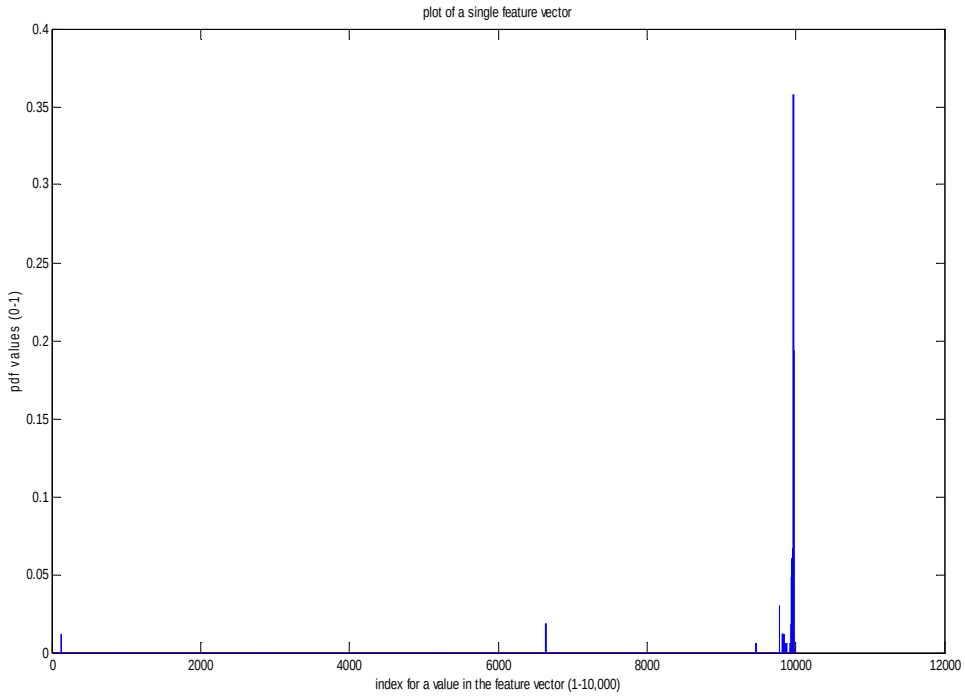


Figure 18: Plot of a single feature vector that shows how sparse the vector is. It has dominant non-zero values near the maxima of the pdf of the objects color.

CHAPTER IV

PERCEPTUAL LEARNING AND SEGMENTATION

4.1 Perceptual Learning

The main problem in this thesis is a typical classic problem that can be addressed by a machine learning algorithm. Among several machine learning algorithms, simplicity and being readily suited for the high dimensionality nature of the problem, the nearest neighborhood algorithm is chosen. In this chapter, first the generic pure k-nearest neighborhood search algorithm is explained and then the distance measure that is used in the approximate search method is discussed. The random tree implementation of the approximate nearest neighborhood search algorithm that is used in the thesis is then presented. Finally, the testing of the algorithm to segment images is explained.

4.1.1 The Pure Nearest Neighbor Algorithm

There is a class of machine learning algorithm called instance learning (28). Instance learning constructs the approximation target function at the time of testing unlike the other classes of machine learning algorithms. The usual learning algorithms construct the description of the target function when the training samples are presented

while instance based learning methods store the training samples themselves. Until a new instance of testing samples is presented for classification, generalization beyond the examples is postponed. These types of algorithms are usually called ‘lazy’ learning methods because processing doesn’t start unless a test instance vector is presented to be classified. This has an advantage of giving the approximation function the ability to encompass the new instance instead of pre-formulating the function from examples and classifying the new instance using the old function.

The nearest neighbor algorithm is one of such instance-based learning methods. The learning phase is storing the example data in memory and when a new query instance point is encountered, a set of similar related instances is retrieved and used to give the query instance a label (classify). In the testing phase, a local distinct approximation function is formed for each query instance to classify. This has an advantage when the target function is complex if it is to be generalized by the whole instance space and hence results in a less complex local approximation.

The big con of this learning method is that the cost of classification both in time and space can be very high as the number of data becomes very large. This is because all the computation is at classification time as opposed to at training time. This means for each test vector, the set of similar training vectors are processed to give the target function. Therefore, there should be an efficient indexing technique for indexing the training examples should be devised. The approximate nearest neighborhood algorithm that is explained below is one of such techniques and is the one that is used in this project.

The k-nearest neighbor learning algorithm assumes all instances correspond to points in the n-dimensional space$\Re^n$. It can be used for discrete set of data and real valued set of data. The mapping function is described by:

$$f: \Re^n \rightarrow V$$

Equation 6: approximation function

Where $V\ is\ a\ finite\ set\ of\ values\ \{v_1, \ldots, v_s\}$

Let x_q be the query instance point. The algorithm a value $\hat{f}(x_q)$ that is determined by the values of the k-nearest neighbors of x_q. For K = 1, the value is $f(x_i)$, where x_i is the closest point to x_q. If K is different from 1, the most common value among the K nearest neighbors is assigned to the query instance. This is the most basic definition of the algorithm. The algorithm can be augmented by giving weights to points which is proportional to the distance squared of a training point from the query point.

The formal mathematical definition of the algorithm with weights for both discrete and real-valued data sets is given below (28).

$Training\ Algorithm$:

- $For\ each\ training\ example\ \langle x, f(x)\rangle,$

  $add\ the\ example\ the\ list\ of\ the\ training\ examples$

$Classification\ Algorithm$:

- $Given\ a\ query\ instance\ x_q\ to\ be\ classified,$
  - $Let\ x_1, \ldots, x_k\ denote\ the\ k\ instances$

    $from\ the\ training\ examples\ that\ are\ nearest\ to\ x_q$
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i)) \ for\ discrete\ data\ set$$

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \ for\ real-valued\ data\ set$$

$$where, w_i \triangleq \frac{1}{d(x_q, x_i)^2}$$

$$and, \delta(a, b) \triangleq \begin{cases} 1, & if\ a = b \\ 0, & elsewhere \end{cases}$$

$$the\ distance\ is\ the\ Eucledian\ distance,$$

$$if\ a\ point\ x\ is\ described\ by\ a\ feature\ vector$$

$$\langle a_1(x) a_2(x), \ldots, a_n(x) \rangle$$

Equation 7: Algorithm of pure nearest neighbor search

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Equation 8: General Euclidean distance

The above defined algorithm is a very robust inductive inference algorithm in the presence of noisy training data. Its effectiveness is linked with the number of training data points provided. The higher the number of training vectors the higher the effectiveness or the accuracy. This algorithm delays all processing until a new query point is presented for classification as discussed above and hence an efficient memory indexing scheme is apparently required to make the classification faster. This is gives rise to a need to implement the approximate nearest neighbor algorithm discussed below. Various methods have been proposed as shown in section 4.1.3. The nearest neighbor

algorithm is used for image segmentation and object recognition, and other tasks in various researches (32) (33) (34).

The pure (aka, naïve) nearest neighbor search calculates the distance from the query point to every point to the database to get the k-best nearest neighbors from the training samples (35). Let the training sample set be S, the cardinality (the number of training vectors) of S be N, and the dimensionality S or the dimensionality of each vector be d: the running time of the comparison is approximately $O(Nd)$. This indicates that when the dimensionality is comparable to the number of elements, the running time will be approximately$O(N^2)$. And this implies that as the number of training samples increases, the running time increases with the square of the numbers and this is the testing time or the response time for the query. This creates a long latency and possibly an unresponsive system as N becomes very large. This is undesirable property of the pure nearest neighbor. Therefore, considering efficient indexing algorithms is inevitable. Many such algorithms are developed. Most of such algorithms use tree data structures as the problem this is a clustering problem and the data sets become more and more alike as they are sub-divided in to sub groups using the distance measure as a similarity measure. In the sections below, we will see, first, the similarity measure and different types of indexing algorithms in general and the random tree implementation that is used in this work in particular.

4.1.2 The Distance (Similarity) Measure

The search algorithm uses similarity as criterion to classify a query point based on the training samples. This similarity is one of the key issues and perhaps the most important for the effectiveness of the algorithm. Specifically, in the nearest neighbor search, the similarity measure is some sort of distance between the query point and a point in the training samples in the feature space as shown in the algorithm's presentation in the above section. Researchers have been investigating different distance for this purpose. Among such distances are the Euclidean distance, the Mahalanobis distance (36), and the others (3). However, most fast nearest neighbor algorithms are specifically using the Euclidean distance and for general metric spaces, progress has been slow (37). Due to its simplicity, the Euclidean distance makes the search faster as the similarity is computed faster although it is scale-variant, which means that it is dependent on the scale of measurements.

The Euclidean distance is among the distance metrics generically known as the Minkowski metric although the Minkowski metric generally includes a time dimension besides the space dimension in which the Euclidean space is based on. The general spatial Minkowski metric in a K-dimensional space is given by:

$$d_{ij} = \left(\sum_{k=1}^{K} \left| x_{ik} - x_{jk} \right|^{p} \right)^{1/p}$$

Equation 9: General Minkowski Metric Formula

These distances are of the form l_p norm equations. It is of particular importance when $p = 2\ (Euclidean\ metric)\ and\ p = 1\ (city - block\ metric)$ as described in (38)

(39) (40) (41). For unitary or holistic stimuli, such as the color pdf, the closest approximation to an invariant relation between data and distances has been achieved by Euclidean distance and though for separable stimuli like the texture a city-block could be used (2), as the texture measure is a single value amongst the 10,001 values using Euclidean distance for all doesn't affect the system performance significantly. Hence, the Euclidean distance is used in this project as a similarity measure. Therefore, the distance between two feature vectors $x_i \, and \, x_j$ as shown below:

$$d_{ij} = \sqrt{\sum_{k=1}^{10001} (x_{ik} - x_{jk})^2}$$

Equation 10: Euclidean distance for the approximate method

4.1.3 The Approximate Nearest Neighbor Algorithm (The Random Tree)

As discussed above, the run time of pure nearest neighbor search algorithm becomes intractable as the number of elements in the training set becomes large. Many researchers have investigated various data structures to find an optimal indexing algorithm that can reduce the runtime of the search from quadratic to at least linear or at best logarithmic (42) (43) (44) (45) (46) (47) (48) (49). Among the representations that are used by the above researchers are Kd-tree, R-Tree, Quad Tree, Metric Tree, Cover Tree, Spill Trees, Approximate Nearest Neighbor Graph and others. Although there are many solutions that can give a running time of $O(\log(n))$ as depicted by (50) (51) (52), they are not suited for very high-dimensional feature space. The ANN library, for

example, is good up to dimension of 20 and its results are not good beyond this dimension.

Therefore, an efficient implementation for very high-dimensional feature spaces is demanded as this project is based on the advantage of the very-high dimensional feature space. [Tugcu and Wang, et al (3) (2)] proposed a tree-structured vector quantization method that is used to generate a 3-way random nearest neighbor search tree. It is basically an implementation of an M-Way search tree with M = 3. However, this implementation is purely random and there is no ordering in generation of the child nodes from the parent node. The approximate search method that is implemented in this project is based on this proposed method. First the tree data structure and then the tree creation and search algorithms that are used in this thesis are presented in the following figures.

The Random Search Tree data structure:

Table 1: This the basic data structure for the search tree.

$Structure\ Name: Node$
$Data\ members:$
$Data$
$dist$
$center$
$nodeID$
$isPure$
$isLeaf$
$label$
$dataLength$
$depth$
$distStat$
$Right$
$Middle$
$Left$
$Parent$

Each node keeps a set of indices to the training feature vector data, a reference to a reference to its parent, a reference to its children, the center index, and some other useful data and statistics about the data.

The Random Tree Building Algorithm is given below:

```
function Tree(pointsList, center, parent, nodeID, dist)
{
    global totalNode;
    if(pointsList is empty)
        return null;
    else
    {
        distStat = calcStat(dist);
        node = Node(pointsList, center, parent, nodeID, dist, distStat);
        totalNode + +;
        if(node is not root node)
        {
            node.isPure = checkPurity(node);
            node.isLeaf(node);
            node.depth = node.Parent.depth + 1;
        }
        else
            node.depth = 0;
        if(node.isLeaf)
            return node;
        else
```

$$
\begin{aligned}
&\{ \\
&\quad [r, m, l] = \ getRandomCenters(node); \\
&\quad [lR, lM, lL, dR, dM, dL] = \ splitList(pointsList, r, m, l, node); \\
&\quad node.Right = Tree(lR, r, node, totalNode, dR); \\
&\quad node.Middle = Tree(lM, m, node, totalNode, dM); \\
&\quad node.Left = Tree(lL, l, node, totalNode, dL); \\
&\} \\
&\} \\
&\}
\end{aligned}
$$

Equation 11: Training algorithm for the random search tree

The tree creation starts with an original list of all indices to the feature vectors presented to the system. The parent node then constitutes this original list. Three indices are then randomly chosen (they must refer to three different feature vectors) and are taken as the centers indices for the three children nodes of the parent node. The original list is then divided in to three based on the similarity measure, Euclidean distance. Vectors that are similar to the right center are grouped to the right node indices and so on. The sub lists are again sub processed by considering as if they are the original lists of their respective nodes. This process will continue until two conditions are met: either the parent node is a pure node (where all training examples that belong to the list of this node have same training labels) or the number of the feature vectors that belong to the node is less than a certain minimum. This minimum and the number of total (original) training vectors are the parameters that are tuned to get optimal tree. Therefore, those are the two

parameters of the random tree that are used in this thesis to get optimal values for the parameters. Therefore, a leaf node is either a pure node or a node with data length less than the minimum.

Sorting the indices based on the training labels of the training example vectors had been tried in this project to get an optimal tree that may have much pure leaf nodes than tree that is generated by randomly selecting the centers. Quick sort have been tried and its space complexity as it uses recursion and much stack space is intractable as the number of points is becoming very large. Next, one of the slow ($O(n^2)$) stable sort algorithms called gnome sort have been tried. However, its run time for the original list was almost comparable to the random tree creation time. Therefore, the idea of sorting was not used and hence the tree used is pure random tree.

The Random Tree Search Algorithm is given below:

```
function search(node, queryPoint)
{
    if(node is null)
        return null;
    if(node.isLeaf)
        if(node.isPure)
            label = node.label;
            return label;
        else
        {
```

$label = knnPureClassify(queryPoint, trainData, trainLbl);$

$\boldsymbol{return}\ label;$

$\}$

$\boldsymbol{else}$

$\{$

$nR = node.Right;$

$nM = node.Middle;$

$nL = node.Left;$

$cR = trainData[nR.center];$

$cM = trainData[nM.center];$

$cL = trainData[nL.center];$

$dist = calculateDist(R, cR, cM, cL);$

$\boldsymbol{if}(\min(dist)\ is\ the\ right\ dist)$

$label = search(nR, queryPoint);$

$\boldsymbol{else\ if}(\min(dist)\ is\ the\ middle\ dist)$

$label = search(nM, queryPoint);$

$\boldsymbol{else}$

$label = search(nL, queryPoint);$

$\}$

$\}$

Equation 12: Search algorithm for the random search tree

Once the tree has been created, given a query point, the tree can be searched using the root node only. The search algorithm is very straightforward and it uses the distance heuristic and hence optimal than a brute force depth and breadth searches at least. It starts from the root node and compares the query point with the three centers of the children nodes. The search will then continue with the node whose center is more similar to the query point than that of the others until it reaches a leaf node. If a leaf node is reached, the node is checked for purity. If the leaf node is a pure node then, the label of the training samples in that node is returned. If the node is an impure node, a pure nearest neighbor search is applied on the sub list of the leaf node. This will increase the accuracy of the search without significantly affecting the search time. The reasons for this are: if, the tree is an optimal tree, the proportion of impure leaf nodes are not significant compared to pure leaf nodes and the sub list that is used for classification is very small (less than or equal to the minimum number of vectors of leaf nodes) compared to the original list. This has been proved from the results obtained. Since, the query point is only compared to the three centers of the children nodes, the search time is approximately $O(\log(n))$. This is also proved in the results obtained. However, there is an additional space requirement due to the tree structure. If the original training data was an $N\ X\ d$, where N is the number of points and d is the dimensionality, an extra space of $O(N)$ is required for storing the indices of the training feature vectors in the leaf nodes. The sum of the number of the entire leaf node vectors is equal to the length of the original list, N.

4.2 Segmentation

Like the training images the test images are passed through the pre-processing (except background subtraction) and feature extraction steps described above. This results in set of query points to be classified and labeled. Once the search tree is formed using the training feature vectors and labels that are obtained from the training images, each query point is labeled using the search algorithm described above. Each labeled query point in the feature space represents a 15 x 15 image ROI (patch) in the image space. As a consequence, the label obtained for the query point is given to the 225 pixels in the image space and that represents a percept. One or more of such image patches constitute the test objects in the scene. The image that is obtained by assigning the labels to the classified pixels and zero to the unclassified non-trained object pixels is the segmented image. The segmented image is then analyzed using morphological operations to get correct blobs for the recognized objects. The blobs are then passed through region property operator to get the region properties like centroid, area, perimeter, bounding box, etc. Since there are misclassifications by the system, getting the blobs would produce false object blobs that belong to a group of misclassified image region. This is corrected by re-assigning labels based on a simple rule. Pixels that are neighbors (in image space) to pixels that represent a percept are re-labeled to the label a new label that represents the percept. A percept is then the label that has the maximum number of pixels in the region. The following figures show two images that demonstrate a segmented image and an image that is re-labeled to get the blobs of the objects in the scene.

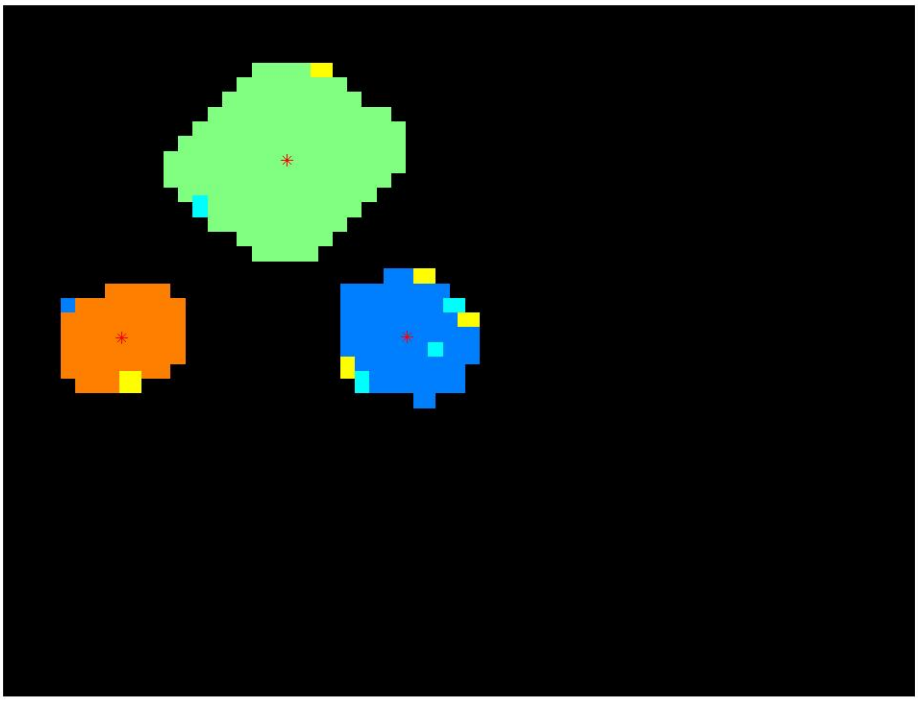


Figure 19: A segmented image that is before the re-labeling with the recognized objects and their corresponding centroids. (Centroids are superimposed for completeness. However, they are calculated after re-labeling.)

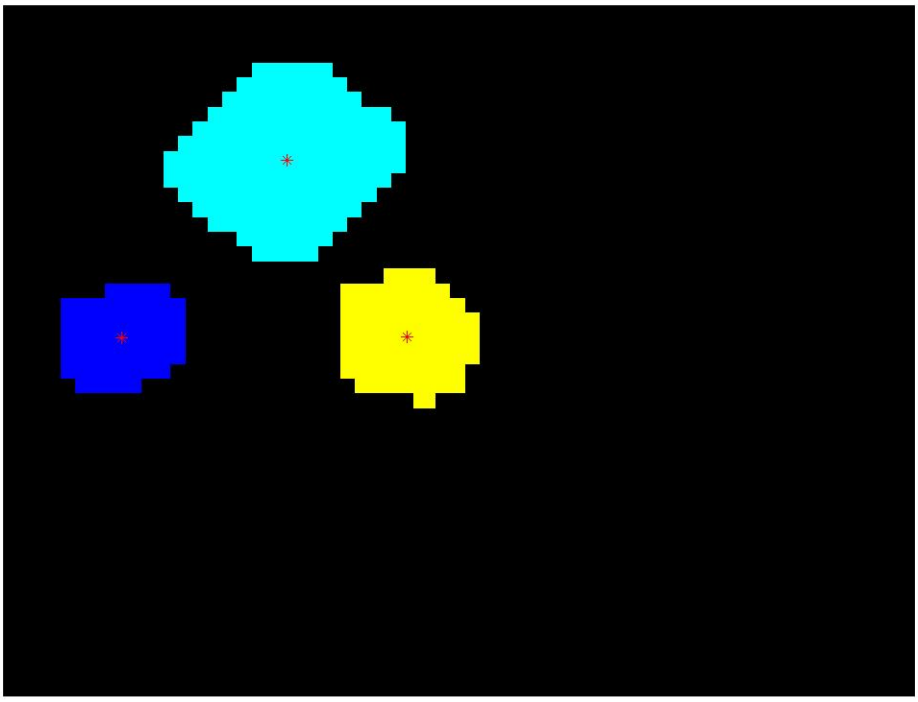


Figure 20: A segmented image that is after the re-labeling with the recognized objects and their corresponding centroids.

4.3 Performance Measures

There are standard performance measures that are used to compare machine learning algorithms. Among such measures are training time, testing time, accuracy, and ROC curve. These measures are used to compare the performance of the random tree implementation with that of the pure (naïve) nearest neighbor algorithm. The first three measures are straightforward and most common measures for most prediction related algorithms. The fourth one is specific to machine learning algorithms and communication

systems and is a standard measure of machine learning methods. ROC (which stands for Receiver Operating Characteristic) is a plot of true positive prediction rate versus false prediction rate. In an experiment with negative outcomes (absence of a particular object) and positive outcomes (presence of a particular object), the four possible outcomes are shown in the figure below.

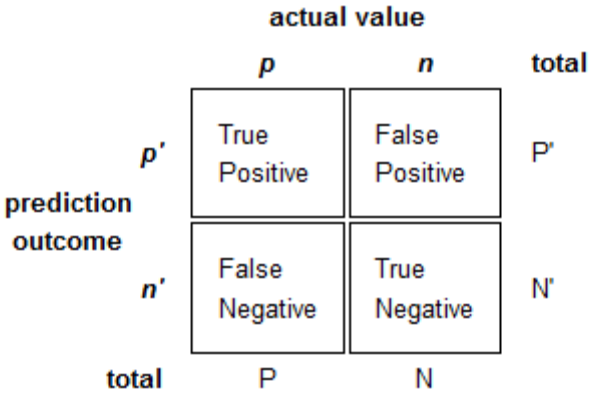


Figure 21: Contingency table or Confusion matrix of the four possible outcomes (53).

$$True\ Positive\ (TPR) = \frac{True\ Positive\ (TP)}{Total\ Positive\ (P)} = \frac{TP}{(TP + FN(False\ Negative))}$$

$$False\ Positive\ Rate = \frac{False\ Positive\ (FP)}{Total\ Negative\ (N)} = \frac{FP}{(FP + TN(True\ Negative))}$$

$$Accuracy\ (ACC) = \frac{TP + TN}{P + N}$$

Equation 13: ROC calculation equations

Therefore, the above three performance metrics together with training and testing times are the performance measures of this project.

CHAPTER V

STEREOPSIS

Stereo-vision is inevitable to get an accurate estimation of depth to a point in the world frame of reference out of the camera coordinates. This is the crucial part of the vision system after segmentation. The depth of a point can be computed from the disparity between points in the right and left images. Disparity is the absolute positional difference between the two projections of a given 3D point in the left and right image planes of the cameras. Intuitively, disparity is inversely proportional to depth. However, there are methods to accurately compute depth from disparity. From the depth the other coordinate values can be calculated directly. The computation of the depth however involves the knowledge of some camera parameters like the focal length (actual). To get such parameters the cameras should be calibrated first. The following two sections describe the camera calibration and the stereopsis that is followed in this thesis to compute the three coordinates of the centroids of detected objects in the segmented images with respect to the world frame.

5.1 Camera Calibration

As explained above, camera calibration is the process of computing useful camera parameters especially the focal length from the series of images taken by the camera using different techniques. The most common method is the Tsai equations (54). The calibration toolbox that is used in this project is the camera calibration toolbox for Matlab that is developed at California Institute of Technology (7).

First, a set of images (30) of a planar check board are taken using the ISAC's cameras. The toolbox requires 20 - 25 images to accurately calibrate the camera. The images that are used in the calibration are shown below.

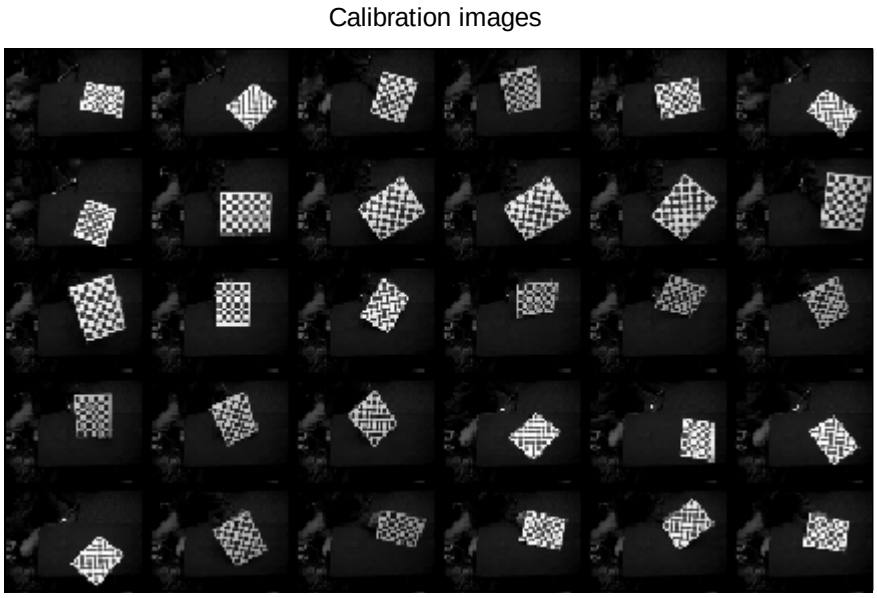


Figure 22: The 30 images that are used in the calibration process

Then corners of each check board in each image are extracted using the software. An example corner extraction is shown below.

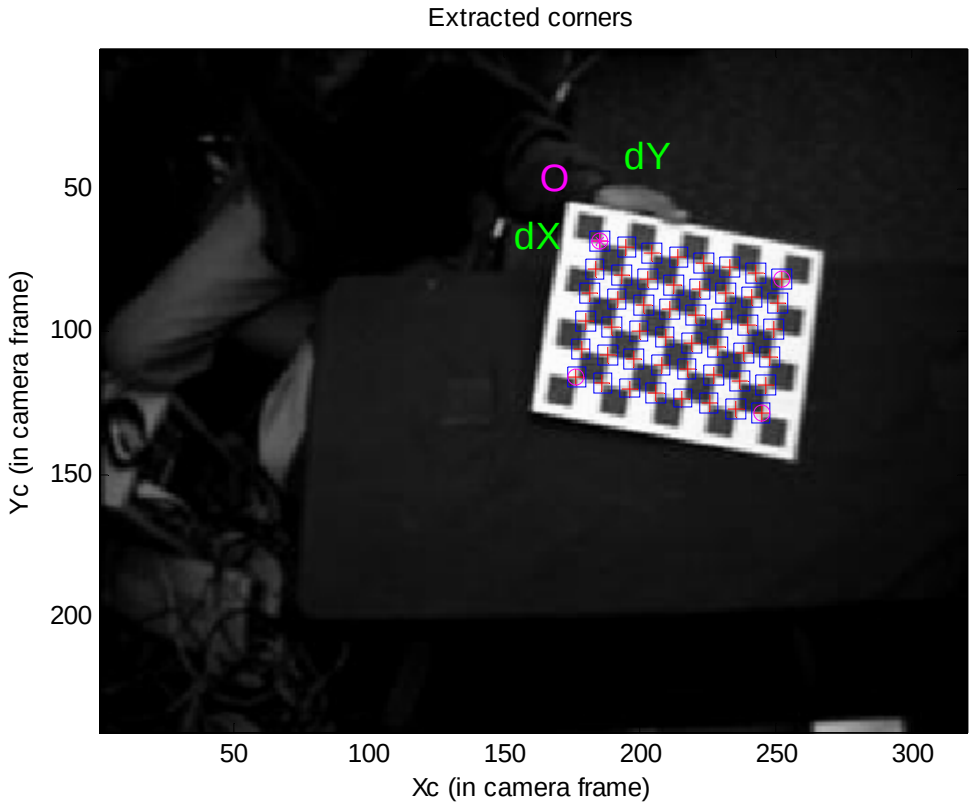


Figure 23: Example corners extracted from example image of the check board

The calibration results that are obtained using the first run of the 30 calibration images are shown below.

$$Focal\ Length: \quad f_c = [\ 310.73573 \quad 310.73573\ ]$$

$$Principal\ point: \quad cc = [\ 159.50000 \quad 119.50000\ ]$$

$$Skew: \quad \alpha_c = [\ 0.00000\ ] \Rightarrow angle\ of\ pixel = 90.00000\ degrees$$

$$Distortion: \quad k_c = [\ 0.00000 \quad 0.00000 \quad 0.00000 \quad 0.00000 \quad 0.00000\ ]$$

Since, the calibration toolbox uses a gradient decent method to iteratively compute the parameters; there is an optimization run to get the best results. After the optimization run, the following parameter estimations were obtained.

$Focal\ Length$: $f_c = [\ 320.21217 \quad 318.17184\ ] \pm [\ 8.20312 \quad 7.79713\ ]$

$Principal\ point$: $cc = [\ 162.86545 \quad 123.91991\ ] \pm [\ 7.18556 \quad 6.02375\ ]$

$Skew$:

$\alpha_c = [0.00000] \pm [\ 0.00000\ ] \Rightarrow angle\ of\ pixel\ axes$

$= 90.00000 \pm 0.00000\ degrees$

$Distortion$:

$k_c = [-0.21542 \quad 0.30542 \quad 0.00031 \quad -0.00032 \quad 0.00000\ ]$

$\pm [\ 0.04866 \quad 0.16166 \quad 0.00314 \quad 0.00652 \quad 0.00000\ ]$

$Pixel\ error$: $err = [\ 0.07453 \quad 0.22937\ ]$

Note: The numerical errors are approximately three times the standard deviations (for reference).

The software will also display the re-projection errors in the calculation and as can be seen in the following figure, the errors are fairly small. The maximum error in both dimensions is less than a pixel. The cameras' nominal focal length is 6mm and the parameter for pixels per mm as find out in (6) is $\sigma = 50\ pix/mm$. The focal length calculated is then calculated in mm using the average focal length

$$f = \left(\frac{320.21217 + 318.17184}{2}\right) pix\left(\frac{1mm}{50\ pix}\right)$$

$$\rightarrow f_{ave} = 6.38384\ mm\ (which\ is\ closer\ to\ the\ nominal\ value)$$

$$f_{ave} = 319.192\ pixels\ is\ used\ in\ the\ computation$$

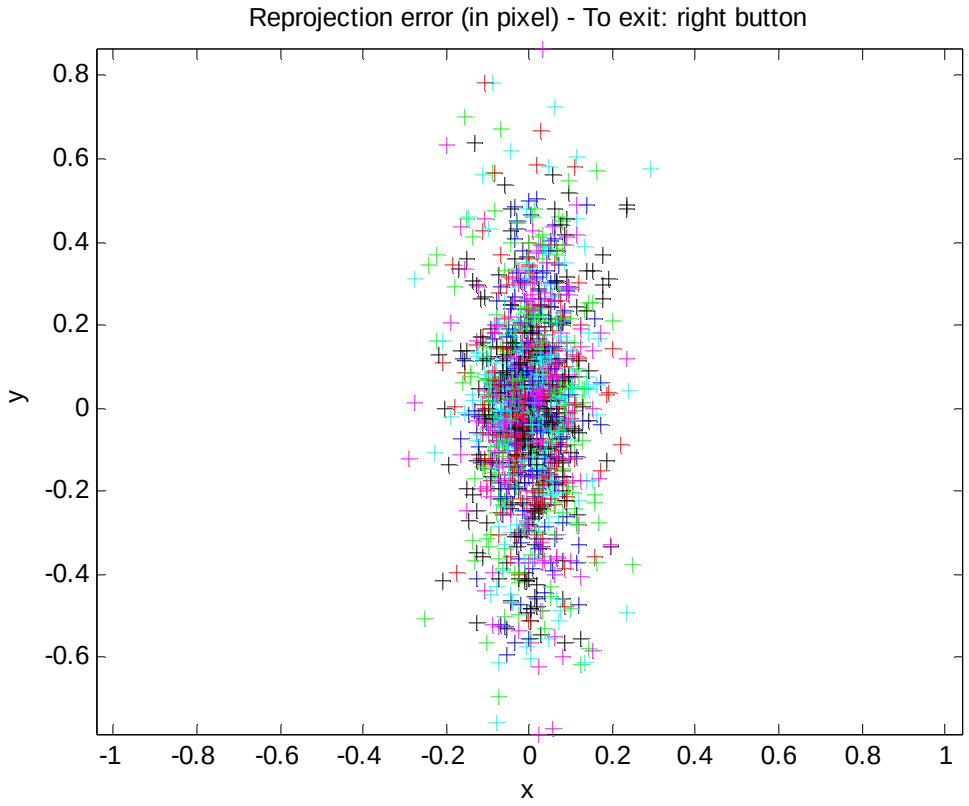


Figure 24: Re-projection errors in the computation of the camera parameters

5.2 Co-planar Steriopsis

After the focal length and the pixels per mm are known, the three coordinates of the centroids of the objects can be computed using stereopsis. There are two different methods to compute the coordinates. The general method involves computation of the rectification of the right and left images using homographies before computing the coordinates and is applied in general camera configurations. The second method involves

the computation of the coordinates from disparity directly without rectification. This method is applicable if the two camera image planes are coplanar or assumed to be coplanar. Due to its simplicity, and the zero pan angles used for both cameras, the second method is chosen for this project. It is an approximation of the general method and hence some intrinsic error is expected. The general method is the accurate method of computing the coordinates.

The method assumes that the two image planes are parallel or coplanar. The following figure shows the scenario and the following set of equations are derived from the geometry of the figure. These equations are used to calculate the coordinates (x, y, z) of a point with respect to the camera head frame that is located at the center of the two cameras (55).

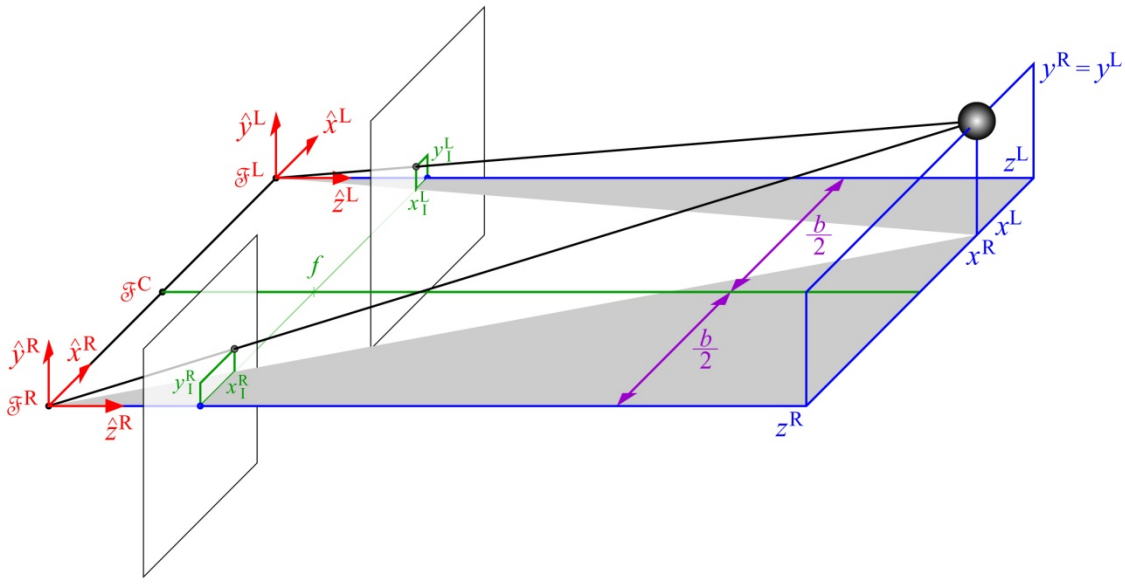


Figure 25: Geometrical configuration of the coplanar image planes, the respective frames of reference and object projection to the image planes. (55)

$$z^R = z^L = \frac{f * b}{x_I^R - x_I^L}$$

$$y^L = y^R = \frac{y_I^R * z^R}{f} = \frac{y_I^L * z^L}{f}$$

$$x^L = \frac{x_I^L * z^L}{f}$$

$$x^R = \frac{x_I^R * z^R}{f}$$

Equation 14: Coplanar Stereopsis Equations

The xyz values are then computed with the above formulae and the result was off the theoretical value by some factor for the depth and in the depth computation that factor was taken in to account. Therefore, the factor was calculated to get a closer z_{min} to the real value. The factor computed is $\alpha = 2.2556$. Then all the depths are multiplied by this factor before computing the x and the y coordinates to get the correct results.

The following figure shows the experimental setup of ISAC and a table in front of him. Using this physical measurements, configuration and field of view of the cameras that is shown in chapter 2, the theoretical minimum and maximum limits of the three coordinates are calculated for comparing the results of the experiment and are shown in the results section of experiment 4.

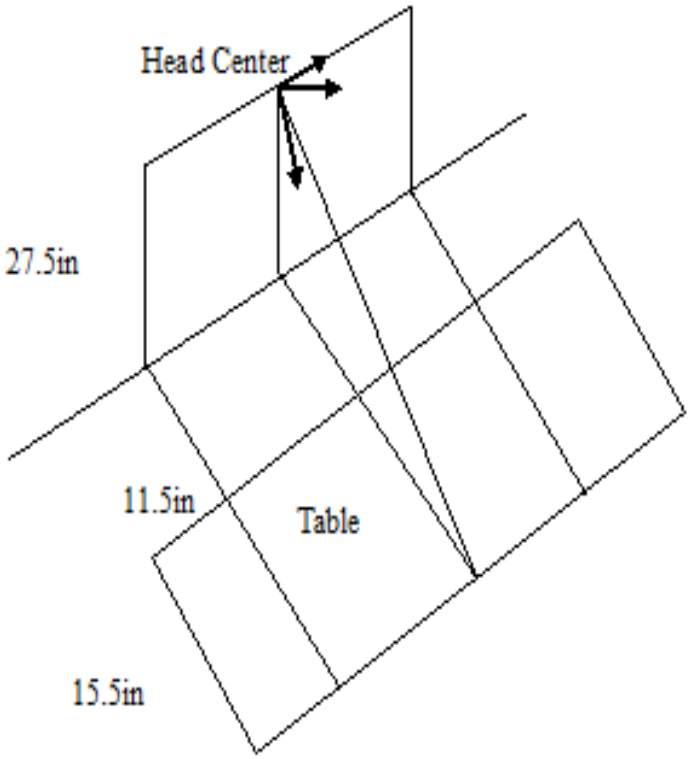


Figure 26: Experimental Setup of center of the head of ISAC and a table in front of him (not drawn to scale)

CHAPTER VI

PROPOSED EXPERIMENTS

The goal of this project is to investigate the efficiency of the random tree implementation of the nearest neighbor search for the purpose of object detection for a robot learning to classify objects with minimal help from human trainer. The experiments are specifically designed to show the time effectiveness and the accuracy of the system. The experiments are undertaken for several training and testing images and they are basically done on ISAC's vision system inside the CRL (cognitive robotics laboratory) of the center for intelligence (CIS) at Vanderbilt University, Electrical Engineering and Computer Science Department. The Test Objects are listed in the following table and are shown in the figure below.

Table 2: List of the experimental objects that are used in the project

Number	Object
1	Red Bean Bag
2	Blue Bean Bag
3	Green Lego Toy
4	Yellow Bean Bag
5	Purple Bean Bag
6	Orange Bean Bag

Figure 27: The Experimental objects that are used by this project

As the system is mainly using the pdf of the color histogram as discriminating feature, the objects are selected in such a way that they can reflect the typical different colors. To show that the system is rotation, translation and size invariant the images are taken by placing the objects at different positions and orientations and one big object relative to the others is included.

6.1 Experiment I

This experiment is designed for investigating the run times for the construction of the random tree. The random tree implementation is different than the pure (naïve) method in that all processing is not done at the time of classification. Instead, like other machine learning algorithms the tree construction is the training phase of the system. The system is presented with the training features that represent the test objects in this phase. The system is then able to construct the tree from this feature vectors. The constructed tree is then serialized for later classification stage.

The key performance criterion in this experiment is the training (tree construction) time. The training time is computed for six different sets of training images and also for six possible values of minimum number of leaf node vectors.

The number of the training images (hence, the number of training features) is varied as follows: the first tree is constructed using only one training image for each test object (6 pair of images). The second tree is constructed using two training images for each test object (12 pair of images) and so on until the sixth tree which is constructed using 6 training images for each test object (36 pair of images).

The minimum number training vectors that belong to a leaf node unless the node is a pure node is the other tree parameter that is varied from 50 to 300 in increments of 50. This effectively produces six distinct trees for each eye.

6.2 Experiment II

This experiment is concerned about the testing or classification run time of the system. It compares the run times using the tree implementation to that of the pure nearest neighbor search algorithm. For this purpose the function knnclassify that is already implemented in Matlab is used. One of the reasons for choosing Matlab for the second half part of the project is to get a common platform to compare the running times of the pure one to that of the random tree implementation as implementing the naïve algorithm is beyond the scope of this project. The second important reason to choose Matlab for the tree construction and search is that Matlab is very efficient in numeric manipulations especially in sparse representations, matrices and indexing them. However, it is well know that the system would be faster if implemented in C++. The image capture, the pre-processing and the feature extractions are done in C++.

Different combinations of the objects are used to get the testing images and totally 38 image pairs are used as testing images for testing run time comparisons. As the run time curves are clearly shown until cumulative feature vectors of 25 testing images and as the overall run times are very large, the experiment is performed until 27 batch of testing images. Two platforms are used to run the experiments in parallel and to see platform behavior of the systems run time. The Platforms are specified in the table below:

Table 3: Platform Specification of the test beds for experiment III

Name: My Laptop	Name: Sally (A computer that is used for vision system of ISAC)
Model: HP Pavilion dv6000	Model: Dell Desktop PC
RAM:2GB	RAM: 1GB
CPU: AMD Turion 64x2 Dual Core Processor with speed of 1.8 GHz each.	CPU: Intel Pentium IV dual core 3.06GHz, 3.05GHz
OS: Windows Vista Home Premium (32 bit)	OS: Windows XP Enterprise

Testing run times are compared using both number of training images (vectors) and training vectors.

6.3 Experiment III

The accuracy of the random tree implementation is compared to the pure algorithm in this experiment. This experiment is specifically designed to show the robustness of the system over the naïve implementation using ROC and accuracy as principal performance criteria. The setting of this experiment is similar to that of the second experiment, but it is separated as an independent experiment because the aspect of the system that it addresses is different to that of the above experiment.

6.4 Experiment IV

This experiment focuses on the performance measure of the system using the coordinates of the centroids of the objects that are computed using the approximate assumed coplanar method. This assumption may introduce inaccuracies in the computation but it is generally acceptable as the main focus of this thesis doesn't include cognition and hence very accurate coordinate may not be required.

For this experiment, a new set of 36 testing pair of images are taken with measured world dimensions that are used to calculate the limits in all the three dimensions.

CHAPTER VII

EXPERIMENTAL RESULTS AND DISCUSSIONS

In this chapter a detailed discussion of the experimental findings is presented together with the experimental results in various formats. The experimental setups are discussed in the previous chapter and hence this chapter focuses on the results and their discussion.

7.1 Training Results (Experiment I Results)

In this experiment, six pairs (for the right and left images, a total of 12) of search trees were created and trained for the 6 set of training data using the number of training images as parameter that is explained in the experimental setups in the previous chapter. Then, six other pairs (for the right and left images, a total of 12) of search trees were trained using the minimum number of training vectors per leaf node. The number of training vectors that are used for the first set of search trees and the set of the minimum number of training vectors per impure leaf node are given in the tables below.

Table 4: Number of training images used and training vectors extracted out of them for the old set of training images

Search Trees	Number of training images used		Number of training vectors used	
	Right	Left	Right	Left
1	6	6	423	413
2	12	12	828	805
3	18	18	1203	1185
4	24	24	1595	1555
5	32	32	2040	1949
6	36	36	2440	2321

Table 5: Number of training images used and training vectors extracted out of them for the new set of training images

Search Trees	Number of training images used		Number of training vectors used	
	Right	Left	Right	Left
1	6	6	475	467
2	12	12	941	921
3	18	18	1467	1424
4	24	24	1954	1892
5	32	32	2414	2367
6	36	36	2880	2827
7	42	42	3358	3320
8	48	48	3824	3761
9	54	54	4277	4197

From the tables above, we can see that the number of training vectors that are extracted out of the right and left images representing the same scene is different. This is partially due to the defect on the right camera that is explained above. It captures noisier images than the left ones and even though a different noise filtering parameters are used, it still produces a little bit more than the left counterpart. If the noise filtering parameter is increased further it will start to suppress even the objects features. The training time result is shown in the following plot.

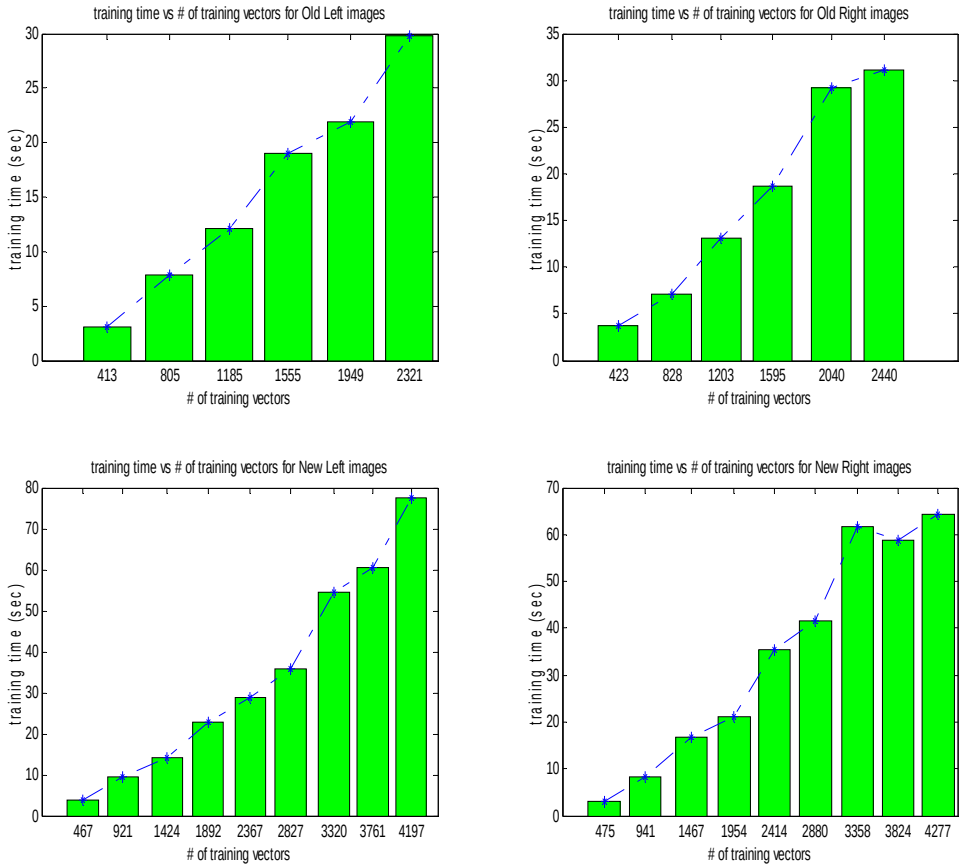


Figure 28: plot of the training times of the old (36 training images) and new image set (54 training images) versus number of training vectors

The plot above depict that the training times vary approximately linearly with the number of training vectors and they are in the orders of 7.42 ms for each training vector. This is the cost that is paid to get an optimal testing time in the test runs. The total training times for even 54 training images is well below 100 seconds. This is an extra overhead over the training of the pure one, because the pure method does all the processing at testing time.

The minimum number of training vectors per leaf node is also varied from 50 to 300 with increments of 50 for the old set of training images and the following performance is obtained.

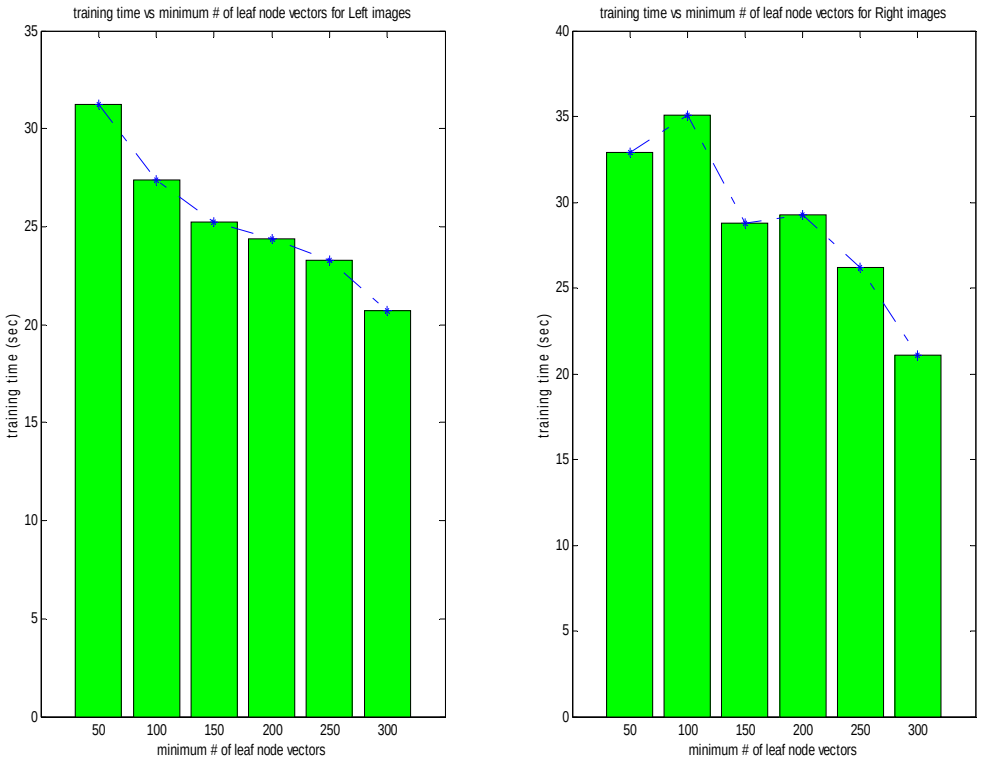


Figure 29: training times in seconds versus minimum number of training vectors per impure leaf node

It is expected that as the minimum number of training vectors in an impure node increases, the total number of nodes will decrease as more vectors can be contained in a single impure leaf node. Hence, the tree becomes shallower and the training time will decrease with increase in the parameter.

The figures above clearly show this fact. The right tree is at some points against this behavior. That can be attributed to the defect of the right eye and the randomness of

the tree. Since the centers of children nodes of a parent node are selected randomly, sometimes the leaf nodes might be more pure and hence shallower tree and sometimes the tree might contain more impure nodes and deeper tree and that will affect the training time a bit. However, the random points are selected using a random number generator that generates a uniformly distributed set of points in a range and hence the discrepancy in the randomness of the tree is a little. This can be seen the above plots that there is only one point in the right search trees that took longer than the previous tree and one that took shorter than the next tree.

Overall, this experiment can be called a success and it clearly shows us that the running time of the training with the number of training vectors is approximately linear or $O(n)$, if the number of training vectors is n.

7.2 Testing Time Results (Experiment II Results)

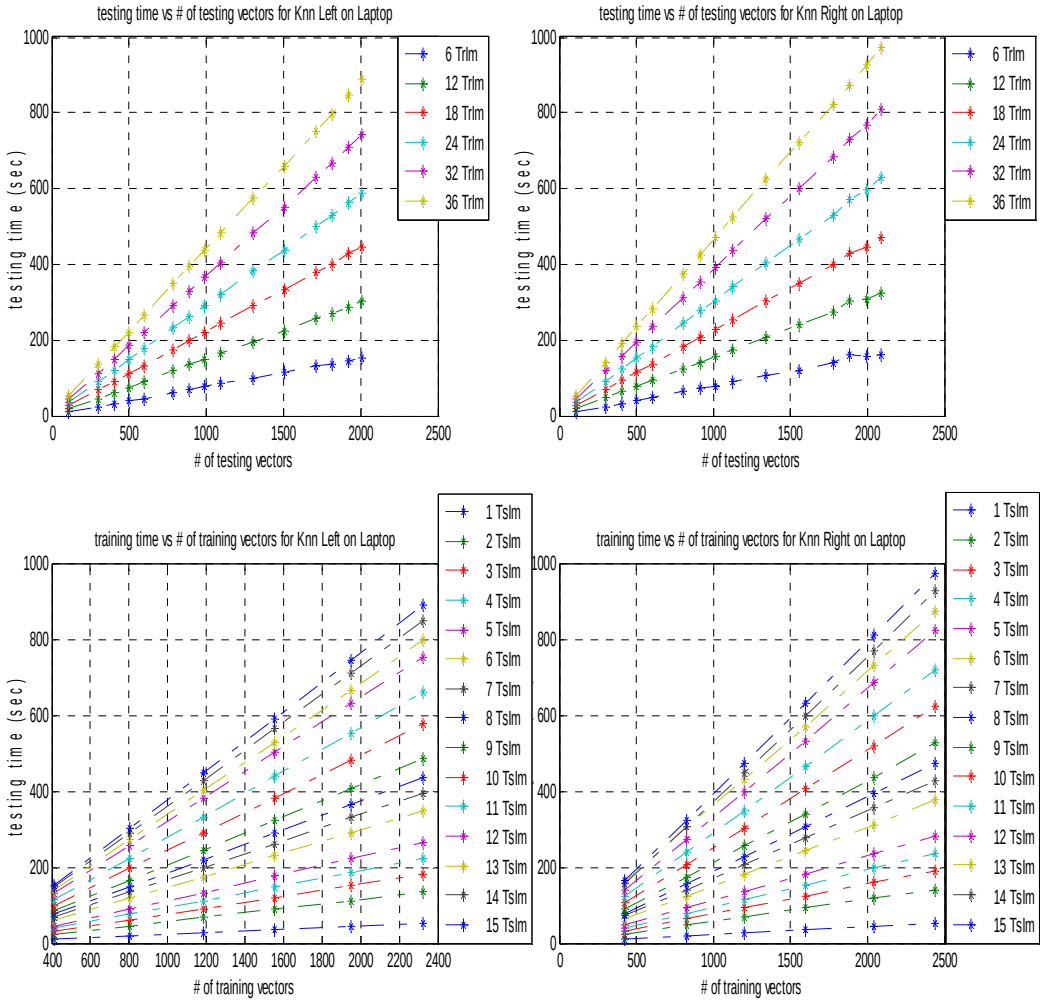


Figure 30: Plots of testing time of the pure nearest neighbor algorithm with number of testing vectors for the right and left images (top) and with number of training vectors (bottom) TrIm represents Training Image and TsIm represents Testing Image in the legends

Figure 30 shows that the run time of pure (naïve) nearest neighbor search algorithm is approximately quadratic with the number of testing vectors and linearly increases with the number of training vectors. This is the result that is obtained by running on the laptop. The result that is obtained on the desktop pc (Sally) will be

discussed shortly. This result shows that the testing time becomes very large as the number of both testing and training images gets larger. Naturally, a nearest neighbor search needs much training vectors to get a good accuracy and hence, the pure method will get the system slower. The reason for this slow classification is primarily that all computation occurs at the time of classification.

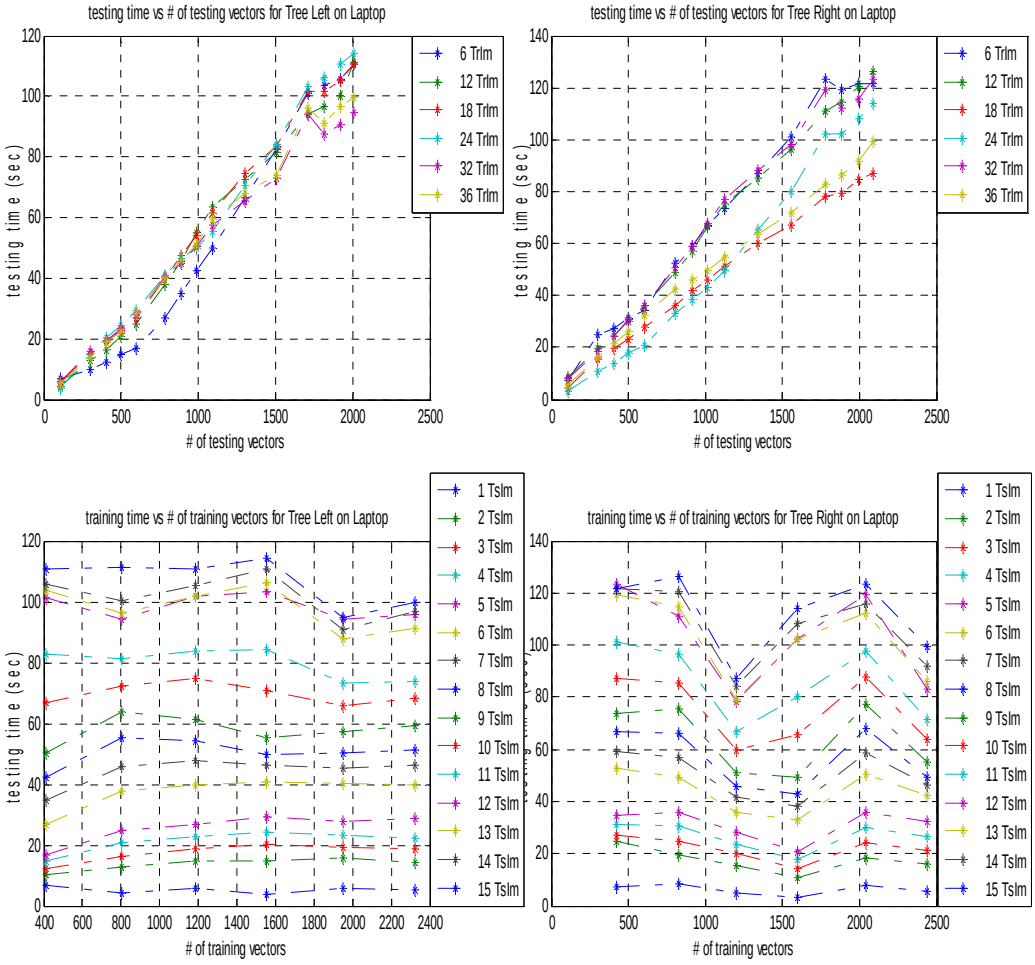


Figure 31: Plots of testing time of the random tree for both left and right images with number of testing vectors (above) and with number of training vectors (below) TrIm represents Training Image and TsIm represents Testing Image in the legends

This result is very encouraging that shows the random tree implementation is surely an approximation with good running time at the time of classification. The running time with respect to the number of testing vectors first seems linear and starts to bend as the number of testing vectors increases. This effect will be shown in the result obtained using Sally as the number gets bigger.

Since the random tree implementation hold the center of a node as a representative of the training vectors in the node, there is only one comparison performed in each node and the search is directed to the similar branch and hence the training time is not affected significantly by the increase in the number of training vectors. The results show that it is more or less constant time with number of training vectors.

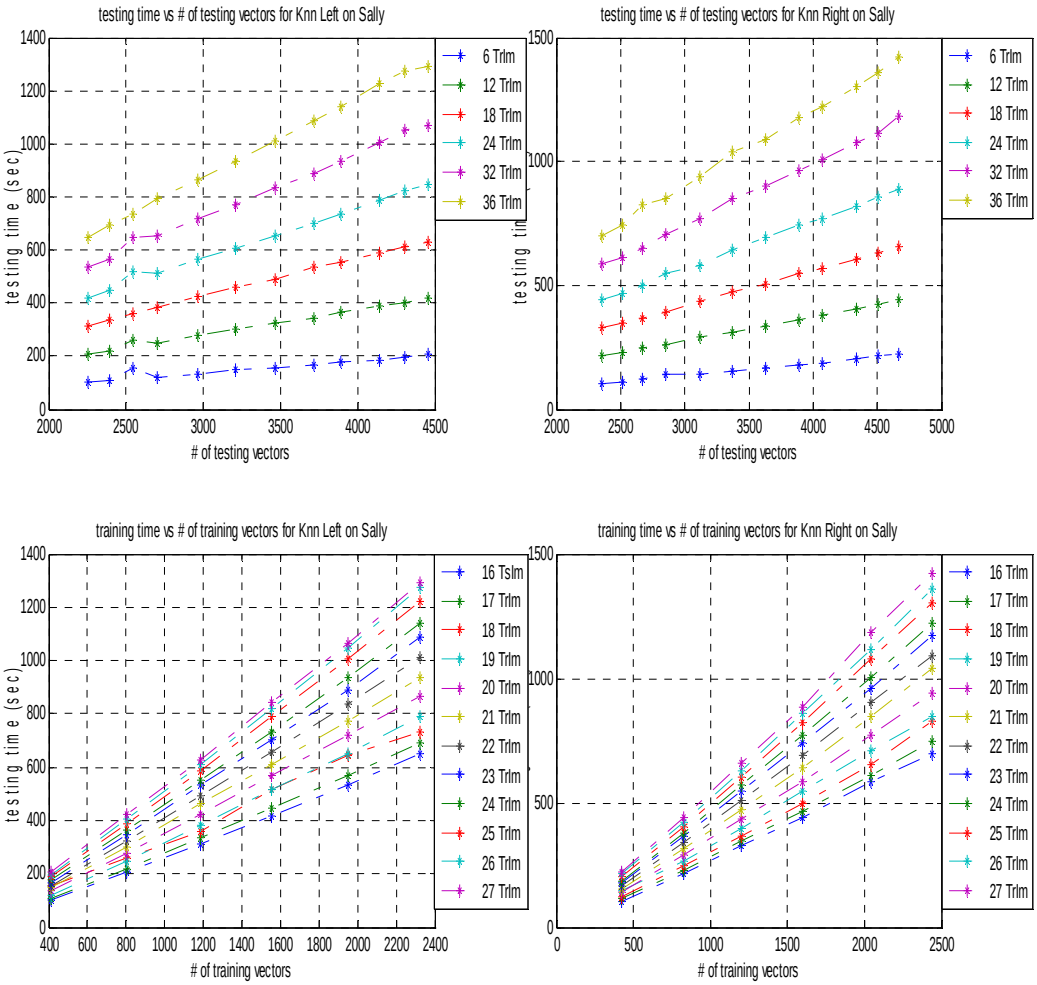


Figure 32: Plots of testing time of the pure algorithm with number of testing vectors and with number of training vectors. This is the result of program running on Sally.

The above is the result of runs on Sally. The number of testing vectors keeps increasing from where it has stopped on the laptop runs discussed above. Here it starts with vectors from 16 testing images and ends on vector sets from 27 set of testing images. This clears shows that as the number of testing vectors becomes larger and larger the run time becomes more quadratic ($O(Nd)$) where N is the number of training vectors

and d is the dimensionality of the vectors. The runtime is also keeps linear ($O(n)$) as the number of training vectors (n) gets larger.

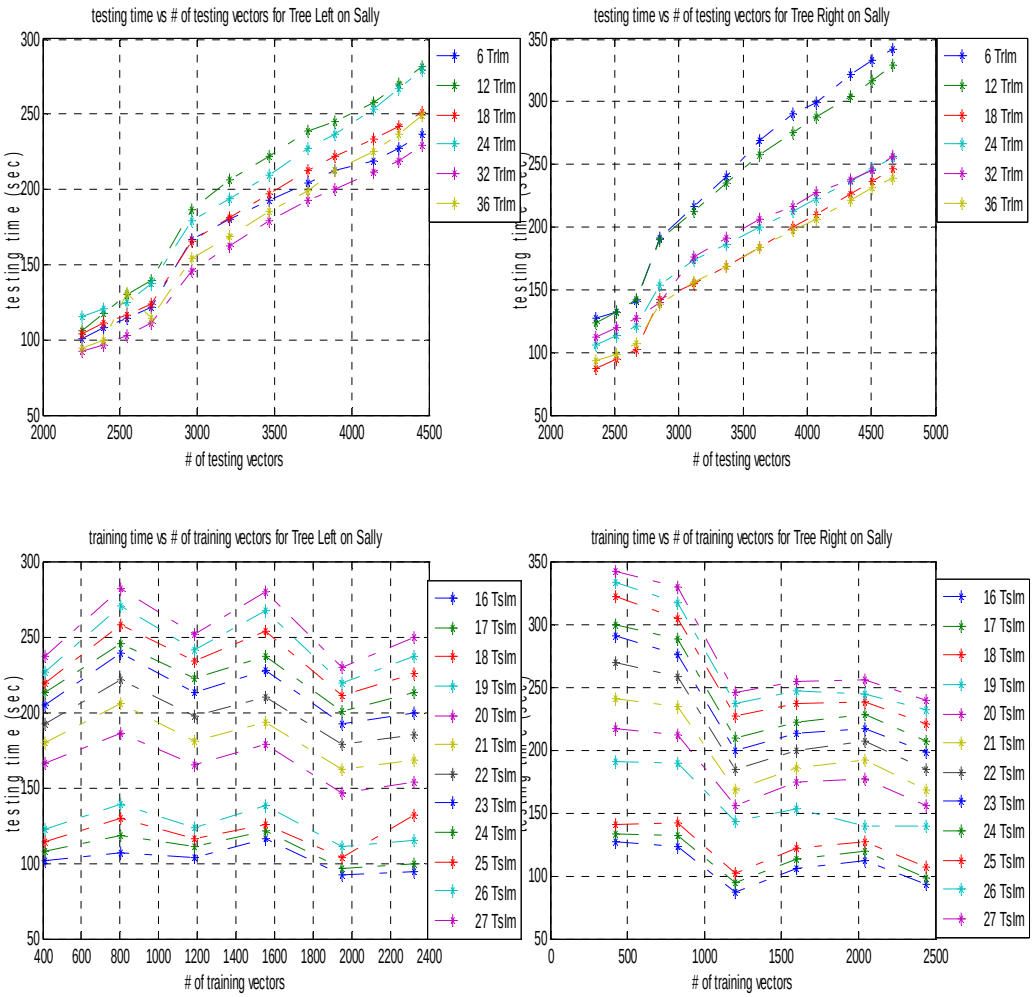


Figure 33: Plots of testing time with number of testing vectors and with number of training vectors that run on Sally.

The run time of the random tree implementation becomes more logarithmic as the number of testing vectors keeps increasing and seems to keep constant with number of training vectors. Due to the randomness of the tree sometimes the run time even

decreases even with increasing number of training vectors. But, overall it can be approximated as linear.

7.3 Testing Accuracy Results (Experiment III Results)

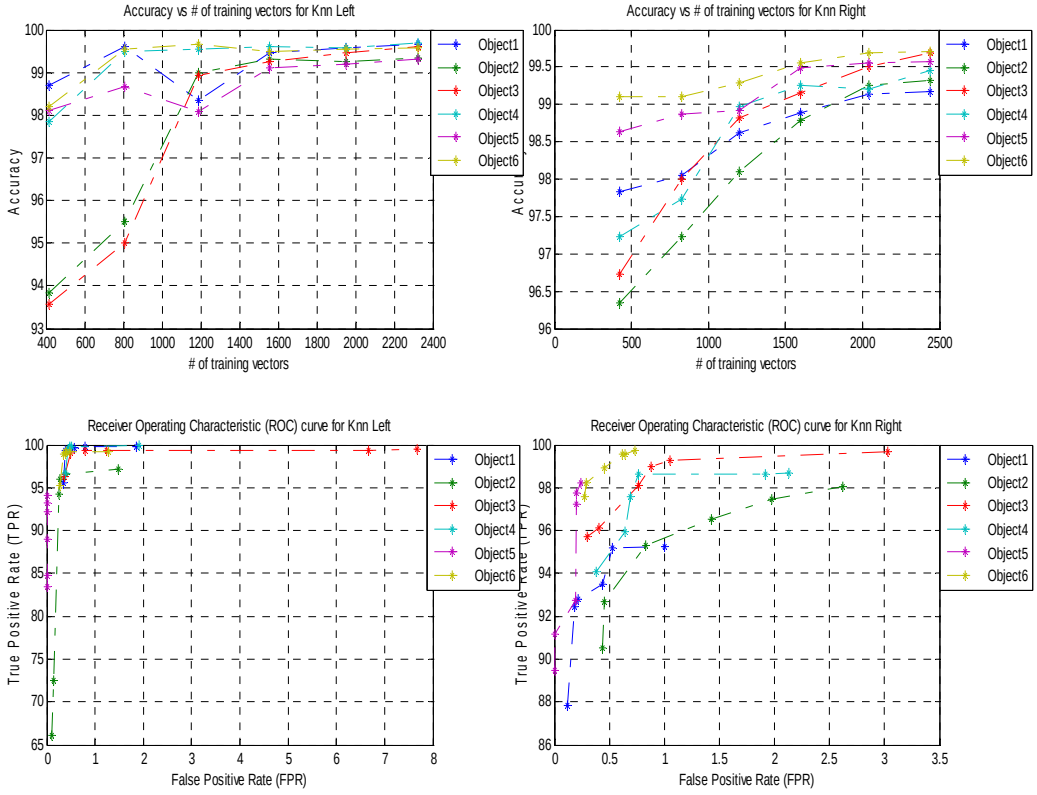


Figure 34: Testing accuracy plots and ROC curves for individual test objects of the pure nearest neighbor search method.

The performance plots in this figure show that the accuracy of the system is high and consistent. The accuracy seems to flatten out with as the number of training vectors

becomes large. This shows that after a certain limit, the addition of training vectors does not add to the overall accuracy.

Due to the difference in color sensitivity to light conditions and texture, the objects seem to significantly contribute to their individual testing accuracy. However, as there is more and more training vectors available from each object the testing accuracies become closer and closer. This suggests that a fairly large amount of training database is required to get optimal result for any kind of object using the nearest neighbor search algorithm.

The ROC curves for individual images are also pretty much consistent with the typical ROC curve with minimal maximum false positive rate which is less than 10 percent and hence the search trees are in the normal operating region of the ROC curves.

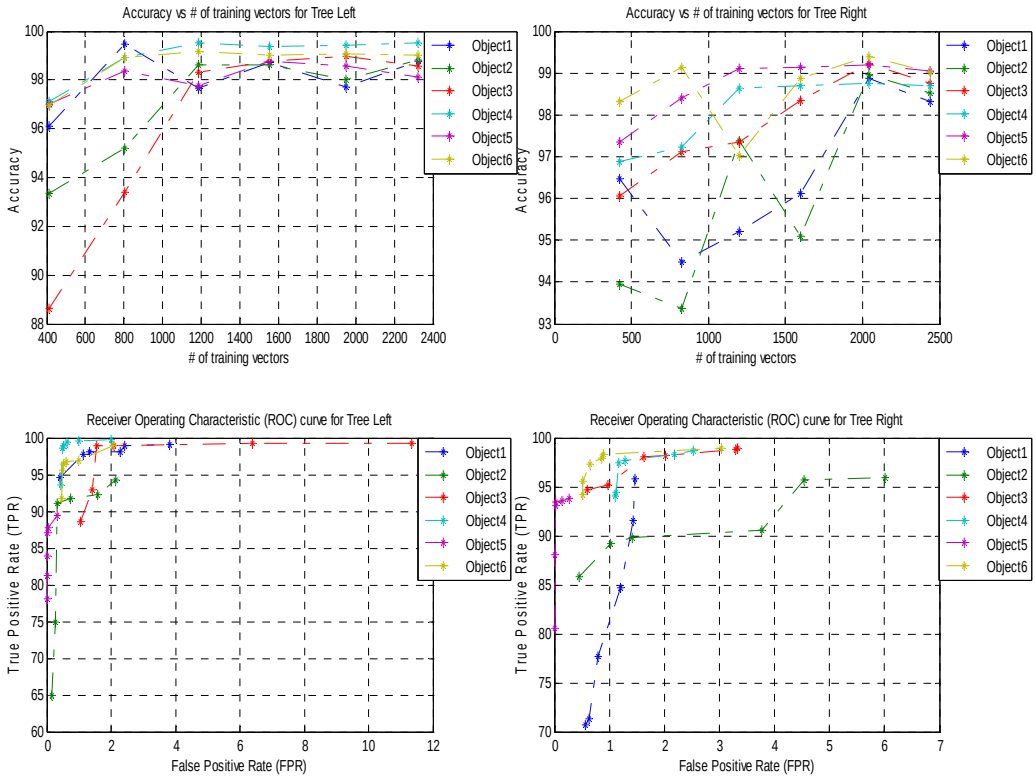


Figure 35: Accuracy of individual test objects with number of training images and ROC curves.

The results above show that the performance of the random implementation is very comparable to that of the pure implementation that is discussed above. The accuracy plot for the left search trees seems almost similar to the pure implementation. The right trees seem to have slightly variable accuracies across objects at the beginning but come closer as the number of training vectors gets larger.

The ROC curves also show that the search trees are operating in the better operating region of the ROC curve with maximum false positive rate of a little bit greater than 10 percent. This is closer to that of the pure implementation.

As we discussed above, increasing the number of training vectors beyond a certain limit does not increase the accuracy and sometimes it might even have a negative effect on it as shown in the right accuracy plot above. As the training vectors become larger in number the accuracy starts to decrease at last. This shows that not all of training vectors contribute to the overall accuracy. There might be a lot of redundancy and may be discrepancy.

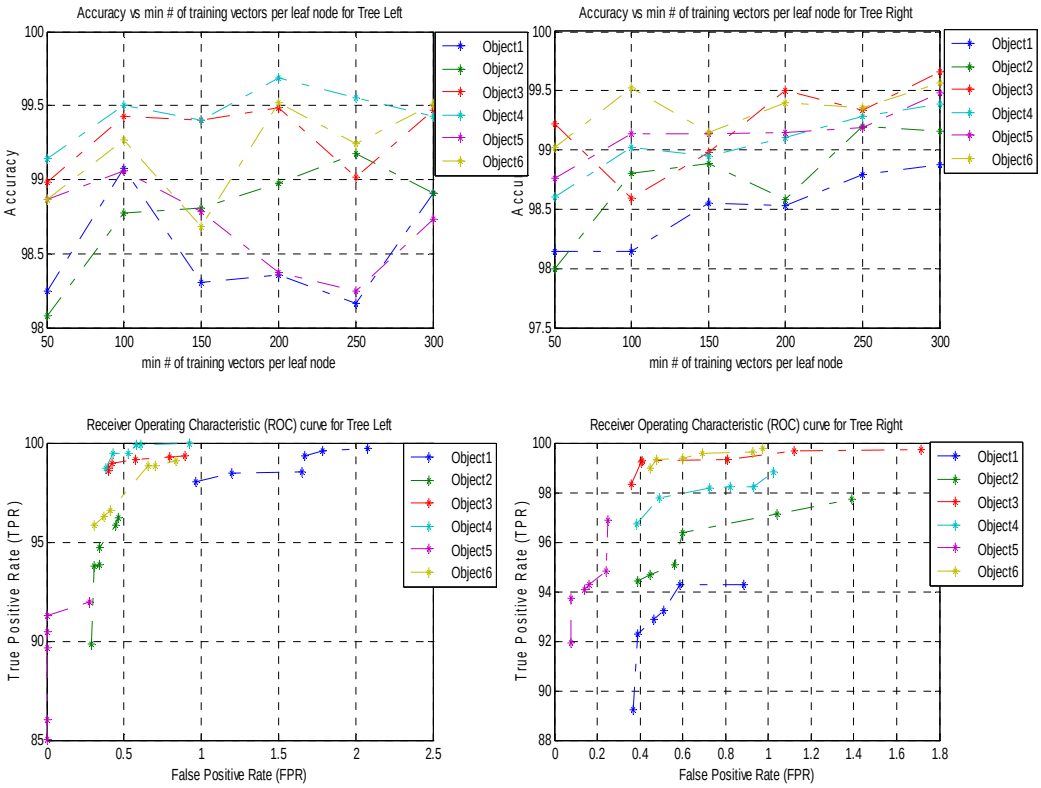


Figure 36: Accuracy plots of the random tree implementation with the minimum number of training vectors in an impure node and the corresponding ROC curves for individual test objects

The best set of training images is used in this particular experiment which is 6 testing images for all the 6 test objects. So, the number of training vectors is constant for this experiment to see the effect of the minimum number of training vectors per each

node. On the above experiments, the minimum number of training vectors per leaf nodes was kept constant and it was 100. As we can see here 100 is an optimal value in accuracy with just 1 percent below the accuracy with minimum number of leaf node training vectors of 300 with significant saving in training and testing time.

The results above suggested that minimum number of leaf node training vectors above 50 has sufficiently high accuracies and increasing the parameter beyond that has little gain in accuracy but with time penalty.

The summary of the performance comparisons of the pure nearest neighbor search implementation to the approximate nearest neighbor search with random tree implementation is given in the table below.

Table 6: Summary of the comparison of the pure nearest neighbor algorithm with approximate random tree implementation

	Pure (Naïve) implementation	Random Tree implementation
Training Time	Not Applicable	$O(n)$
Testing Time Vs TsVr	$O(Nd)$	$O(\log(N))$
Testing Time Vs TrVr	$O(n)$	$O(1)$
Space	$O(Nd)$	$O(Nd + N)$
Accuracy	Higher	Comparably Higher
False Positive Rate	Lower	Comparably Lower

7.4 Stereopsis Results (Experiment IV Results)

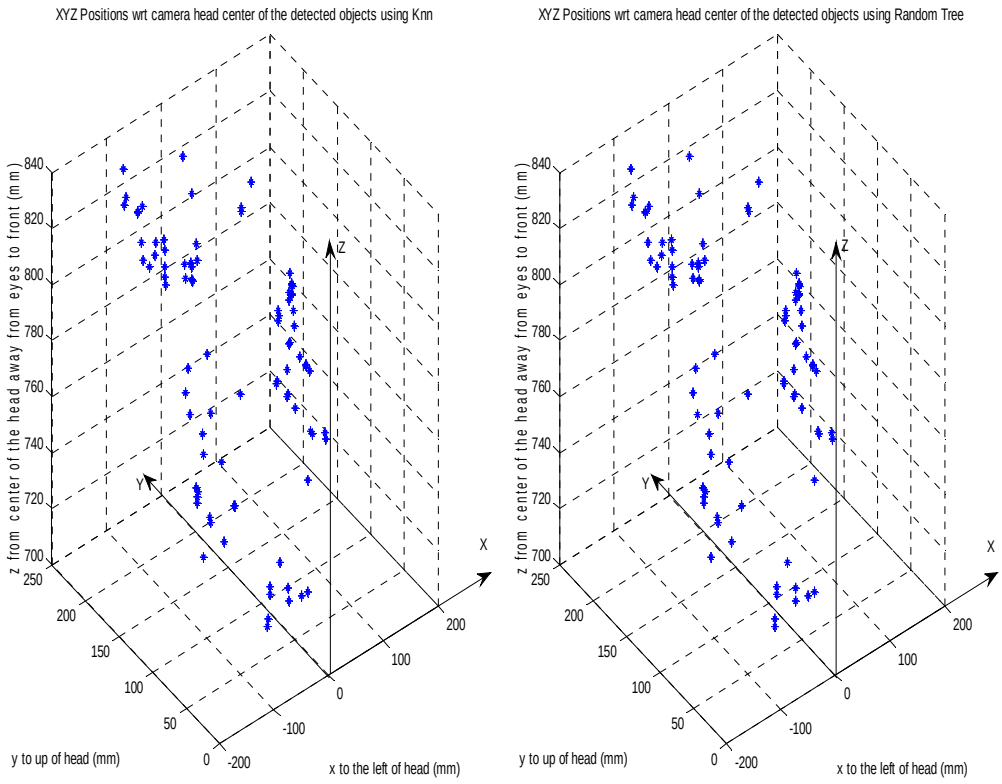


Figure 37: 3D plots of the coordinates of the 352 centroids of the detected experimental objects (dimensions are in mm)

Table 7: Theoretically computed 3D coordinate ranges

	X	Y	Z
Minimum	-269.2474 mm	-37.764 mm	725.748 mm
Maximum	269.2474 mm	334.48665 mm	853.9210 mm

Table 8: Experimentally found 3D coordinate ranges for both pure and random tree implementations

	X	Y	Z
Minimum (pure)	-119.1697 mm	26.2413 mm	717.9863 mm
Maximum (pure)	111.9748 mm	225.9821 mm	838.4570 mm
Minimum (Tree)	-119.1697 mm	26.2413 mm	717.9863 mm
Maximum (Tree)	111.9748 mm	225.9821 mm	838.4570 mm

The above results illustrate that the coplanar assumption was quite good. The calculated three dimensional coordinates are almost in the theoretical range except the Z_{min} and that is just off the minimum by 7.7617 mm. This is quite a good result because it is assumed that the camera image planes were parallel approximately. If the values were computed using the general non-coplanar technique using homographies, the results would be more accurate than the above results.

From the 3D plots we observe that almost all the points seem to lie on a plane that is at a certain angle to the xy-plane. This is good because the experimental objects were all lying on the table in front of ISAC in all the experimental images.

CAHPTER VIII

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The perceptual learning system that is employed in this work shows very promising results. The approximate nearest neighbor search: random tree implementation shows a very good run time performance and outfits the pure implementation by a large extent in the overall run time performance with a comparably high accuracy. The system was able to maintain its internal representation of the test objects and was able to classify them without explicit programmer information like thresholds encoded to the system. This is one of the desired requirements of the system.

The logarithmic classification run time will make it equal to most efficient implementations like the kd-tree while the ability to incorporate very high dimensional feature spaces makes it superior. The most widely known approximate nearest neighbor search algorithm which is the kd-tree implementation called ANN (52) is able to classify feature vectors as high dimensional as only 20 while the random tree implementation that is used in this work was able to go up to 10001 and has a potential for even higher dimensions.

The comparisons made in Table 2 are actually based on a sparse vector representation for both pure and approximate method used in this work. As long as the

sparse coding is used, increasing the feature vector dimension will not affect the processing time. Because the total number of non-zero feature vectors elements is dependent on only the size of the image patch which in this work was 225 from which they are extracted not on the actual dimension which in this work was 10001. The use of a very high dimensional feature space makes the system as discriminatory as possible by incorporating a higher resolution in color.

In general, even though the system is tried in a relatively less complex and unstructured experimental setting for the goal of making an algorithmically sound comparisons, it can be proved robust even in more complex and unstructured environments for example with rapidly changing background.

8.2 Future Work

There can be a lot of future progress on this work to integrate it to a robotic cognition and control like ISAC and the implementation can be used even in other vision system that need high response time and accurate recognition with less complex representation and processing.

One of such future improvements would be integrating this algorithm to ISAC's visual system and develop a more robust vision for ISAC by further experimenting with the algorithm and comparing it with other machine learning algorithms like neural networks and support vector machines.

The other improvement that can be made to the system is to modify it so that it can incorporate novelty detection. Since the search trees keep distance statistics in each node about the statistics of the distance of the training vectors in that node from the center of the node, novelty detection can easily be integrated easily. If a new object is detected, the distance from the feature vector obtained from its images the center of any leaf node will be greater than the maximum distance and hence by making simple comparison with the maximum, a new object can be detected. In this classification run, the feature vectors from the new object can then be added to the tree leaf node under consideration very easily.

The other improvement that can be added is the use of the general epipolar geometry based stereopsis algorithm for the computation of the centroids. An attempt has been done to incorporate this in this work, but due to time limitation it was not incorporated in this work. The homographies that are computed for the cameras of ISAC can be used to rectify the segmented images before computing the coordinates of the centroids.

The incorporation of additional features in the feature vector like shape features more texture feature like the Gabor texture measures that are introduced in (2) can be another future direction for the system. Using basically many different features in the feature vector will make the system more accurate. However, caution should be taken not to significantly affect the desirable logarithmic run time performance.

REFERENCES

1. **Norvig, Stuart J. Russell and Peter.** *Artificial Intelligence A Modern Approach, second edition.* New Jersey : Printice Hall, 2003.

2. **Wang, Xiaochun.** *A Vision-Based Perceptual Learning System For Autonomous Mobile Robot, PhD Dissertation.* Nashville : Vanderbilt University, Graduate School, 2007.

3. **Tugcu, Mert.** *A Computational Neuroscience Model With Application to Robot Perceptual Learning, PhD Dissertation.* Nashville : Vanderbilt University, Graduate School, 2007.

4. **Sony, Corporation.** *XC-999/999P Video Camera Module Oprating Manual.* Tokyo, Japan : s.n., 1991.

5. **Sony, Co.** *VCL-06S12XM fixed focus (6mm) lenses.* Tokyo : Sony, 1991.

6. **Beagley, Sean.** *Gesture Recognition and Mimicking In a Humanoid Robot.* Nashville : Vanderbilt University, 2008.

7. **Bouguet, Jean-Yves.** CalTech Camera Calibration Matlab Toolbox. *Camera Calibration Toolbox for Matlab.* [Online] June 2, 2008. http://www.vision.caltech.edu/bouguetj/calib_doc/.

8. **Perception, Directed.** Directed Perception. [Online] 2 2009. http://www.dperception.com/pdf/specs-ptu-d46.pdf .

9. **Imagenation, corporation.** Imagenation Vision System Specialists. *PXC200 Precision Color Frame Grabber Manual.* [Online] imagenation, December 1997. http://www.imagenation.com/dnfiles/pxc/mn_200_02.pdf.

10. **Intel, Corporation.** Intel Co. *Intel Open Computer Vision Library* . [Online] Intel, Februray 7, 2009. http://opencv.willowgarage.com/wiki/.

11. **Wikipedia.** Wikepedia-OpenCV. *Wikipidia the free encyclopedia.* [Online] 2009. http://en.wikipedia.org/wiki/OpenCV.

12. **Bradski, Gary.** OpenCV 2009-06 Wiki. *OpenCV Wiki.* [Online] 2009. http://opencv.willowgarage.com/wiki/OpenCV200906.

13. **Wikipedia, wiki.** Wikipedia-Noise Reduction. *Wikipedia the free encyclopedia.* [Online] Wikepedia.org, 2009. http://en.wikipedia.org/wiki/Noise_reduction.

14. **Kaehler, Adrian and Bradski, Gary.** *Learning OpenCV: Computer Vision with the OpenCV Library.* Sebastopol, CA : O'Reilly, 2008.

15. **Review, Digital Photography.** Fujifilm latest CCD. *Digital Photography Review.* [Online] Digital Photography Review. [Cited: 03 14, 2009.] http://www.dpreview.com/news/article_print.asp?date=0809&article=08092210fujifilmEXR.

16. **Dale Purves, David Fitspatrick, S. Mark Williams, james O. McNamara, George J. Augustine, Lawrence C. Katz, and Anthony-Samuel LaMantia.** Anatomical Distribution of Rods and Cones. *Neuro Science: Second Edition.* [Online] Sinauer Associates, Inc., 2001. [Cited: 03 06, 2009.] http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=neurosci.section.762.

17. **Hubel, David H.** Eye, Brain, and Vision. *Harvard University Medical School.* [Online] Harvard University, 1995. [Cited: 03 06, 2009.] http://hubel.med.harvard.edu/index.html.

18. **Wiki, Wikipedia.** Visual Cortex. *Wikipedia.* [Online] Wikipedia.org. [Cited: 03 06, 2009.] http://en.wikipedia.org/wiki/Visual_cortex.

19. **Paragios, Nikos, Chen, Yunmei and Faugeras Olivier.** *Handbook of Mathematical Models in Computer Vision.* New York : Springer, 2006.

20. **Gonzalez, Rafael C. and Woods, Richard E.** *Digital Image Processing: second edition.* Upper River Sadle, New Jersey : Prentice Hall, 2002.

21. **Gevers, Theo and Smeulders, Arnold W.M., ISIS Faculty of WINS, University of Amesterdam.** *Color based Object Recognition.* Amsterdam

22. **Hunter, J.E., Wilkes, D.M., Levin, D.T, Heaton, C., Saylor, M.M.** *Autonomous Segmentation of Human Action for Behaviour Analysis.* Nashville : Vanderbilt University.

23. **Hunter J.E., Tugcu M., Wang X., Costello C., Wilkes D.M.** *Exploiting Sparse Represntations in Very High Dimensional Feature Spaces Obtained from Patch Based Processing.* Nashville : Vanderbilt University.

24. **Schwarz Margaretha, Grewe Lynn, and Kak Avi.** *Represntation of Color and Segmentation of Color Images.* Purdue University, School Of Electrical Engineering.

25. **Juraj, Horvath.** *Image Segmentation Using Clustering.* Kosice : Technical University of Kosice, Slovak Republic.

26. **Peter, Comaniciu Dorin and Meer.** *Robust Analysis of Feature Spaces: Color Image Segmentation.* Piscataway, NJ : Rutgers University, Department of Electrical and Computer Engineering.

27. **Bruce James, Balch Tucker, and Veloso Manuela.** *Fast and Cheap Color Image Segmentation for Interactive Robots.* Pittsburgh, PA : Carnegie Mellon University, School Of Computer Science.

28. **Mitchell, Tom M.** *Machine Learning.* McGraw-Hill Science/Engineering/math ; School of Computer Science; Carnegie Mellon University, 1997.

29. **Wiki, Wikipedia.** HSL and HSV Color spaces . *Wikepedia: The free encyclopedia.* [Online] Wikipedia.org. [Cited: 03 10, 2009.] http://en.wikipedia.org/wiki/HSL_color_space.

30. **authors, A group of contributing.** CV Reference Manual. *Intel Open Source Computer Vision Library.* [Online] University of Pensylvania. [Cited: 03 10, 2009.] http://www.seas.upenn.edu/~bensapp/opencvdocs/.

31. **Vinje, Willimam E., and Gallant, Jack L.** Sparse Coding and Decorrelation in Primary Visual Cortex during Natural Vision; University of California at Berkley. *Science Magazine.* 2000, Vol. 287, 5456.

32. **Tran, Thanh N.** *Knn Density-Based Clustering for High Dimensional Multispectral Images.* Nijmegen : University of Nijmegen, The Netherlands, 2003.

33. **Beis, Jeffrey S., and Lowe, David G.** *Shape Indexing Using Approximate Nearest-Neighbor Search in High-Dimensional Spaces.* Vancouver, B.C., Canada : Department of Computer Science, University of British Columbia.

34. **Anna Atramentov, and LaValle, Steven M.** *Efficient Nearest Neighbor Searching for Motion Planning.* Department of Computer Science, Iowa State University and University of Illinois Urbana.

35. **Wiki, Wikipedia.** Nearest Neighbor Search. *Wikipedia: The free encyclopedia.* [Online] wikipedia.org. [Cited: 03 14, 2009.] http://en.wikipedia.org/wiki/Nearest_neighbor_search.

36. **Wiki, Wikipedia.** Mahalanobis distance. *Wikipedia: The free encyclopedia.* [Online] wikipedia.org. [Cited: 03 14, 2009.] http://en.wikipedia.org/wiki/Mahalanobis_distance.

37. **Kollar, Thomas.** *Fast Nearest Neighbors: cover tree implementation, Litrature Review.*

38. *Similarity, kernels, and the triangle inequality.* **Jakel, Frank, Scholkopf, Bernhard, Witchmnann, Felix A.** Berlin, Germany : Elsevier Inc.; Journal of Mathematical Psychology, 2008, Vols. 52: pages 297-300.

39. *Toward A Universal Law of Generalization for Psycological Science.* **Shepard, R.N.** 1987, Vols. 237: Pages 1317-1323.

40. *Attention and the Metric Structure of the Stimulus Space.* **Shepard, R.N.** Journal of Mathematical Psychology, 1964, Vols. 1: pages 54-87.

41. *The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function.* **Shepard, R.N.** Psychometrica, 1962, Vols. 27: pages 125-140.

42. **Mozaffari, Saeed, Faez, karim and Ziaratban, and Majid.** *Character Representation and Recognition Using Quadtree-based Fractal Encoding Scheme.* Tehran, Iran : Electrical Engineering Department, University of Technology.

43. **Panigrahy, Rina.** *Nearest Neighbor Search using Kd-trees.* Computer Science Department, Stanford University, 2006.

44. **Motwani, Piotr Indyk and Rajeev.** *Approximate Nearest Neighbors: Toward Removing the Curse of Dimensionality.* Stanford, CA : Department of Computer Science, Stanford University, 1999.

45. **Liu, Hongzhi, et al.** *Fast Image Segmentation using Reagion Merging with a K-Nearest Neighbor Graph.* Shangai, China : Department of Computer Science and Engineering, Fudan University.

46. **Cano, Javier, Perez-Cortes, Juan-Carlos and Salvador, and Ismael.** *Comparison of Two Fast Nearest-Neighbor Search Methods in High-Dimensional Large-sized Databases.* Valencia, Italy : Politechnique University of Valencia.

47. **Berchtod, Stefan, et al.** *Fast Nearest Neighbor Search in High-dimensional Space.* Munich, Germany : Institute for Computer Science, University of Munich.

48. **Liu, Ting, et al.** *An Investigation of Practical Approximate Nearest Neighbor Algorithms.* Pittsburgh, PA : School of Computer Science, Carnegie Mellon University.

49. **Garcia, Vincent, Barlaud, Michael and Debreuve, and Eric.** *Fast K-Nearest Neighbor Search using GPU.* Sophia Antipolis, France : University of Nice-Sophia Antipolis.

50. **Ruhl, D. Karger and M.** *Finding Nearest neighbors in Growth Restricted Metrics.* Proceedings STOC, 2002.

51. **Lee, R. Krauthgamer and J.** *navigating Nets: Simple Algorithms for Proximity Search.* s.l. : Proceedings of the 15th Annual Symposium on Discrete Algorithms (SODA), pages 791-801, 2004.

52. **David M, Mount and Sunil Arya.** ANN: A Library for Approximate Nearest Neighbor Searching . *David M Mount's Website.* [Online] Department of Computer Science, University of Maryland. [Cited: 03 14, 2009.] http://www.cs.umd.edu/~mount/ANN/.

53. **Wiki, Wikipedia.** Receiver Operating Characteristics (ROC). *Wikipedia: The free encyclopedia.* [Online] wikipedia.org. [Cited: 03 15, 2009.] http://en.wikipedia.org/wiki/Receiver_operating_characteristic.

54. *A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses.* **Tsai, Roger Y.** No. 4, IEEE Journal of Robotics and Automation, 1987, Vols. RA-3: Pages 323-344.

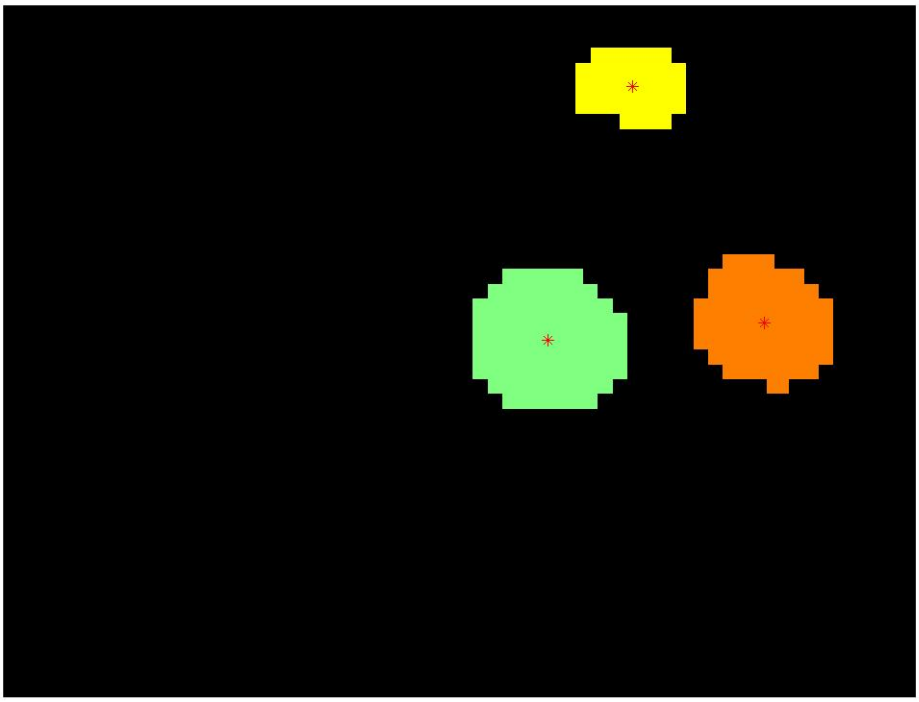
55. **Peters, Alan II.** *Coplanar Stereopsis, Computer Vision Lecture Notes.* 2008.

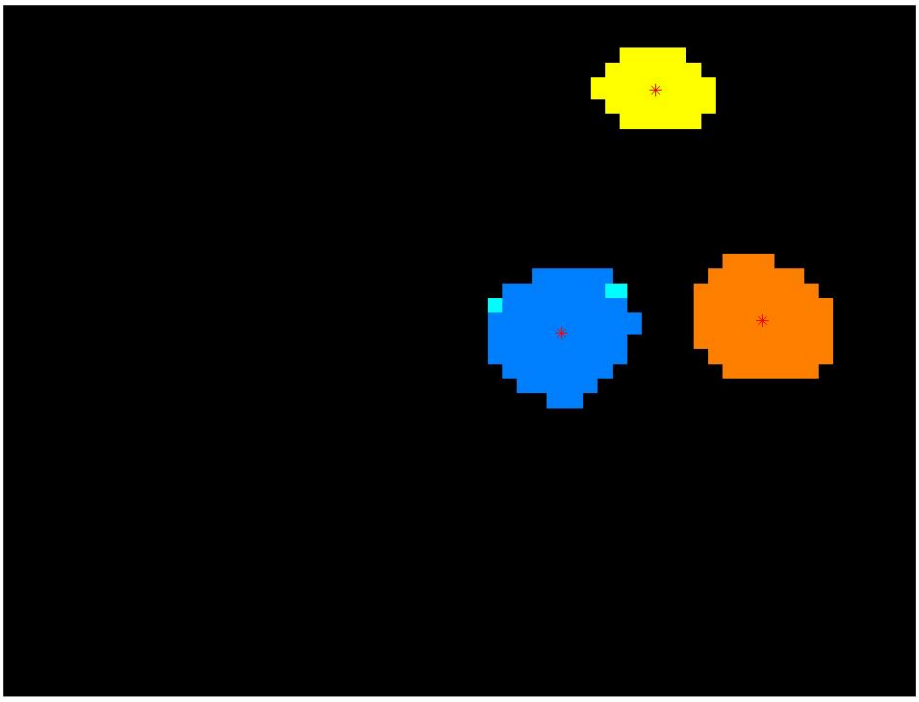
APPENDIX

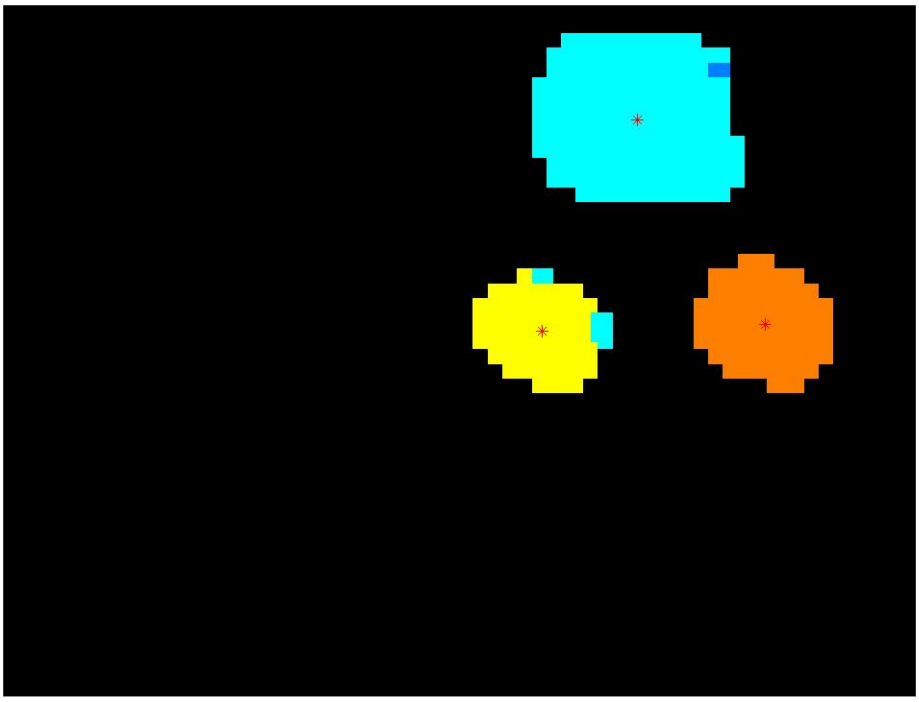
SELECTED SEGMENTED IMAGES

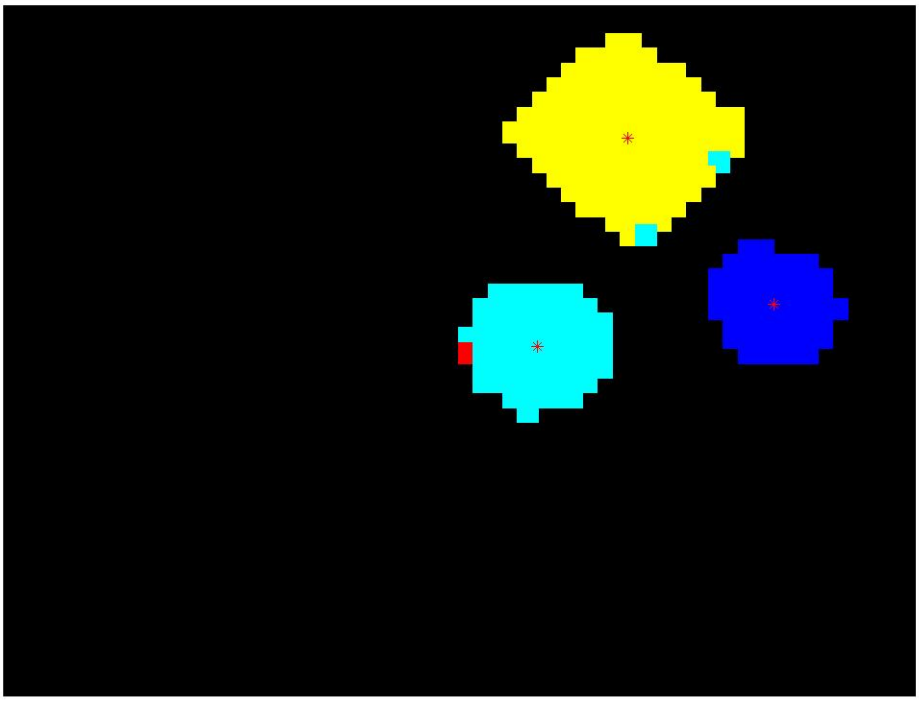
KNN

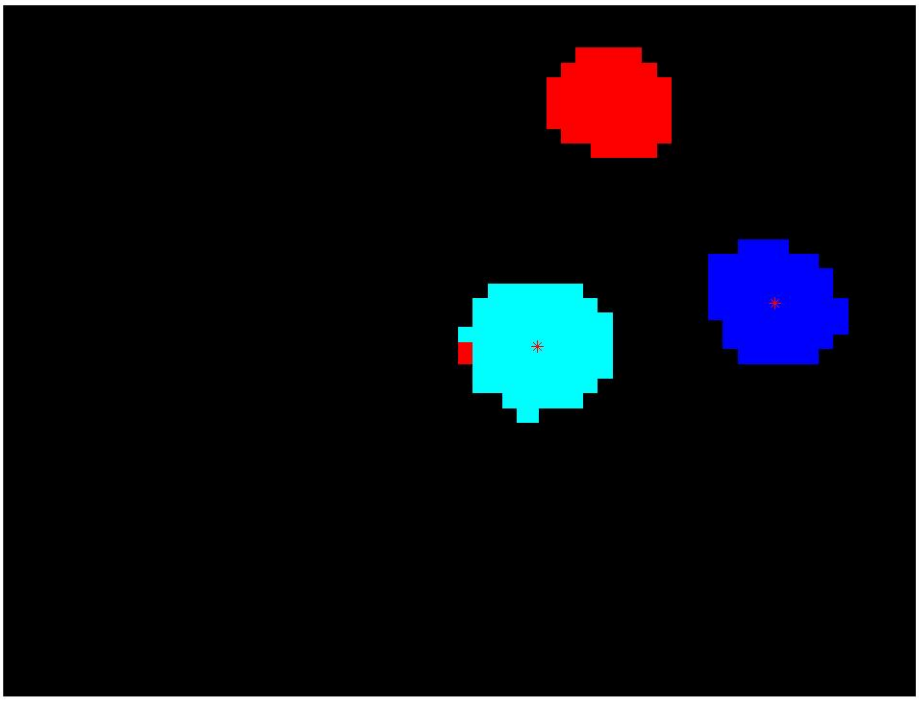
Left

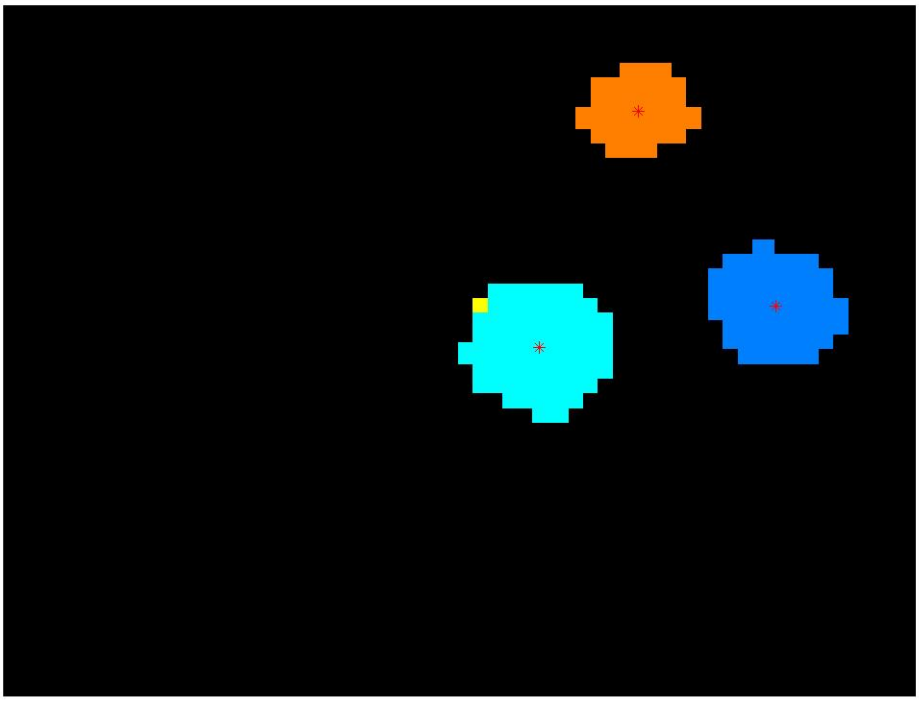


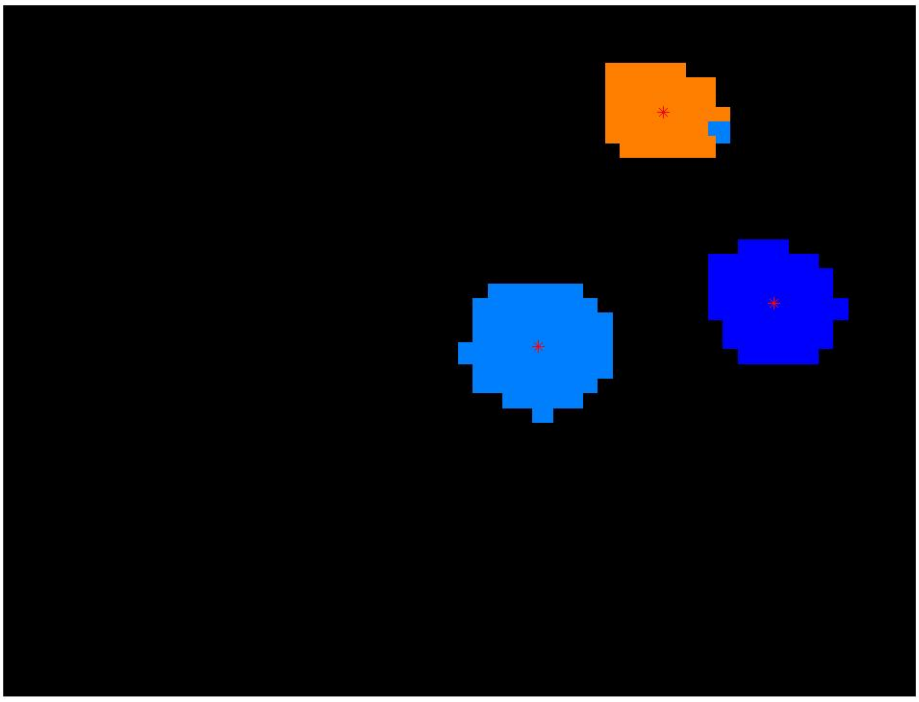


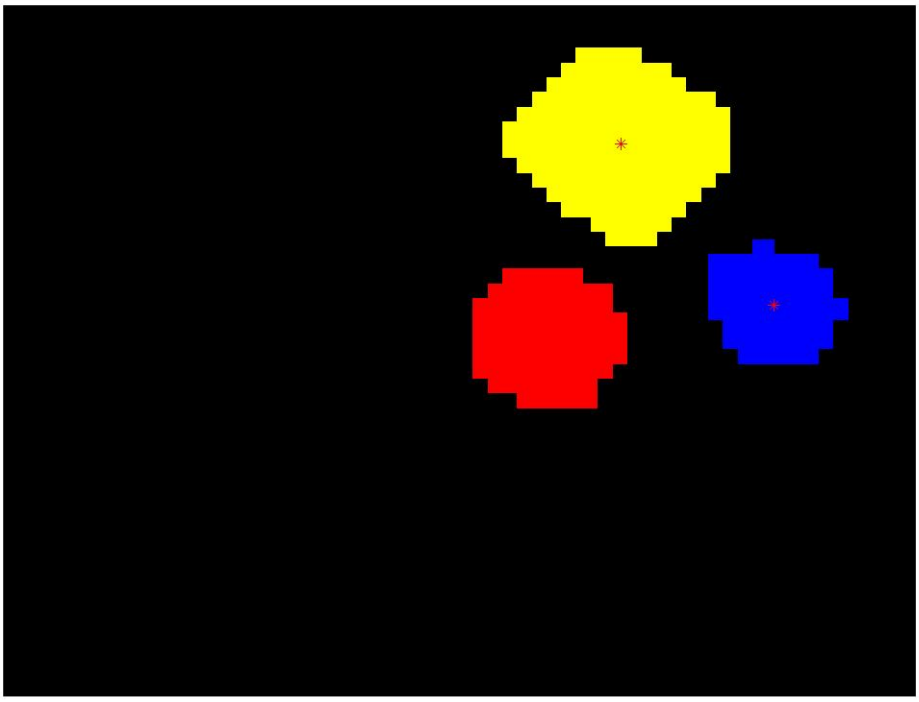


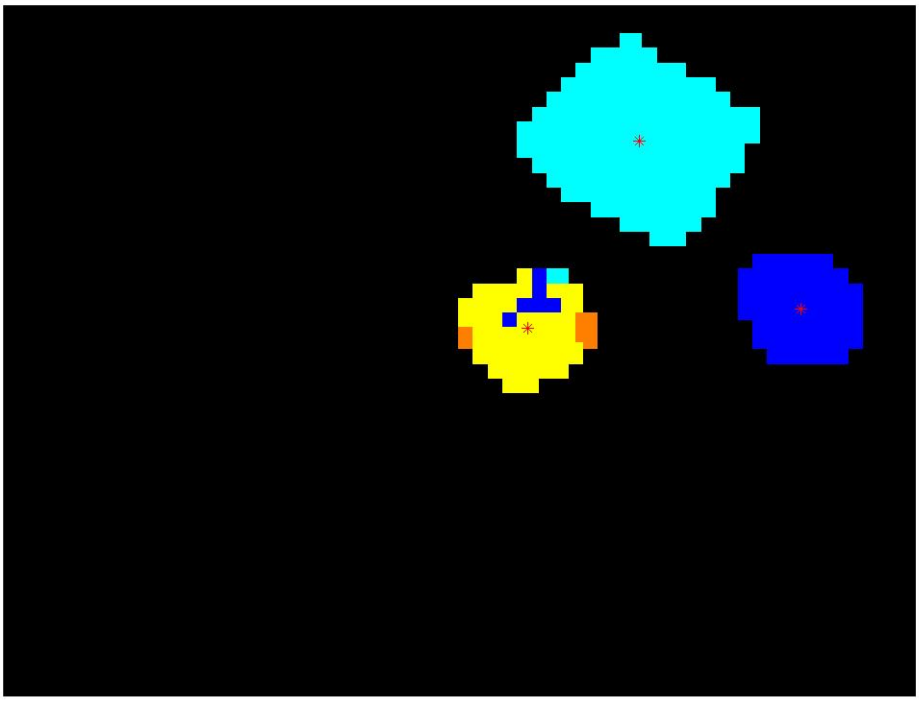


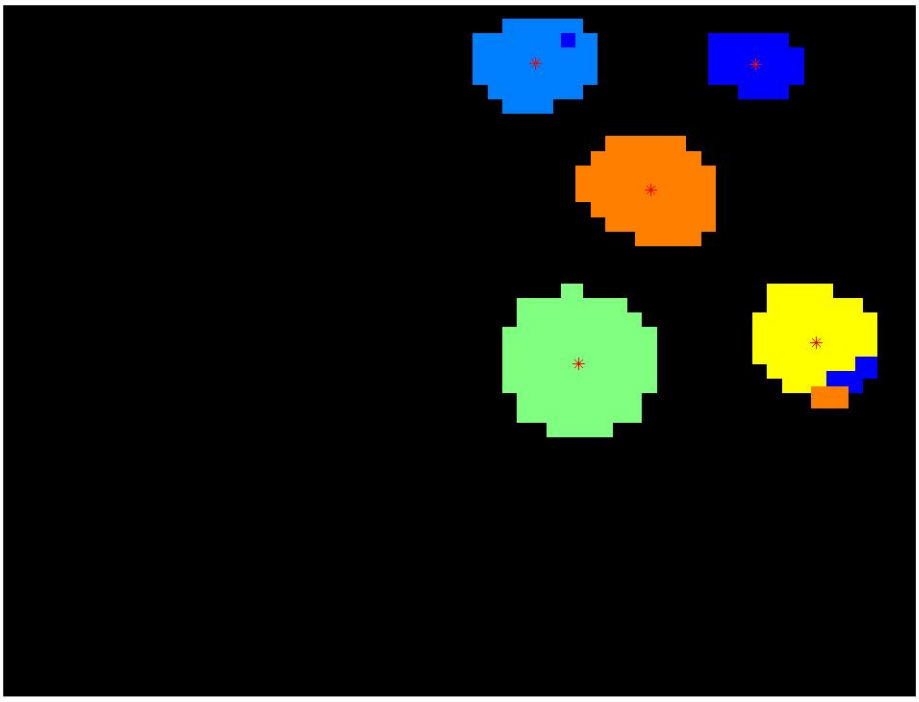


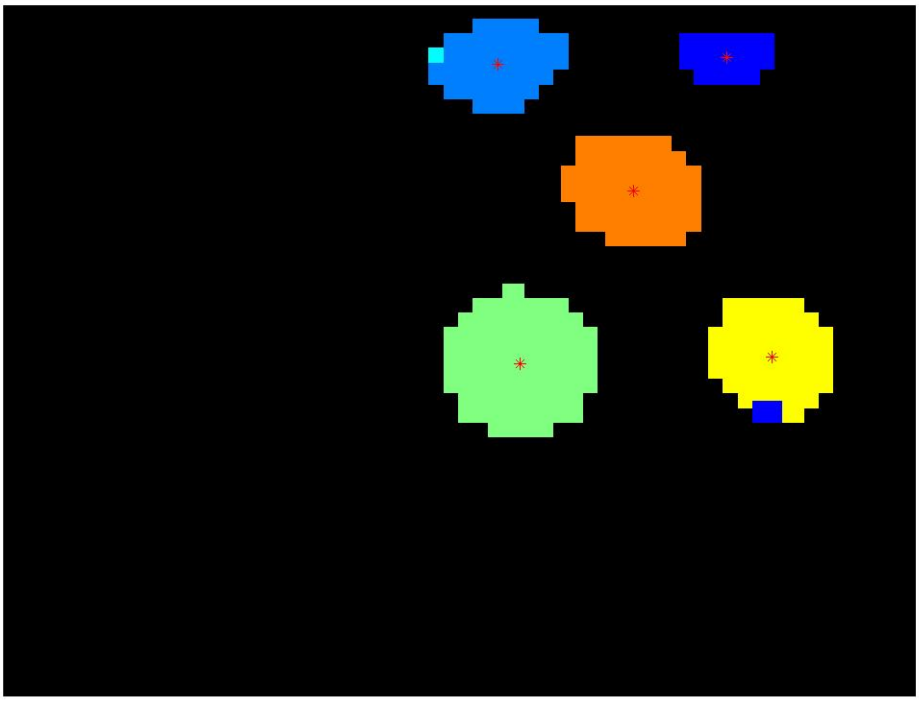


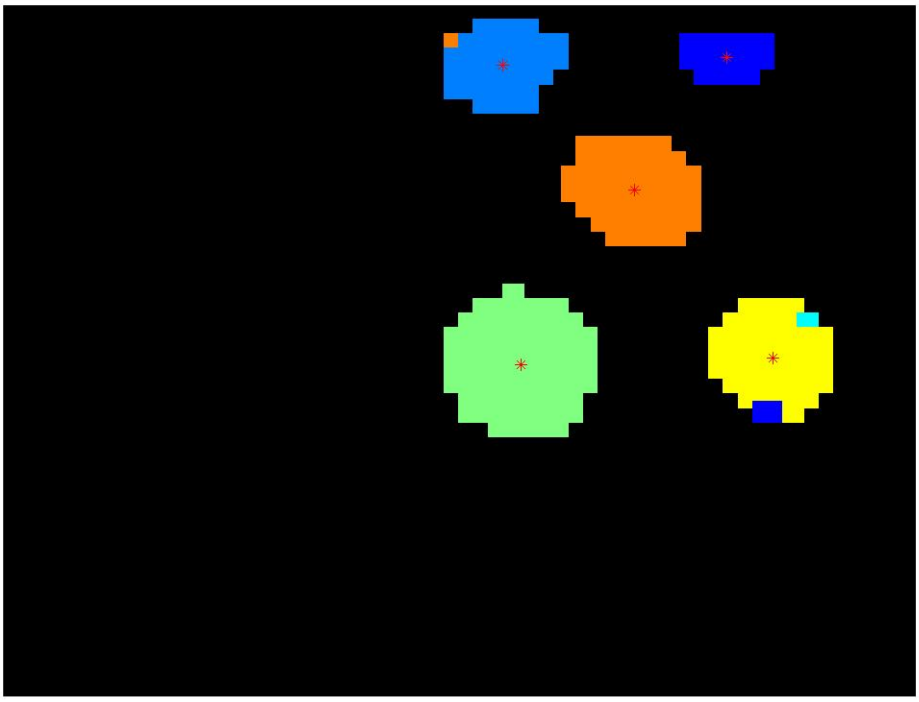


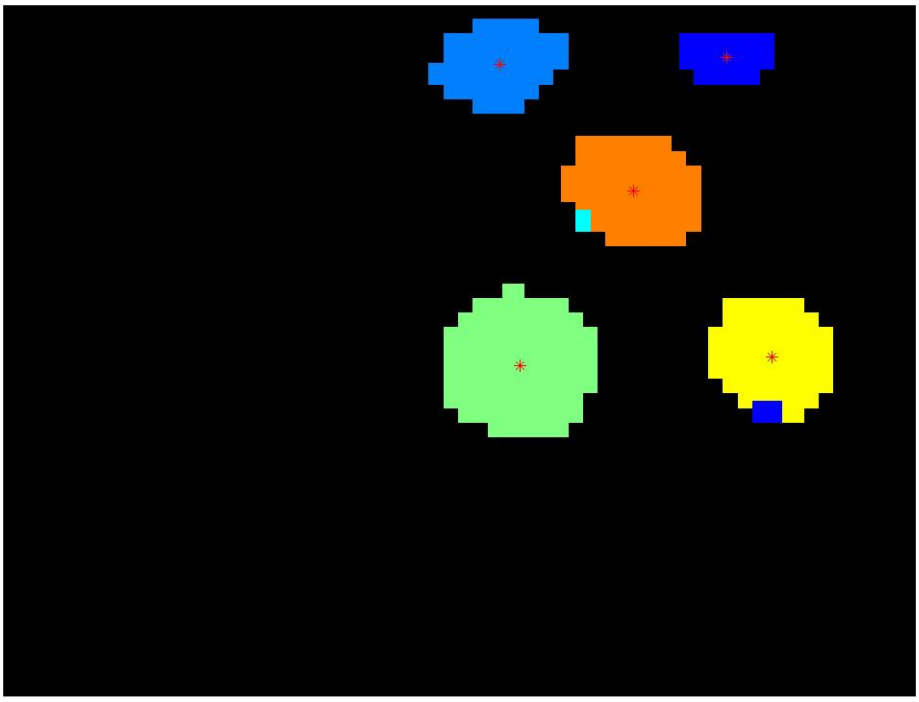




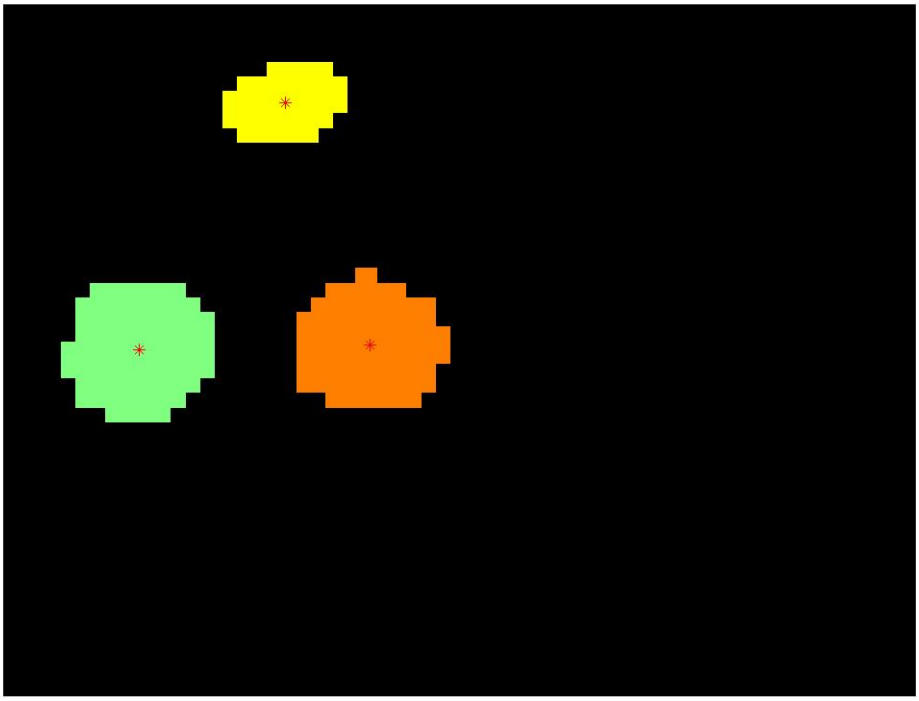


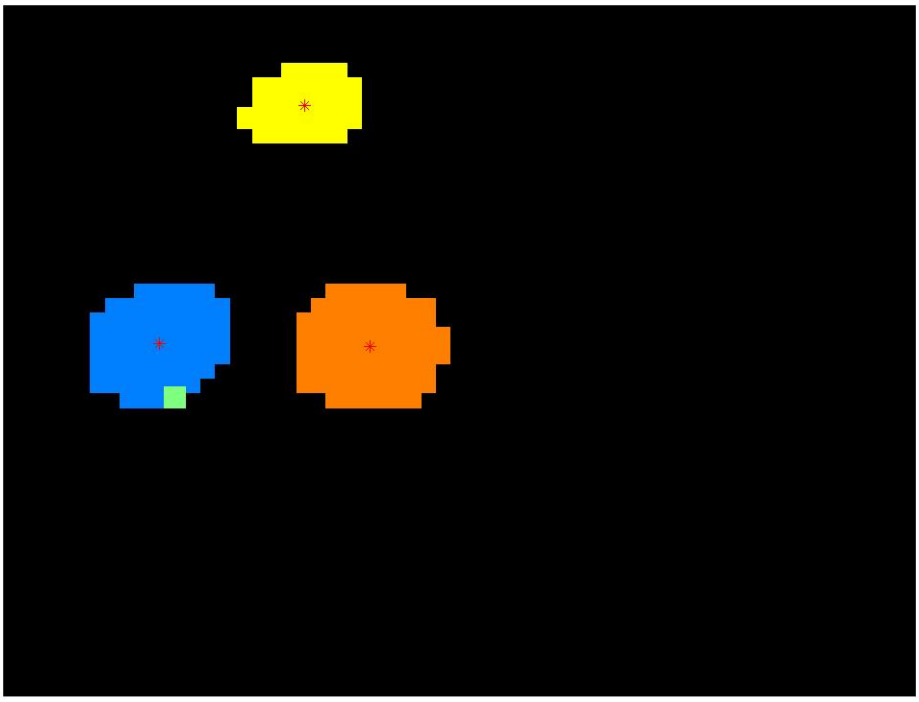


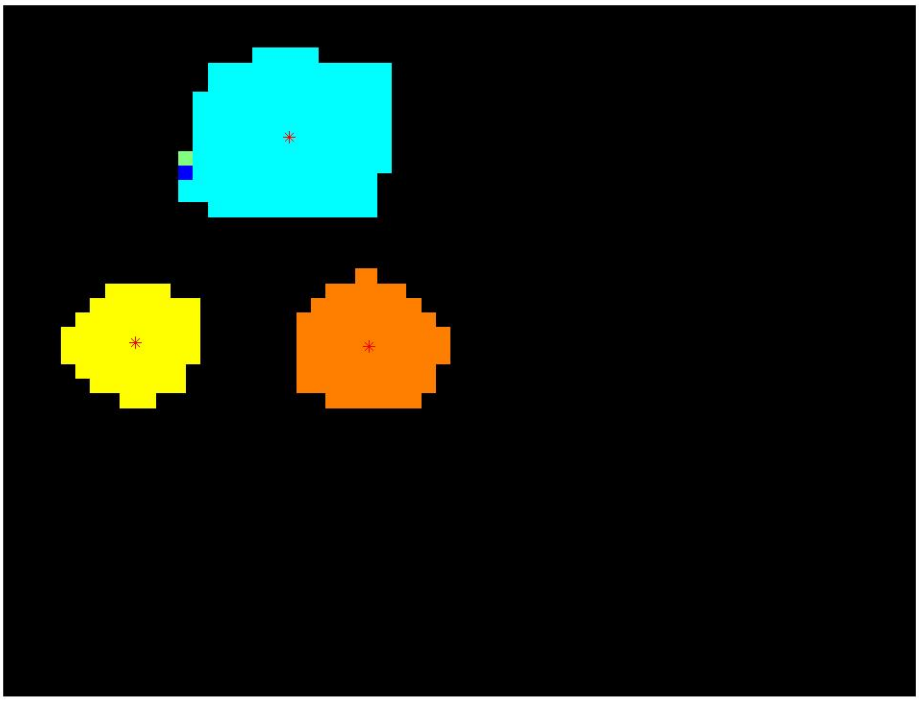


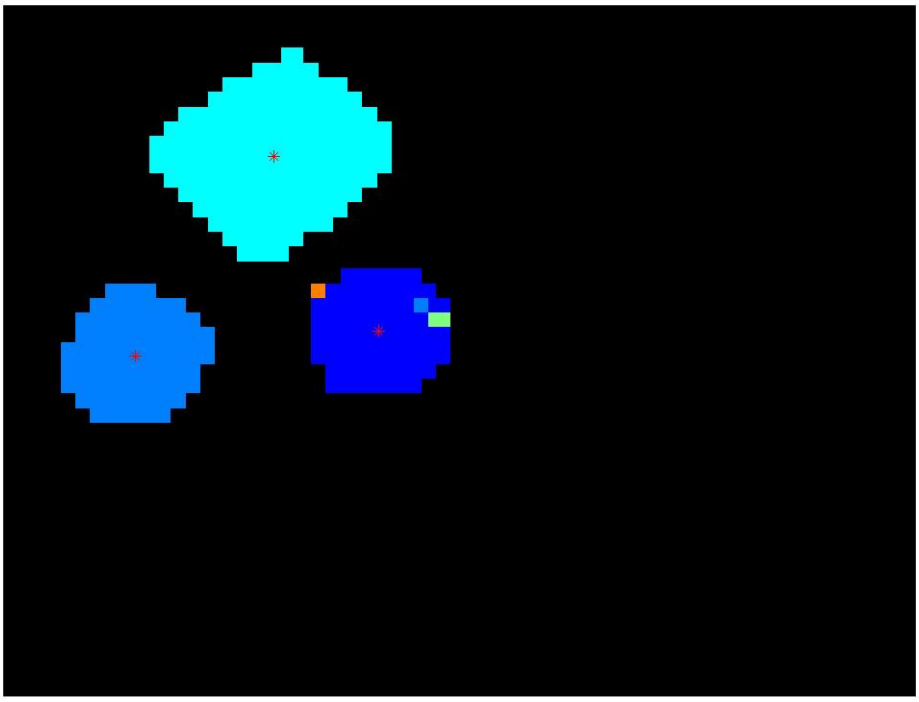


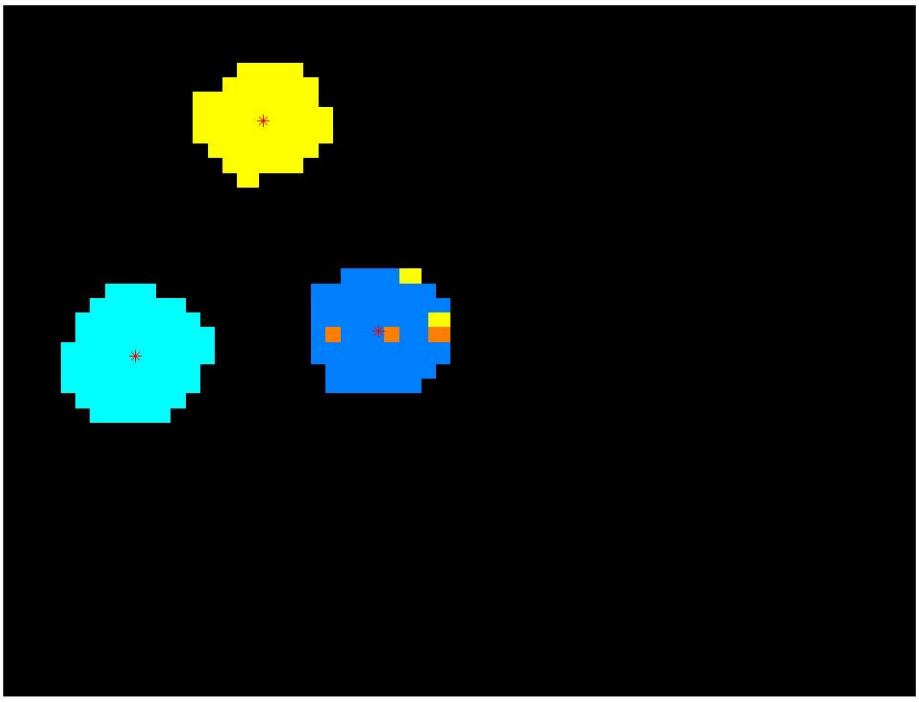
Right

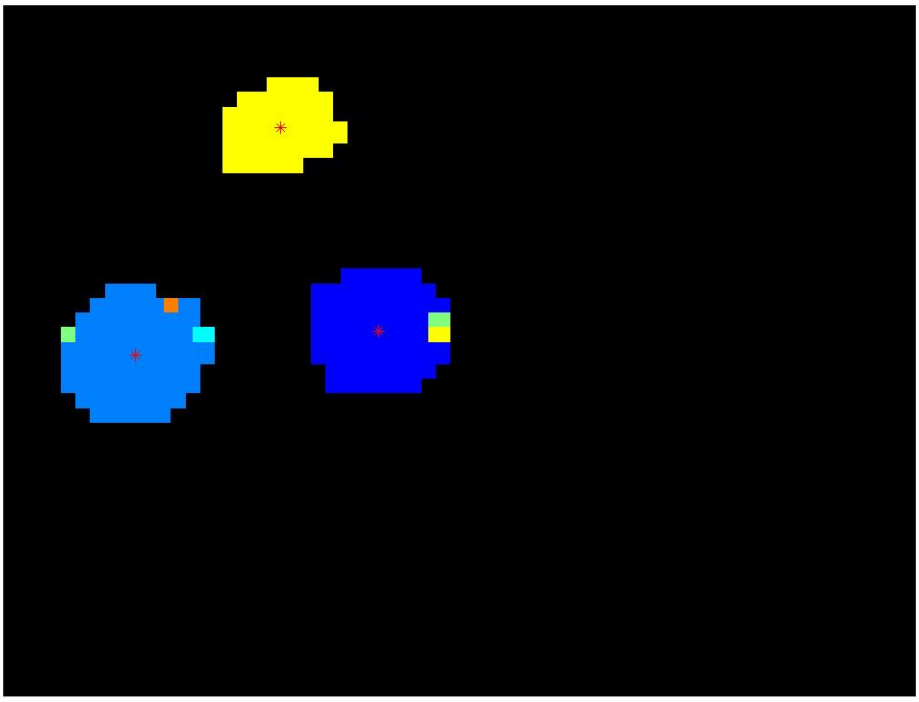


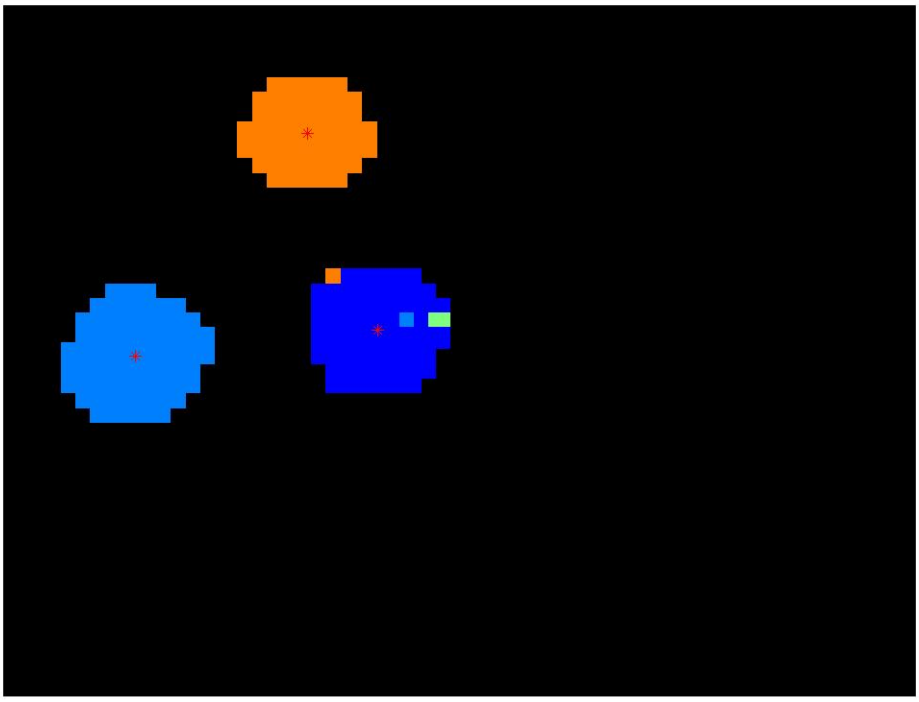


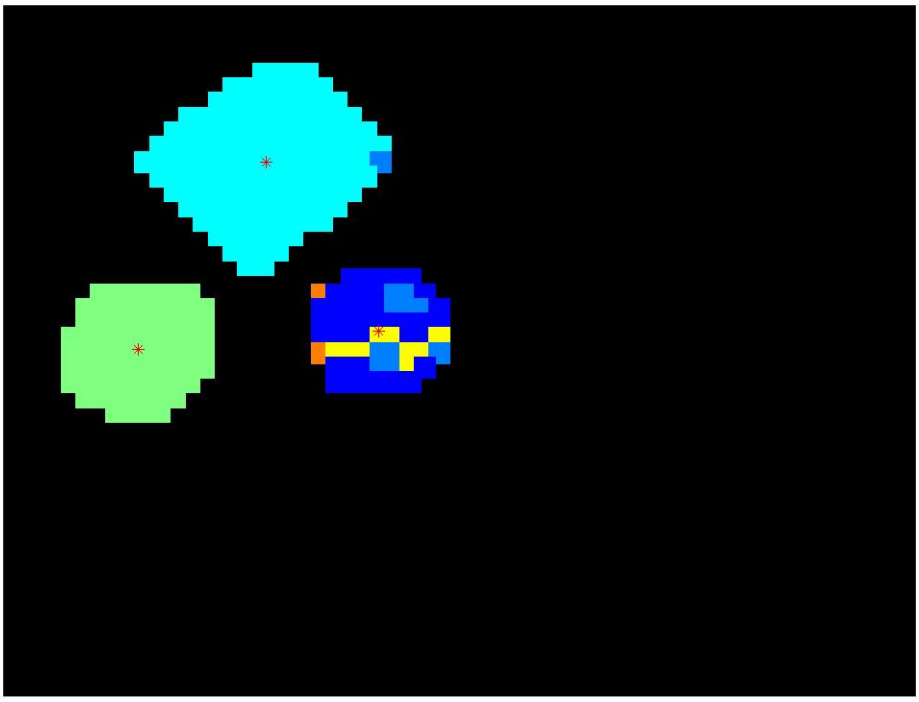


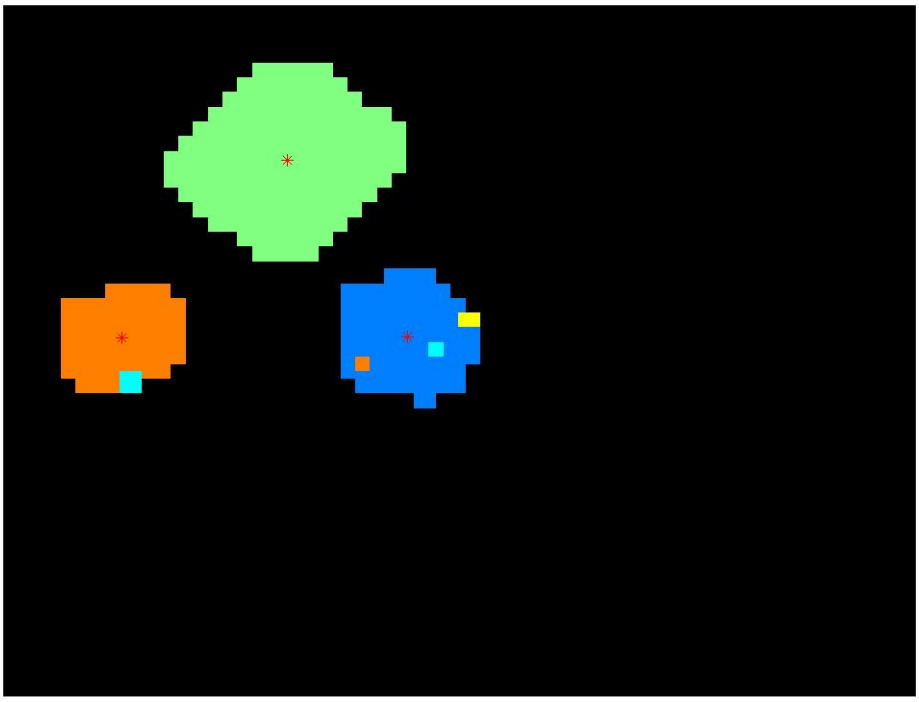


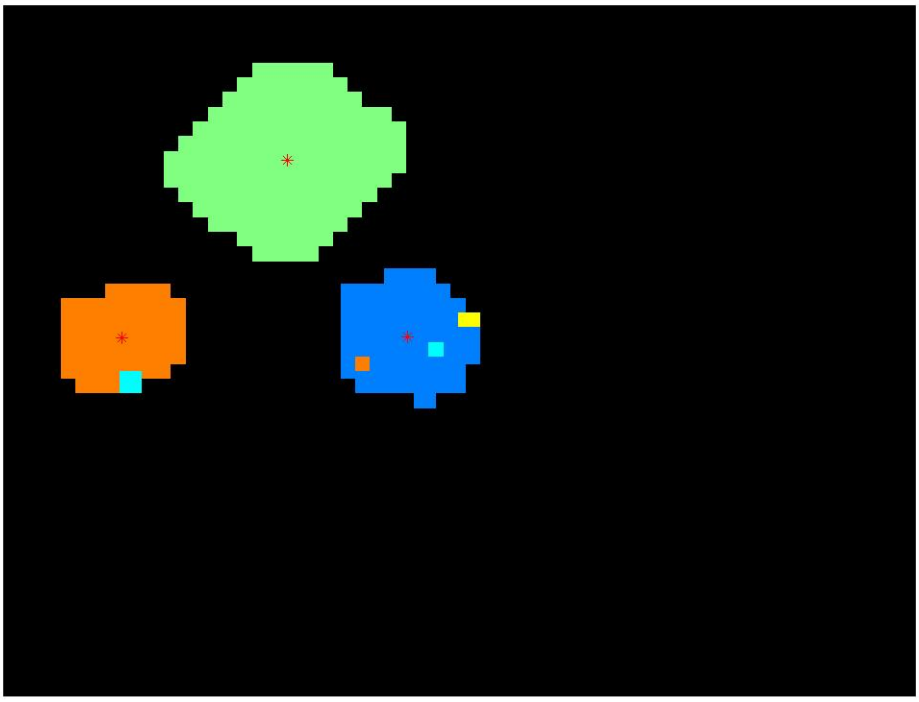


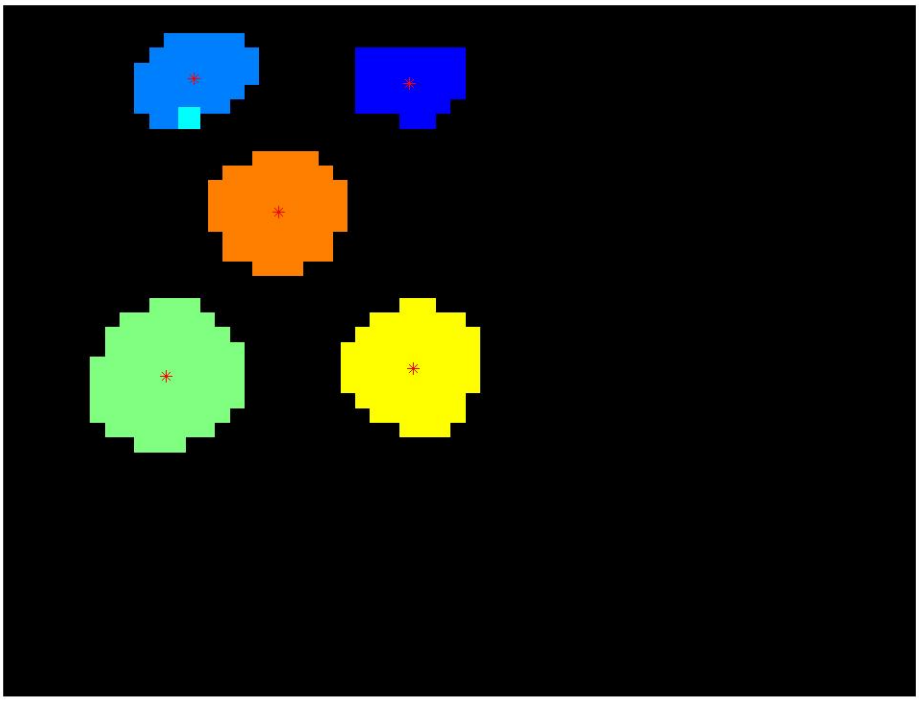


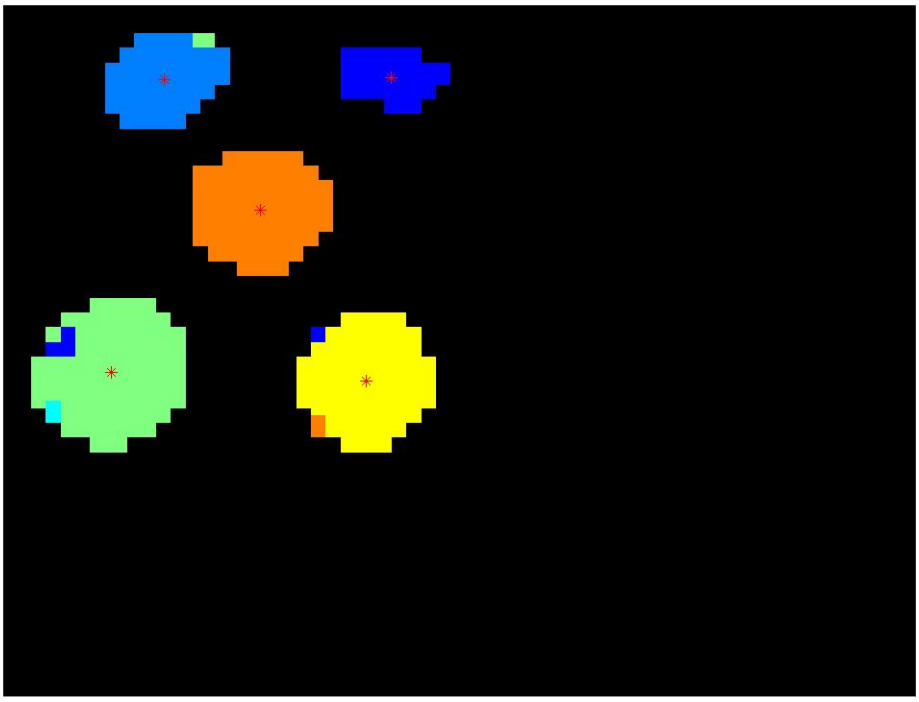


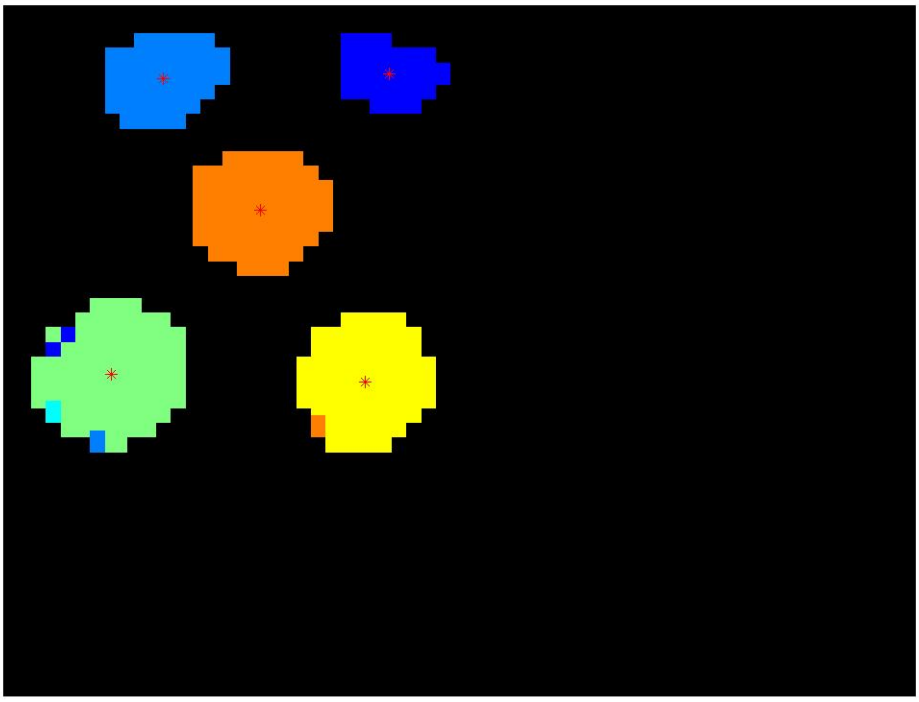


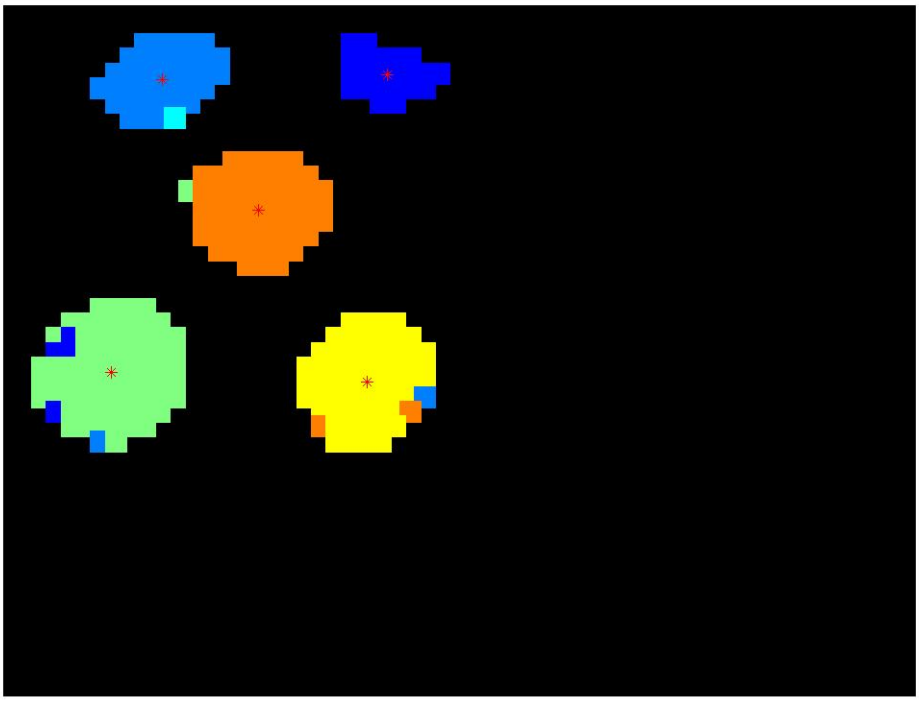






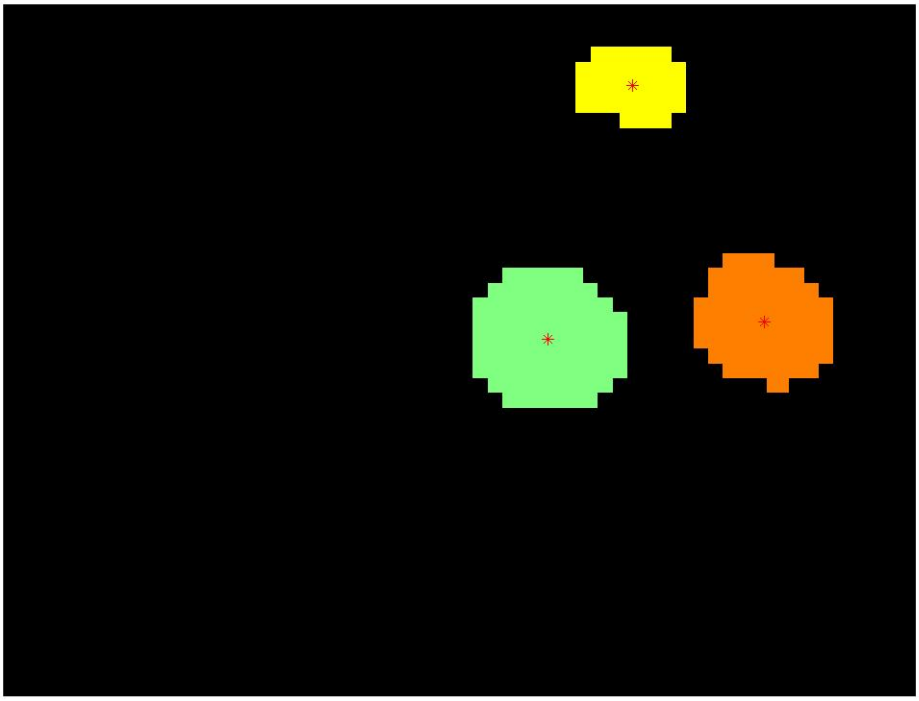


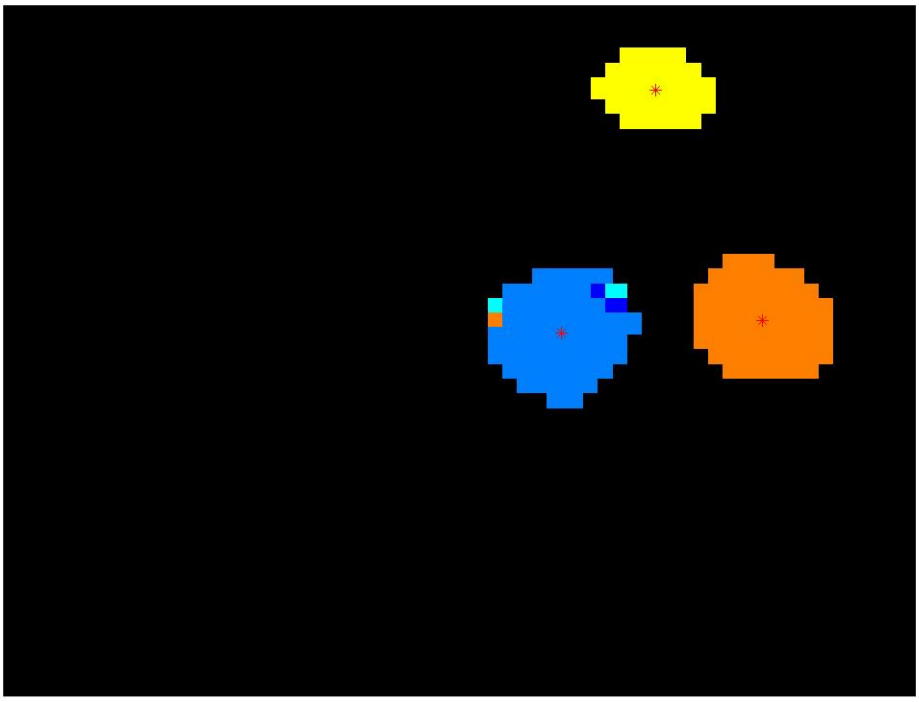


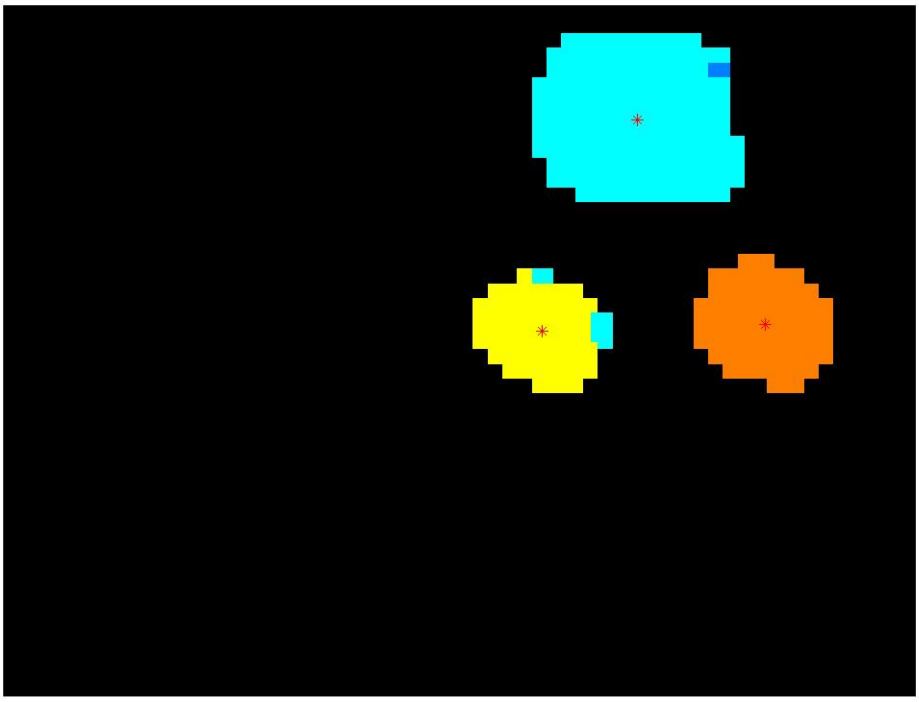


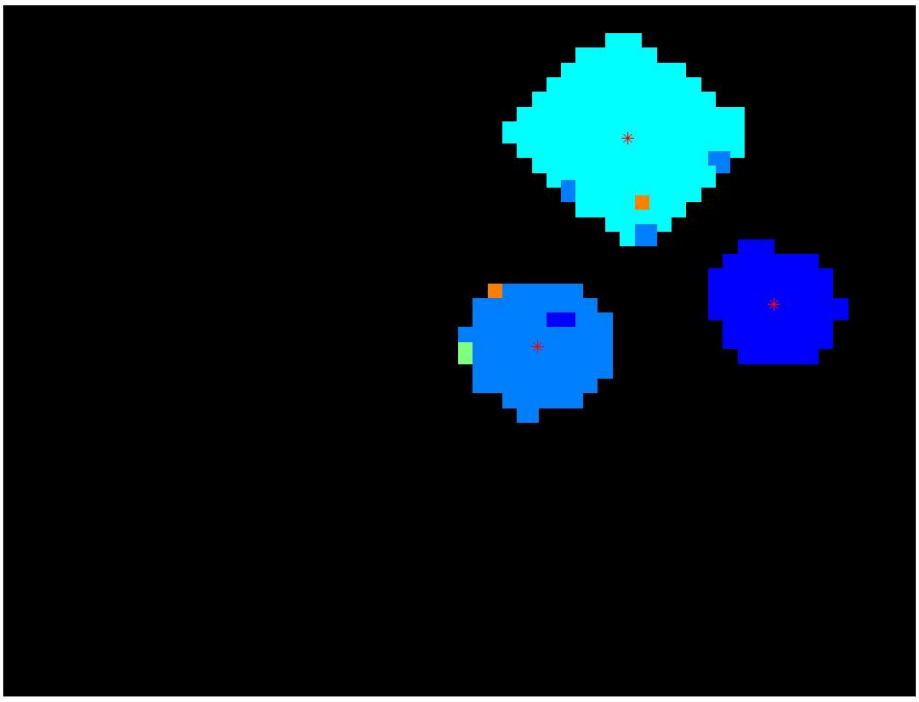
RANDOM TREE

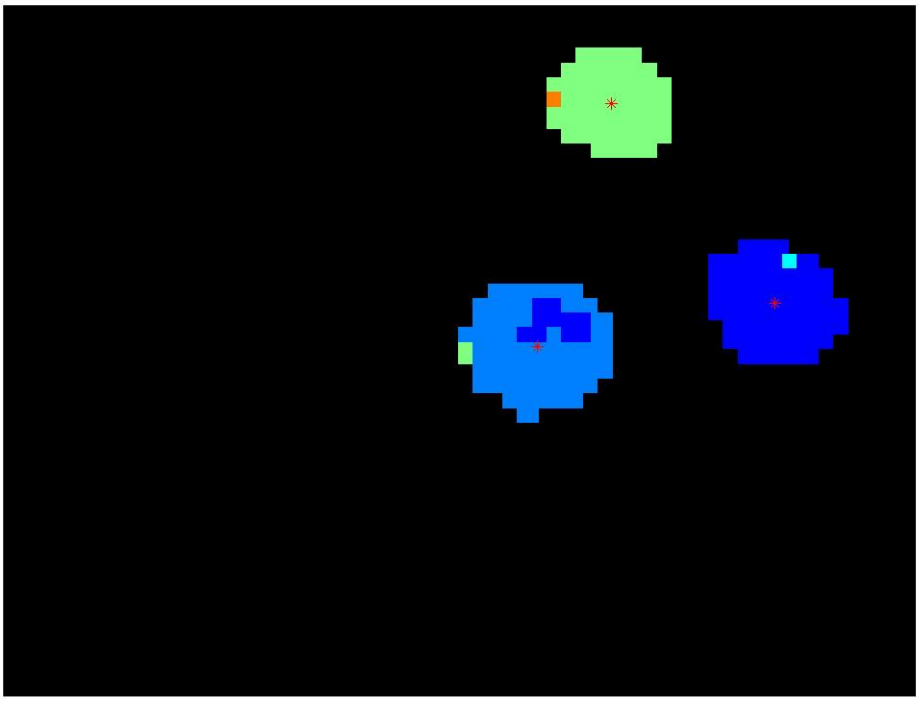
Left

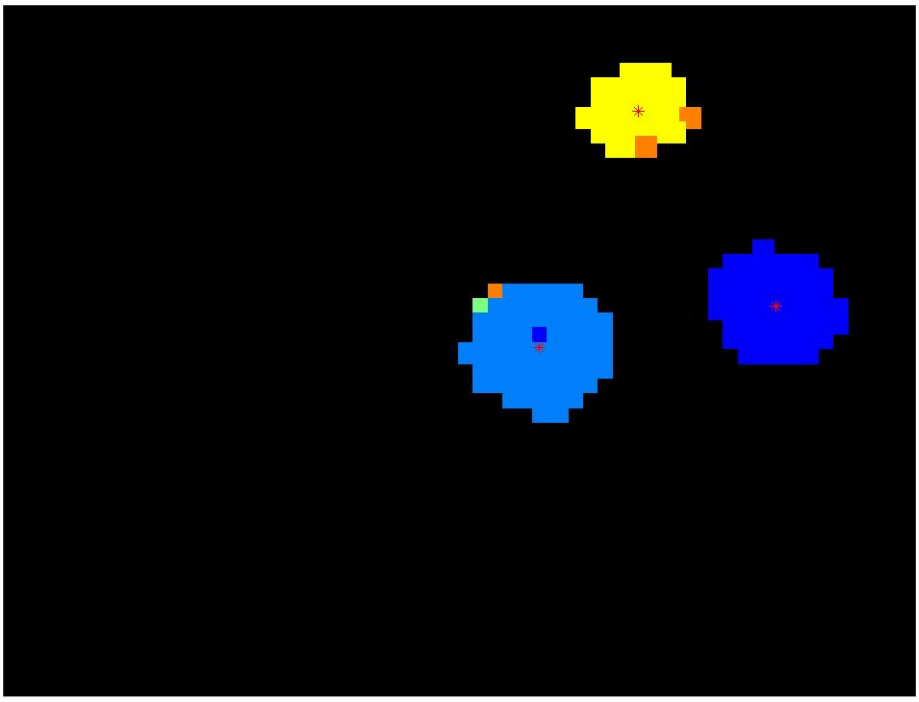


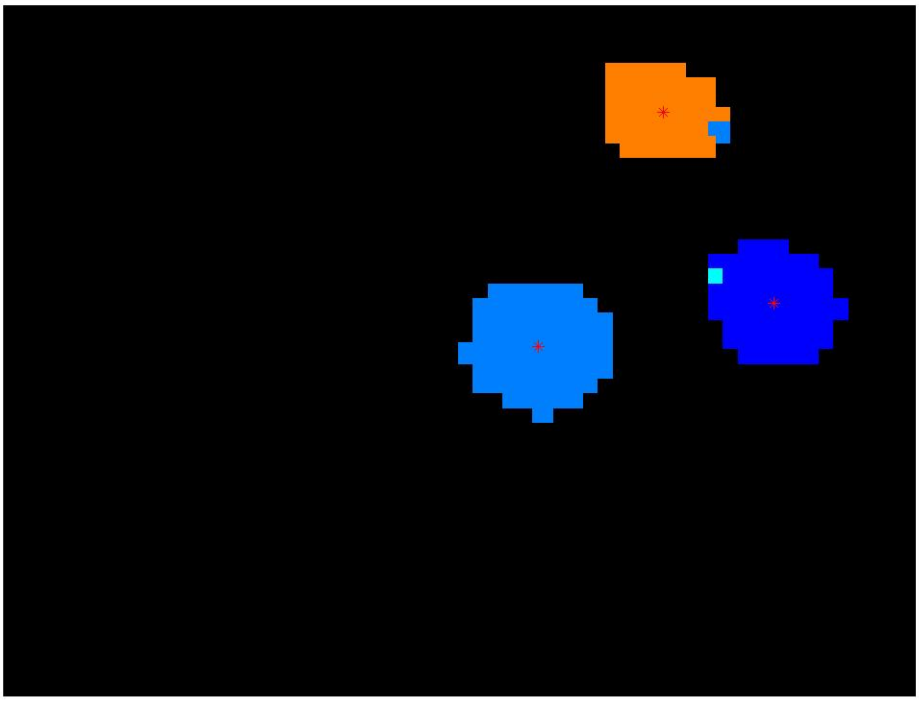


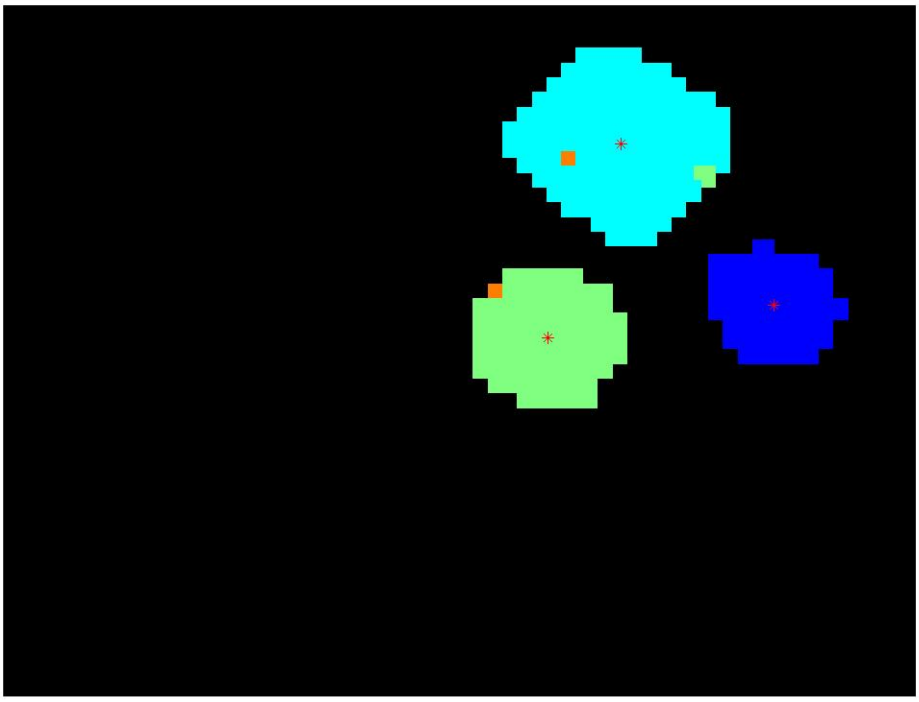


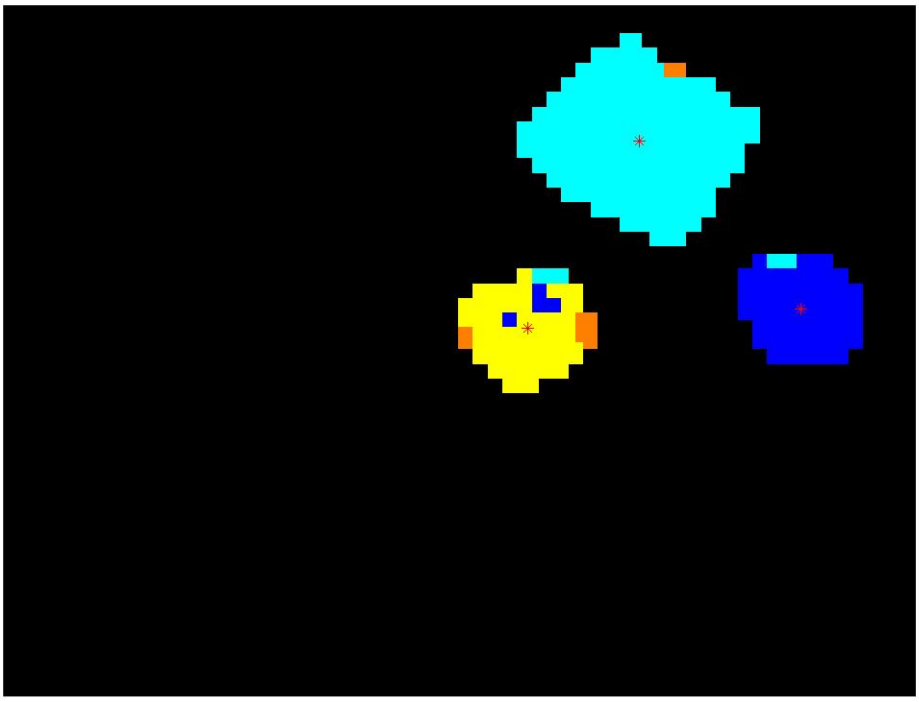


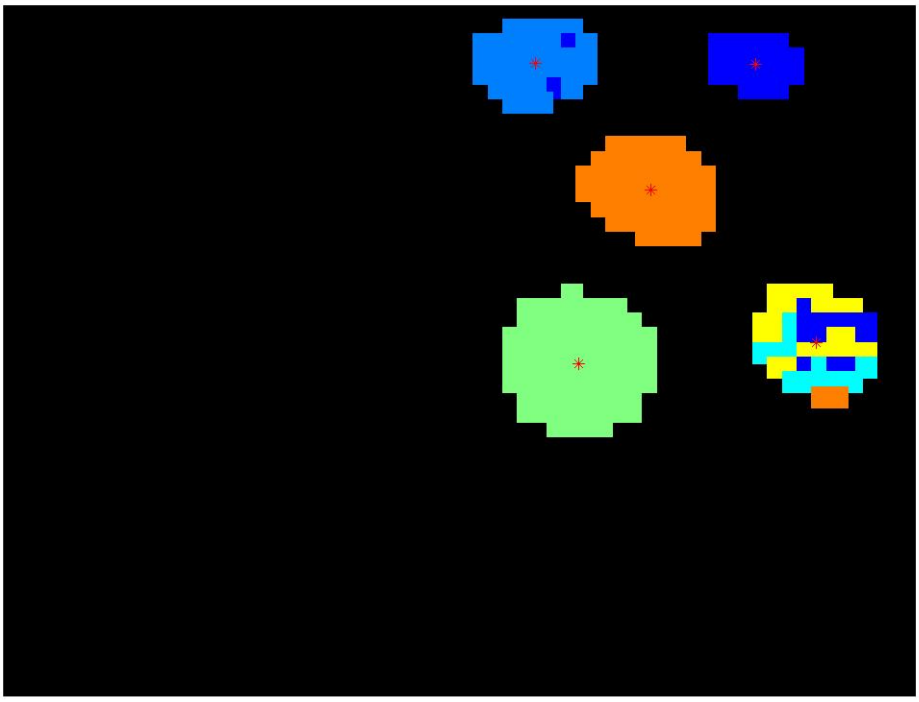


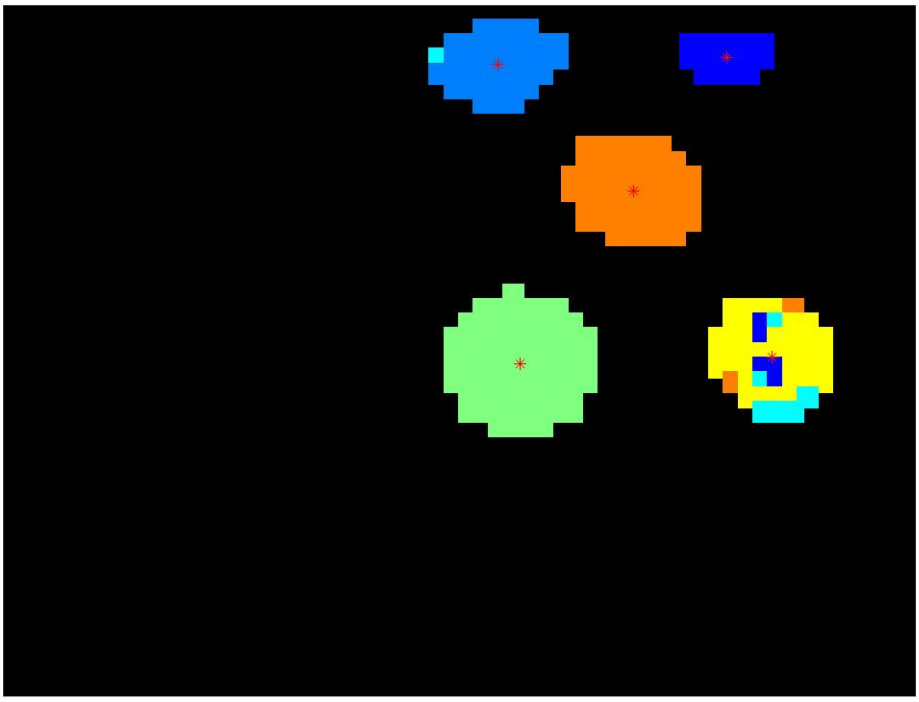


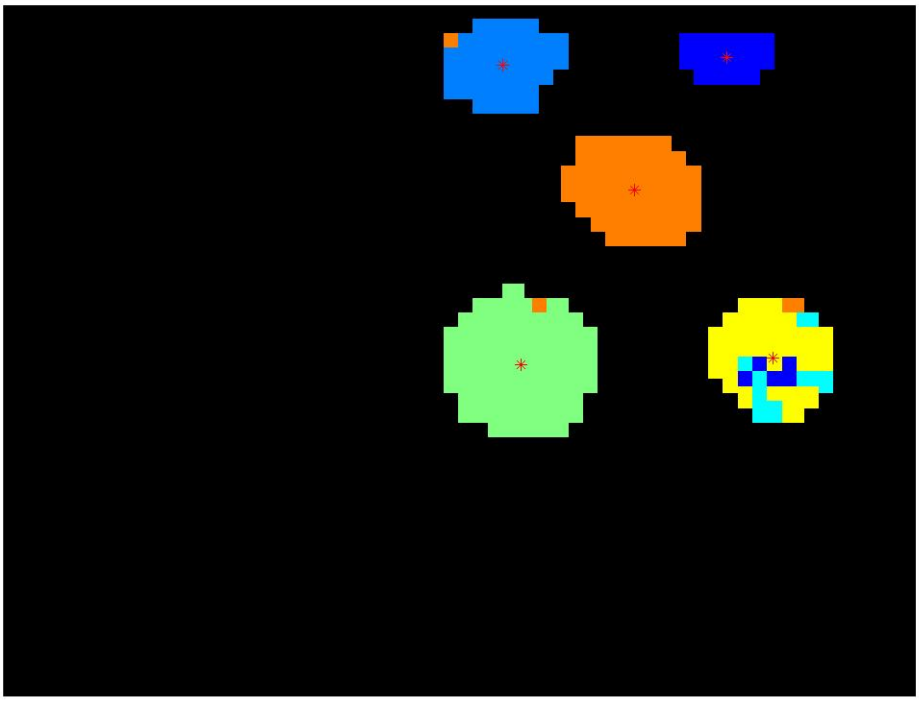


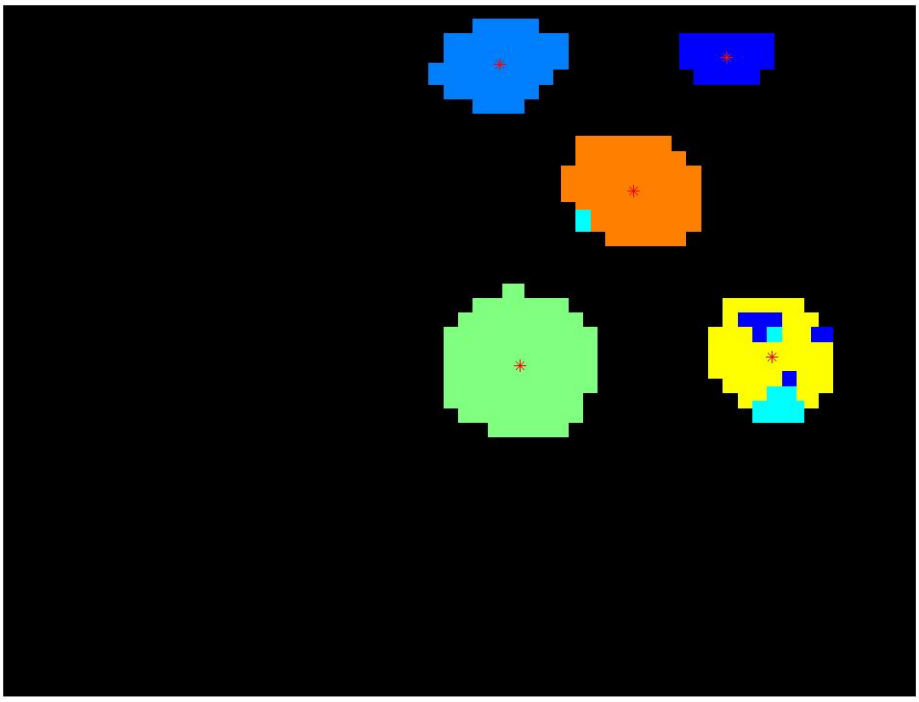












Right