
Project Number: MQP ETJ P01


STEPS TO DEVELOP A PLATFORM FOR ROBUST VISION

A Major Qualifying Project
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science
By
Samir Zutshi
4/26/2012

Approved:
Professor Eduardo Torres-Jara, Major Advisor

Abstract

The scope of this work is centered on real time and active systems. An active system is desired, as a more robust system is needed for robust vision where we can implement algorithms that are non-dependent on illumination. There are four degrees of freedom from the optical point of view, zoom, focus, light filtering and motion. Motion is addressed because it is less dependent on illumination for applications such as detecting edges. The work presented here is based on using image subtraction to detect changes in motion between two frames in black and white. When looped, this effectively presents any motion detected in the viewing window of the camera. To further apply this concept, the detection is done with the moving of the camera rather than the detecting motion of the environment. This effectively creates edges of the area in view and in addition, gives more valuable information about the subjects in the environment when compared with that of a canny rendering of the same view. There is concern with the performance of this in real time and therefore, we are using techniques such as integral imaging for clustering the edges, as regular clustering can be expensive. Currently there is a platform with one degree of freedom that is intended to be tested with the techniques mentioned above. The results are represented as visual comparisons frame by frame while the camera is given a simple motion path to detect such changes. The difficulties of real time are what are meant to be highlighted in this work and the techniques presented will attempt to address some of these difficulties



with the goal to find an alternative and efficient improvement that address these areas of vision.

Table of Contents

1. List of Figures
2. Introduction and Motivation
 - a. Active Vision and the MQP
3. Platform specifications
4. Motion Detection
5. Color Detection
 - a. Masking and Filtering
 - b. Integral Image
6. Challenges and Discussion
7. Summary



List of Figures

- 1) Object segmentation issues.
- 2) Anatomy of a poke.
- 3) Logitech C920 Camera, 1 DOF base by Nigel Cochran.
- 4) Frames 1 and 2 demonstrating the motion detection algorithm.
- 5) The difference between frames 1 and 2 as abs. value.
- 6) Test image processed using edge('sobel') algorithm in Matlab.
- 7) Test area snapshot taken during active motion detection.
- 8) Canny algorithm and Motion Algorithm snapshot.
- 9) HSV color cone.
- 10) Unmodified and Strain of Red masked still.
- 11) Inefficient clustering pseudo code.
- 12) Still image values.
- 13) The values of the pixels summed across and vertically.
- 14) Integral Image calculation formula.
- 15) The values of the pixels summed across and vertically.
- 16) Area to be evaluated through the Integral Image.
- 17) Still of points being clustered using the Integral Image.
- 18) Biomemetic Robotic Head by Nigel Cochran.

Introduction and Motivation

The motivation behind active vision comes from understanding some of the problems of still image processing and trying to improve them. Illumination and Segmentation are two areas that can always be assessed and improved in computer vision. In still image processing, an edge detection algorithm for example can yield different results based on illumination. In turn, illumination can also affect segmentation of objects detected from set still image. There are also situations where an image may be difficult to segment based on color and orientation as seen in the work of Paul Fitzpatrick below(1).

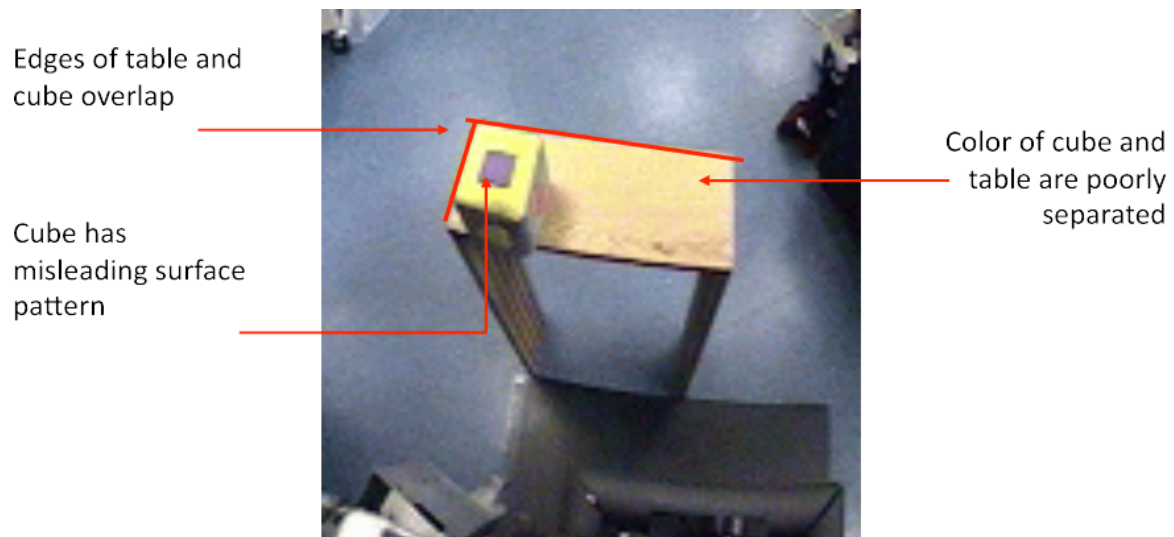


Figure 1: Object segmentation issues. Source: Fitzpatrick(1)

Where some of these issues may be difficult to segment from a still image, Paul's work with motion and edge detection was able to segment the block from the table as seen in the images below. The technique used was called a "poke".

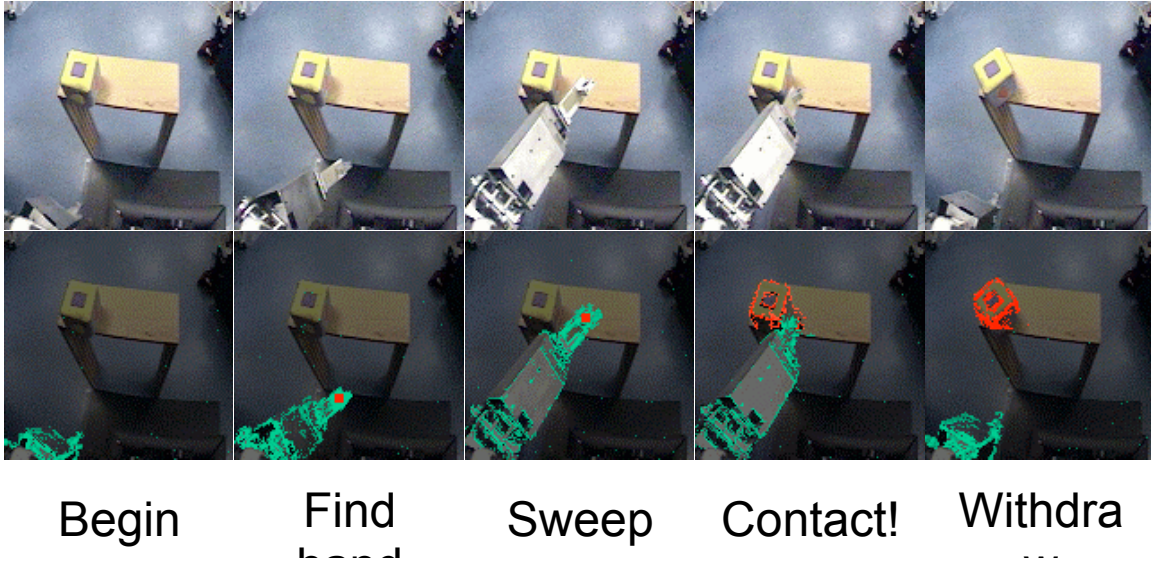


Figure 2: Anatomy of a poke. Source: Fitzpatrick (1)

This experiment as well as the motive to be active can be seen as the driving force behind the motivation of the project.

Active Vision and the MQP

There are four optical degrees of freedom in active vision; Zoom, Focus, Light Filtering, and Motion. Zoom can be used to obtain detail from an image. Focus can be used to retrieve different information from different levels of focus. Light Filtering can handle the intensity of light being used to prime and image for filtering, and finally motion can be used to detect and gain information from edge detection. This MQP dealt with motion detection actively and in real time, as well as color detection and efficient clustering through Integral Image. Real time is emphasized because

though some of the still image operations work well offline, they can be costly in real time.

Platform specifications

Programming was done on a Linux OS with Ubuntu 11.04 installed. Image capturing storing and displaying were done using OpenCV. The vision algorithms were all written in C and C++ from scratch to provide complete customization for the user and a greater sense of understanding of the algorithms for the project as they were written. The camera used was a Logitech C920 HD Pro Webcam with a custom build 1 DOF base.

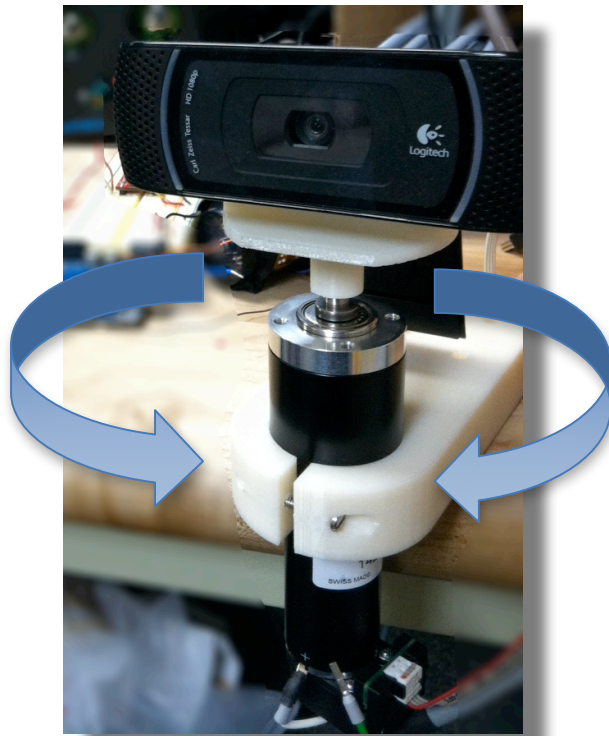


Figure 3: Logitech C920 Camera, 1 DOF base by Nigel Cochran

Motion Detection

The goal was to gather edge information actively through motion. The process involved subtracting two frames in a loop to consistently represent the changes as an absolute value displayed in white. As an example, the difference in the images below is represented as the absolute value in the image that follows the two frames.



Figure 4: Frames 1 and 2 demonstrating the motion detection algorithm.

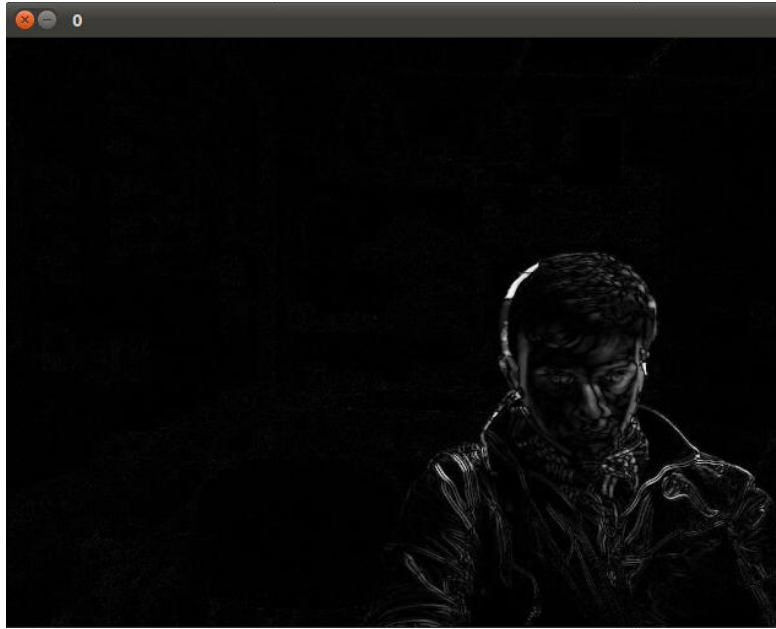


Figure 5: The difference between frames 1 and 2 as abs. value.

The standard for edge detection with still image processing has been using the canny algorithm. The goal of this experiment was to use active motion detection to present results that could be comparable to what the canny algorithm would achieve on a still image. Below are examples of both the canny and motion algorithm compared.



Figure 6: Raw test image to process through canny

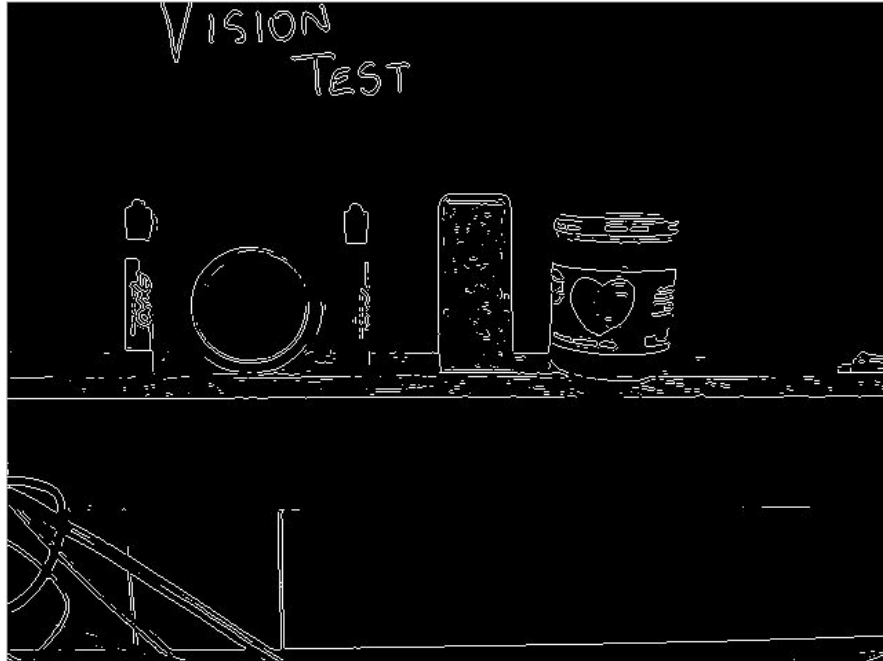


Figure 7: Test image processed using edge('sobel') algorithm in Matlab.

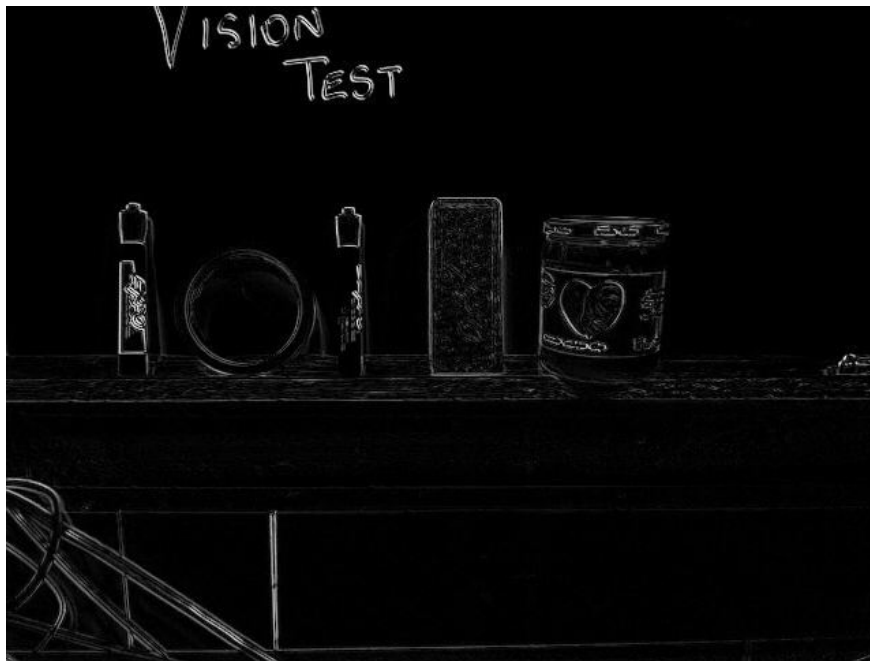


Figure 8: Test area snapshot taken during active motion detection.

The idea was to make an efficient simple algorithm that would be cost efficient for use on an embedded platform in real time. Another example of the differences can be seen below. The canny algorithm and the motion algorithms are very close in what they represent, but there are subtle facial features and lines in the background that are more complete with the motion algorithm versus the canny algorithm.

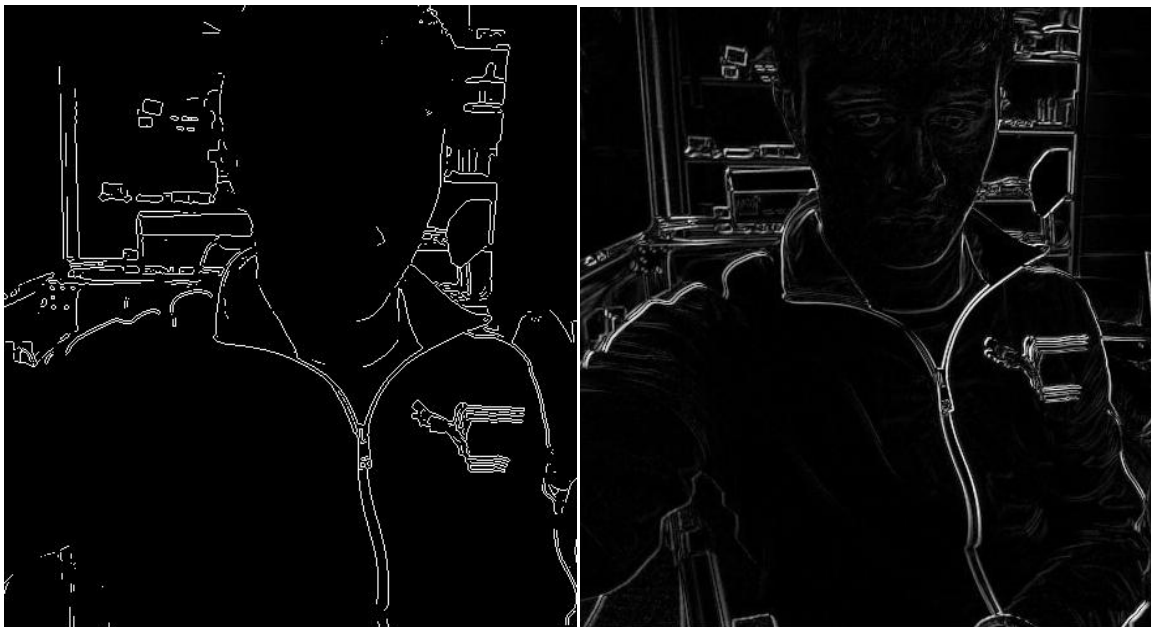


Figure 9: Canny algorithm (left), Motion Algorithm snapshot (right)

Color Detection

Masking and Filtering

The objective behind color detection was to develop and understand filtering, masking and a use for the Integral Image technique to eventually combine with motion adding another layer of information to be received from vision. The process involved first converting an image from RGB to HSV. The figure below explains the parallel of RGB and HSV.

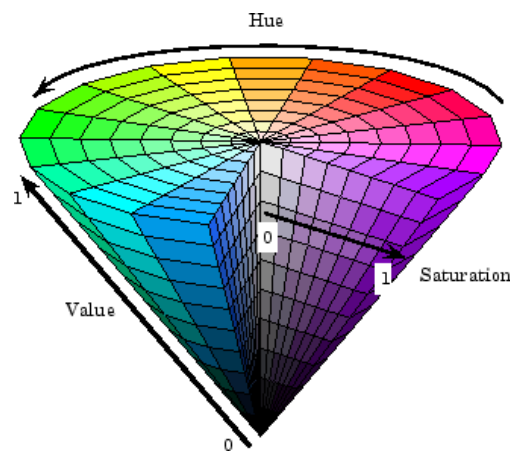


Figure 10: HSV color cone.

Once converted to HSV, a color value to be filtered was chosen as in the code snippet below, where `s.val[2]` represents H, `s.val[0]` represents V and `s.val[1]` represents S.

```
(s.val[2] > 180 && s.val[0] < 120 && s.val[1] > 102)
```

This filtered out a certain range of this color to be then masked by an algorithm that turned anything that wasn't this color, black and the filtered regions white as shown below.

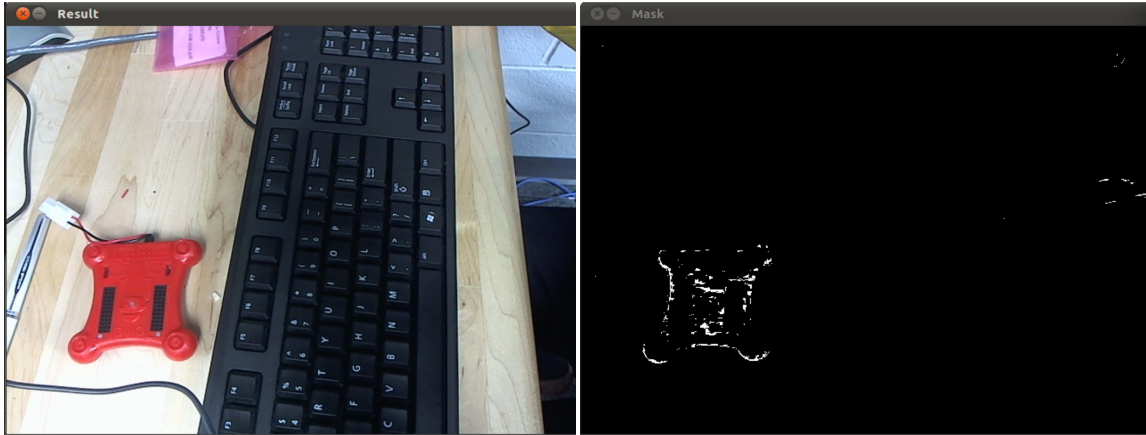


Figure 11: Unmodified still (left), Strain of Red masked still (right)

Once filtered, the next step was to use a technique to cluster the points found to add yet another layer of information to the image. To achieve this, the technique known as the Integral Image was used. To understand the necessity of its efficiency, a naïve approach was taken first. The pseudo code snipped below gives a brief description of the initial method used for clustering.

- Func 1:
 - For loop: y (from 0 -> image height -10 + 1)
 - For loop: x (from 0 -> image width -10 + 1)
 - Run helper Func 2-
- Func 2
 - For loop from 0 to 10 (y's)
 - For loop from 0 to 10 (x's)
 - get the value for H S and V
 - sumH = sumH + value of H;
 - etc for S and V}}
 - avg them here by /100
 - For loop from 0 to 10 (y's)
 - For loop from 0 to 10 (x's)
 - get the Values;
 - set the values to avg;}

Figure 12: Inefficient clustering pseudo code.

Function 1 involved two **for** loops that traversed each pixel on the image leaving 10 pixels to the right and on the bottom of the image, as the window being used was a

10x10 window. Next, for each iteration of the first function, a function 2 was run. Function 2 simply took a window of 10x10 and polled the value of H,S, and V and summed it with the previous value in the 10x10 window and then finally locally set an average value. Finally the last part of the code snipped polled the values again with the same window and set the values to the averaged values. This was just one method that slowed FPS (seen visually) and therefore seemed to be costly for processing power in real time.

Integral Image

The integral image provided a simpler way to achieve the same results more efficiently with a smaller cost. Formally, Integral Image is an algorithm used to quickly efficiently generate the sum of values in a rectangular subset of a grid (Source:Wiki). Below is an image that represents a 4x4 window of pixel values.

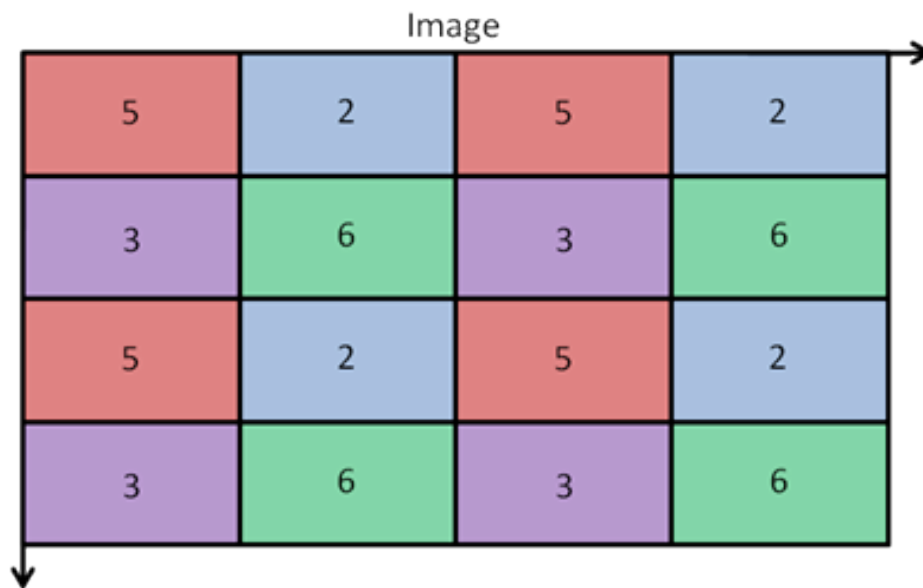


Figure 13: Still image values. Source(3).

Summed Area Table

5	7	12	14
8	16	24	32
13	23	36	46
16	32	48	64

Figure 14: The values of the pixels summed across and vertically. Source(3).

Each row is summed across and then each column vertically. Once this process is complete on the entire viewing space, the area of any region can be easily computed by the equation below.

$$i(x',y') = s(A) + s(D) - s(B) - s(C)$$

Figure 15: Integral Image calculation formula. Source(3).

The variables A, B, C and D represent the pixels to the upper left of the top left corner, the pixel above the upper right corner, the pixel to the left of the bottom left corner, and the pixel in the bottom right corner of the area desired respectively. The image below shows an example area being calculated using the formula above.

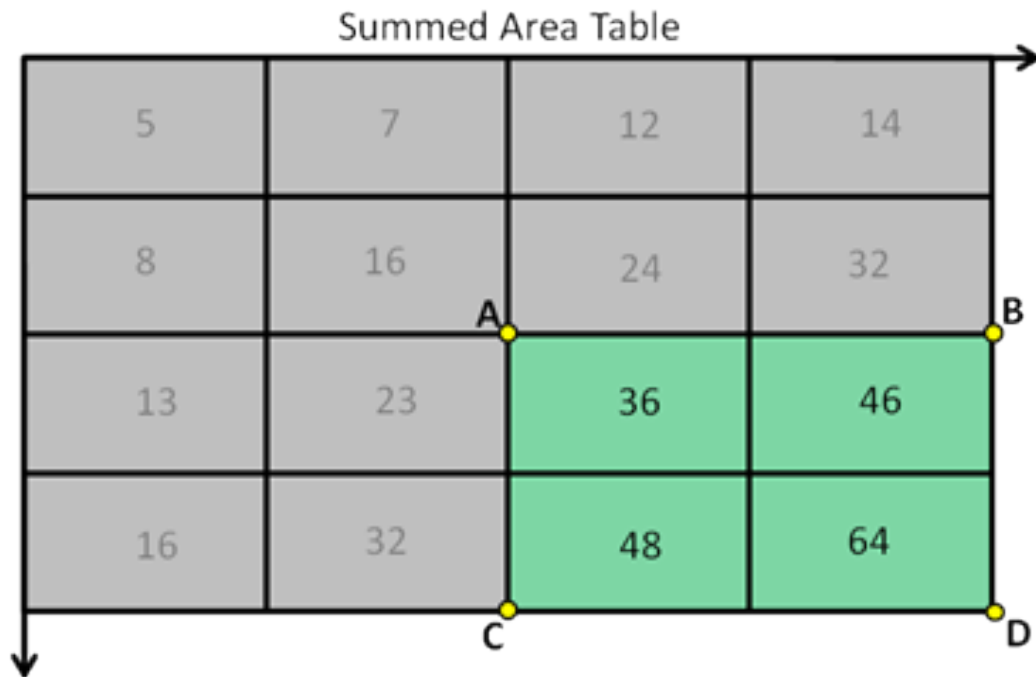


Figure 16: Area to be evaluated through the Integral Image. Source(3).

By the equation, the values to compute the area are as follow, $16+64-32-32 = 16$.

Therefore once a summed area table is computed, no more summing is required or processing of the entire image and thus, the simple area calculation can be done to quickly compute the area desired to update the image. When performing clustering, this process can provide efficiency as well as ease when averaging pixels together in real time.

The masked still from filtering above is represented below as a still taken while applying this algorithm after masking.

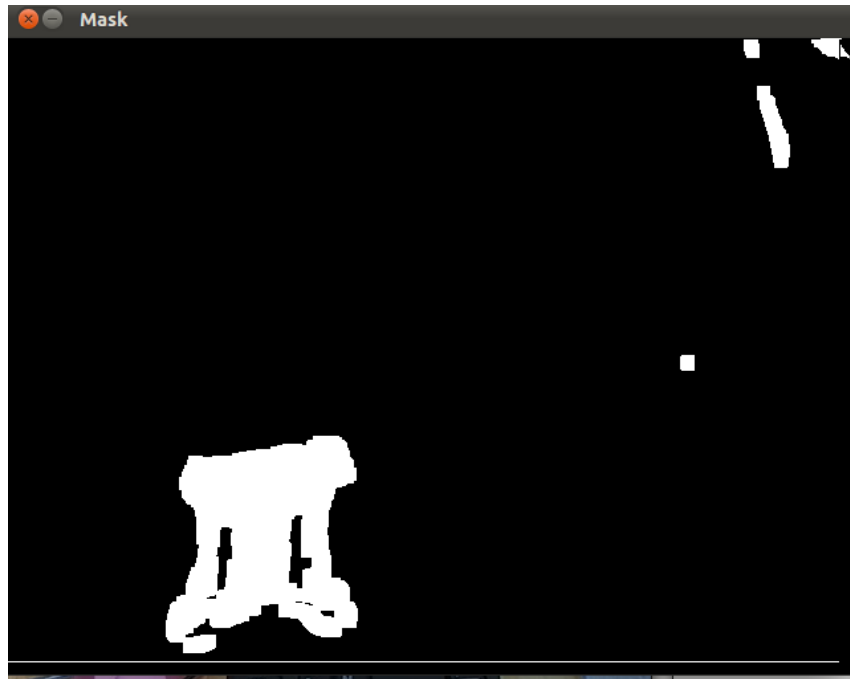


Figure 17: Still of points being clustered using the Integral Image.



Challenges and Discussion

Challenges during this project varied from dealing with just the Linux OS to working with the many different integration platforms using this OS. The goal of this project was not only to develop these vision algorithms for efficiency and robustness but also to give the student a learning experience with the Linux OS as well as with programming in the environment. Unfortunately the version of Linux that was chosen was Ubuntu 11.04 that ended up having many dependency issues along the way. Initially, these issues were hard to fix as when one dependency issue was fixed, yet another had a problem. Down the line even the Ubuntu software manager stopped working and posed an uncanny problem to install different software's/libraries that were pertinent to advancing in this project.

Initially the goal was to work with V4L (Video for Linux) and its libraries to program some of this code, but it seemed that the camera's being used were not supported by V4L and if they were, there were other compatibility issues that were unable to be explained. The web offered different angles of information to solve certain problems, but without a lot of other resource to go to, a lot of these issues took a decent amount of time to overcome. If they weren't overcome, the strategy had to be redrawn and the task moved in another direction.

As the project moved along, Professor Paul Fitzpatrick was introduced and helped shed light on some of the open CV issues. Once these issues were addressed, and the compiling flags were found, the algorithms began to take shape. While some

of the techniques had CV libraries already, the algorithms were written without them but using them as a reference for comparison. The goal of writing every piece of code was to fully understand the functionality of the process and the code so that its results could be equally understood.

Summary

In summary, the project was an experience in understanding a new OS and developing in its environment, as well as working on steps to achieve robust vision. It spanned from working with motion on a still setting to working with motion actively, and thus drew areas for comparison to the techniques used in still image processing today. The overall objective was to learn as much as possible about the vast area of computer vision and work on adding as many layers of work to gain as much information as possible. The problem of computer vision is not something to be solved but to be improved upon as much as possible.

Once motion detection was established, techniques to filter, mask, cluster and detect color were put in place. These techniques were stepping stones to be combined in future experiments alongside the motion algorithms to add yet another layer of detail to be examined in computer vision. Currently the project strives to compute qualitatively and quantitatively the consistencies in the techniques presented here using encoders and serial communication. The future can be seen in embedded applications as well as in instances where all four optical degrees of freedom can be applied as done in the robotic head below.



Figure18: Biomemetic Robotic Head by Nigel Cochran.

References

- [1] Paul Fitzpatrick, "**Better Vision through Poking**", to Leg Lab group at MIT AI Lab, June 2002, <http://people.csail.mit.edu/paulfitz/presentations.shtml>

- [2] Wikipedia, Integral Image, www.wikipedia.org

- [3] Badgerati, Computer Vision – The Integral Image,
<http://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/>

- [4] M.Jones & P.Viola, Robust Real-time Object Detection,
http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_IJCV.pdf

- [5] Mathworks Documentation,
<http://www.mathworks.com/help/toolbox/images/f8-20792.html>