Expected Outline:

- Abstract
- Introduction
- Background
- Method
- Results
- Discussion
- Acknowledgements
- Bibliography

Report On Summer Research

Carlos Andres Berejnoi Bejarano
Advisor:
Dr. Matt Jadud
Berea College

1. ABSTRACT

During a summer research for the Computer Science department at Berea College, Berea, KY, USA, a traffic counting system was implemented using computer vision techniques. The purpose of this system is to be able to count the number of vehicles that transit a road using only one stationary camera and an affordable computer. As such, the system must be able to run with limited resources while being reliable for this task. The target low level computer is a Raspberry Pi B with Raspbian.

The tool chosen for the project is OpenCV, which is a library of computer vision functions and algorithms implemented in optimized C++, with Python bindings. The C++ code provides the efficiency needed to do fast image processing, while the Python bindings allow for quick development without sacrificing speed of execution.

After two months of work, the counter is able to count moving vehicles with reasonable accuracy under specific conditions. More work and development is needed for it to be complete and ready for actual applications.

Keywords: Computer Vision, OpenCV, Python, Image processing, Raspberry Pi, traffic counting, Berea College, URCPP, Computer Science Department, CSC Berea College, CSC

2. INTRODUCTION

There are many problems that could benefit from the aid of computers. This project consisted in applying computer vision techniques to develop a traffic counting system that can run on an embedded device with limited computational power. The advantages of computer vision over other techniques is the range of information that we can get out of it if it is implemented correctly. Additionally, installing a computer-based counter into a road would not

be invasive or detracting to the traffic, whereas other methods, such as inductive loops can be inconvenient to put in place. After two months of work, a non-model based traffic counter was implemented in Python, using OpenCV to provide the necessary libraries to make the system work. Following is a description of how the program was designed and the results it is able to achieve.

3. BACKGROUND

BACKGROUND SUBTRACTION AND RUNNING AVERAGE

Background subtraction is a method to detect movement based on the difference between the current frame of the video, and the empty background. This method requires that an image of the empty background is obtained before, which is not always available.
With an empty background image, when a new frame appears, the computer can recognize if there has been any change. By taking the current frame and subtracting the background from it, pixel by pixel, the result will be a mask showing the changes. If the intensity of this mask surpasses a preset threshold, then it is considered as movement [1]. This method also assumes that the camera is static, so that the only thing moving are the actual objects in movement.

One complication with this technique is the difficulty in obtaining an image of the background when there are no objects in the scene. Another problem is that background subtraction by itself is vulnerable to changes in the environment. For instance, as the day progresses, the lighting changes but the image of the empty background used as a reference is not being updated. As a result, every new frame will potentially be different enough that even if no object is moving, movement is detected. This means that background subtraction works better for static cameras, in places where the lighting and background are constant, such as inside a building or other artificially illuminated area.

To deal with the problem of a constantly changing background, a different method was implemented: Running Average of the Background[2]. This technique improves on background subtraction because an original empty background is not needed to make it work. Instead, every new frame is used to create a running average. The result is a mask that is more resistant to changes in the environment. Newer frames will change the average, and as a result, the background will be constantly adapting. For instance, the transition from day to night would pose no problem because the system's background is already updated. However, this algorithm can have problems in situations when an object stays in the background for a sufficiently long time and then leaves the scene. This will cause the object to be incorporated into the background and will take longer to disappear from it once it is gone. An example of this could be a car parking in front of the camera for a few minutes and then leaving. This can be a source of confusion for the counter system.

4. METHOD

1 Mahesh C. Pawaskar, N. S. Narkhede, Saurabh S. Athalye, "Detection of Moving Object Based on Background Subtraction," pp. 215
2 S. Susrutha Babu, S. Suparshya Babu, Dr. Habibulla Khan, M. Kalpana Chowdary, "Implementation of Running Average Background Subtraction Algorithm in FPGA for Image Processing Applications",

The project created is a traffic counting system. It should be a system that can perform relatively well on a low-level computer such as a Raspberry Pi. Before explaining the program, the specifications of the computer are listed below:

Rapberry Pi B:

- Single-core ARM processor @ 700MHz
- 512 MB of RAM
- OS: Raspbian

Laptop:

- core-i5 @ 2.50GHz Intel processor
- 8 GB of RAM
- OS: Ubuntu Gnome 15.04 64-bit

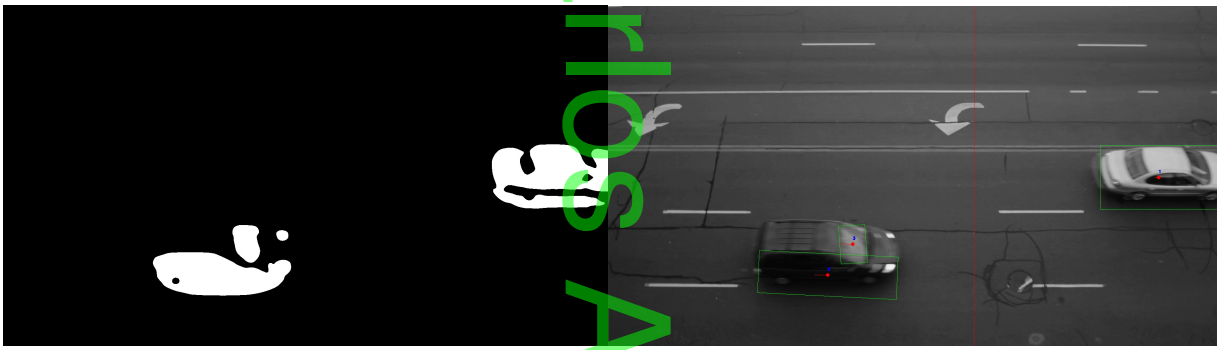
The program was designed to be able to run on the Raspberry Pi, however, a laptop was used to test and develop the code faster. After setting up Raspian on the Pi, a tool for working with images was necessary. A quick search online showed that OpenCV is a widely used library for computer vision applications, and was selected as the tool for the project.

OpenCV stands for “Open Source Computer Vision” library. It contains a large collection of computer vision algorithms. It is very convenient that it is written in C++ but still available on other languages, such as Python, through the implemented bindings. The C++ code is optimized to perform fast enough for real time applications, something crucial in many computer vision projects. Furthermore, the Python bindings provide a high level API that is not hard to use. Anyone with some knowledge of Python, or some of the other languages supported by OpenCV can start developing his/her own applications and test results without having to understand the underlying concepts thoroughly. The version used for the project is OpenCV 2.4.9.

4.1 INITIALIZING THE PROGRAM

There are some important steps that must be taken before the image processing takes place. Some of these steps are hard to complete correctly. Moreover, failure in carrying them on correctly can significantly hurt the system’s performance. This is due to the short period available for development. The main system was developed with as much thought as possible. However, the initialization steps are not at a stage where someone not familiar with the system could just run it.

For starters, when running the program for the first time, the user must indicate a video source - which could be a webcam or a pre-recorded video file - to use, then a minimum area size must be specified. The area size indicates what is the minimum area that a contour or blob must have in order for it to be recognized and tracked by a bounding box. A contour is a shape that the program recognizes as a single object; it is represented with white pixels. If a contour does not meet the criteria, it will be ignored. The negative side is that setting the minimum area size is at the moment a process of trial and error. A small number will result in many minor contours being counted, even if they belong to the same object:



Left: The car's windshield appears in the image separate from the main body.
Right: The program recognizes the windshield as another object and assigns a bounding box to it. When this car crosses the red line, the counter will increase by two, damaging its accuracy.

The image shows that the windshield's area surpassed the minimum set. However, increasing the lower limit can also affect performance by causing the program to miss some vehicles that are difficult to notice. For instance, motorcycles and dark vehicles usually present a relatively small area compared to the rest of cars, so setting up a higher minimum can cause the program to not see them.

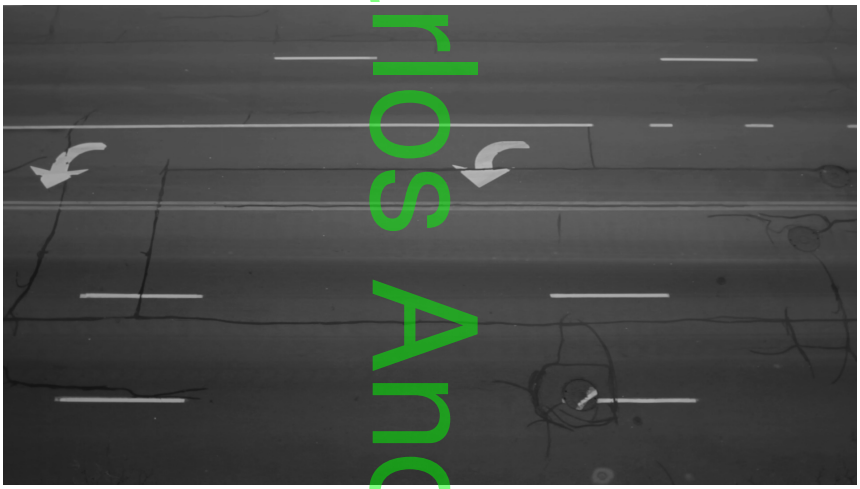
The next step to initialize the program is to select the orientation of the counting line. Currently, the line can only be placed vertically or horizontally. The user has, however, control over where the line goes along the axis chosen. Trial and error experiments showed that places close to, but not exactly at, the center of the frame improve accuracy.

Finally, when the initial parameters are set, the user needs to select a Region of Interest (ROI). The ROI is an area of the frame that the user selects so that anything outside of it will be disregarded. The main advantage of this step is that it can greatly improve the speed of the program by reducing the matrix size (a frame will be stored as a matrix or array of pixels), therefore allowing more frames to be processed each second.

4.2 FRAME PROCESSING

A number of steps must be completed on each frame of the video source in order for the program to be able to recognize movement on the road. The system implemented here is a non-model based traffic counter. This means that the program does not perform object recognition on the vehicles, instead it depends on changes in the background image with respect to the current frame. Any moving object with sufficient size can cause the counter to increase. As a result, the system is highly vulnerable to confusions and mistakes. On the other hand, it does not require too much computational power to run.

The first step is to read the frame and convert it to grayscale in order to start a running average of the background. The conversion to grayscale reduces computational costs by creating a new image with only one color channel since the others are not necessary for the running average. The running average is obtained through OpenCV's function *cv2.accumulateWeighted().*



This seems to be an image of an empty background. However, it is the average of the background at the moment of the snapshot. Cars are moving on the road, but their short presence has a small effect on the average.

Once the running average is set up, a new frame from the video source is read and an absolute difference is obtained. The function cv2.absdiff() takes the background frame (running average) and the current frame and returns a new image consisting only of the pixels that changed. However, as background is constantly updating itself with each new frame, some false positives could affect the counter. In order to remediate the problem, a threshold needs to be applied to the absolute difference of the frame and the background. The prefered method is to use a binary threshold, so that there is a clear difference between what is moving and what is static.

Left: The absolute difference mask from the frame to the background.
Right: The motion mask obtained after applying a binary threshold to the difference.

The binary image above has also been processed to eliminate noises such as small white pixels that do not belong to the moving cars, or black spots inside the white blobs. These results were achieved using three OpenCV functions: *cv2.blur(), cv2.erode(), and cv2.dilate().* The result is a cleaner binary image that facilitates finding the contours of vehicles in movement.

The next step in the process is to detect all the contours of cars and surround them with a bounding box. This has a number of benefits: the human user will get a clear visual signal of what the computer is detecting as movement, helping with improvement or debugging

procedures; on the other hand, the bounding boxes provide an easy way to calculate the blobs' centroids and be able to count them. The centroid is simply the coordinate position (x,y) for each bounding box, relative to the pixels in the frame.

4.3 COUNTING SYSTEM

Being able to track the moving vehicles on the road is made relatively easy with OpenCV functions. However, being able to accurately count those moving objects can present some challenges. For instance, one must decide how the counter should be increased. For this project, two points on the frame must be selected and a counter will be increased whenever an object crosses the line segment created by those points. The current implementation only allows the line to be horizontal or vertical. When a single car is in the frame, the counting is very straightforward. For this, on each consecutive frame, the (x,y) position of the centroid in the contour that is being tracked is stored. Then, on each new frame, the program evaluates if the counting line lies in between the previous centroid's location and the current one; if it does, the counter is increased.

The counting line is on the lower half of the image. Some detected contours can also be seen, as well as their respective centroids.

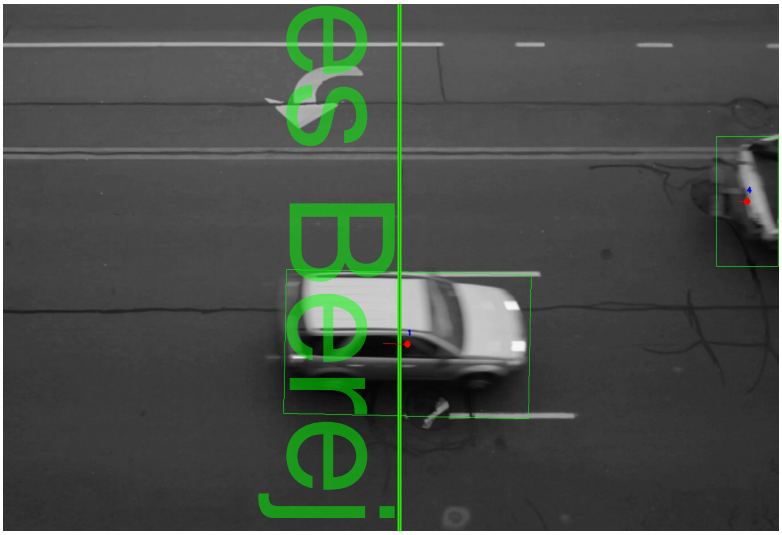
A complication arises when multiple cars are in the frame at once, crossing the line next to each other, which is a more likely scenario in real life conditions for a highway. Due to the fact that the project uses a non-model-based tracking system, cars are detected because they are moving, not because of any kind of object recognition being applied to the image; anything moving in the frame will be detected and tracked. With the method used, contours of moving vehicles are detected on each frame, however, the system has no memory of the previous frame. Also, it is not possible to predict exactly how many objects will be detected at a given time. This makes it hard to create variables to store the positions of all the contours and then be able to know which ones belong to which.

The counting problem was addressed with a simple method that made it possible to count cars with a considerable level of accuracy. For each frame, all the centroids of detected objects are stored in a list. When the program moves to the next frame, a loop iterates over the current contours. For each one, distances from its centroid to all the centroids in the previous

frame are computed with a numpy method : numpy.linalg.norm(a-b) (this method computes the euclidean distance between ‘a’ and ‘b’). With this procedure, the program determines that the centroid that results in the shortest distance belongs to the current contour. The idea for this is that an object’s position will not change drastically from one frame to another, and therefore when it moves, the closest point to it from the previous frame will be itself.

By applying the method above, the application was able to perform counting, although not with high accuracy. In fact, the program struggles and is easily confused when there is only one object in the scene and it disappears before another object is detected, and both objects are on different sides of the counting line. The consequence is that the program associates the last object’s position as the previous position of the new object and therefore it finds that the counting line is in between the two points, so the counter is increased.

A way to make the system more robust to these kind of problems is to set up a limit to what the maximum distance between two points can be. In this case, the maximum distance is set to be half of the width of the bounding box for a given contour.



An example of the counting method implemented. The red circle at the center of the green box is the current centroid for the object. The other end of the red line is the position of the centroid in the previous frame. Since the counting line lies in between them, it turns green, and the counter is increased.

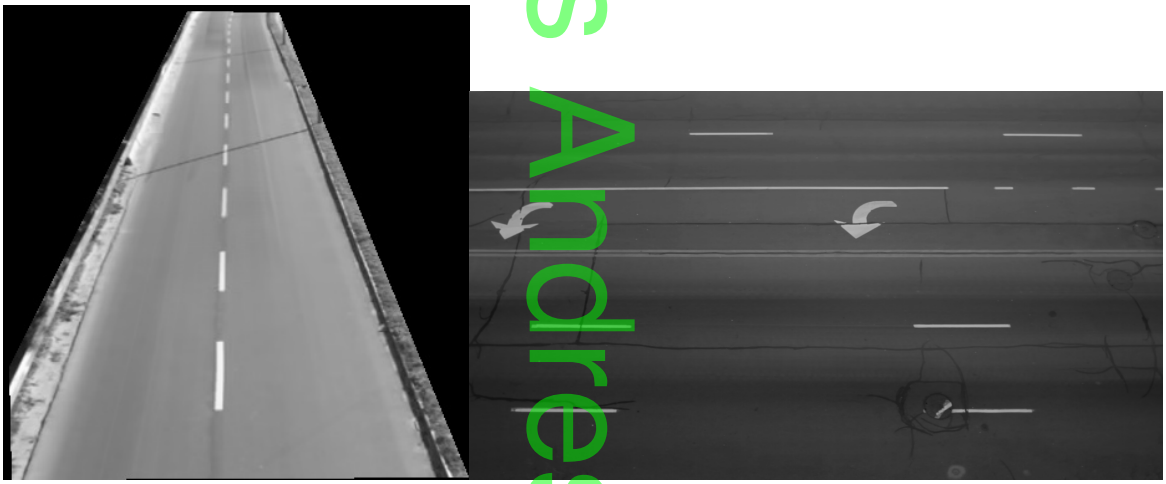
5. RESULTS

At the current state of the project, the traffic counting system has a mixed success rate. In some tests, it was able to count and be off by less than 3 vehicles. Additionally, in some of the tests, 100% accuracy was achieved. However, these results depend on very specific and ideal situations, which are not always present. Moreover, the system is still not robust or independent enough to be used reliably due to its need for good human input at the initialization phase.

5.1 SUCCESS

The program works best when the cars are clearly defined and not too close to each other. This differentiation helps the computer to recognize and separate contours of different objects, instead of counting them as one.

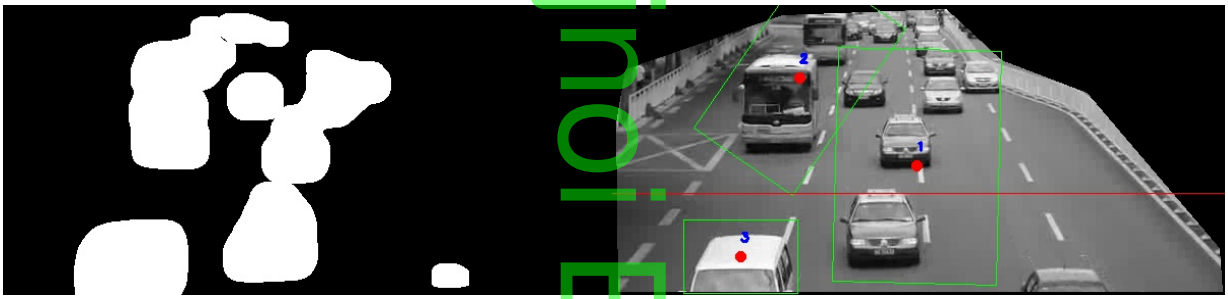
The success of the program also depends greatly on the angle chosen for the static camera. Angles that look at the road from above work best, as well as angles when the with camera facing the road directly in front. The reason for this is that these positions help avoid occlusions between cars.



To the left, an image of a road facing facing the camera. To the right, a road as seen by a camera from above. These two examples show the best angles for a static camera for the program.

5.2 FAILURE

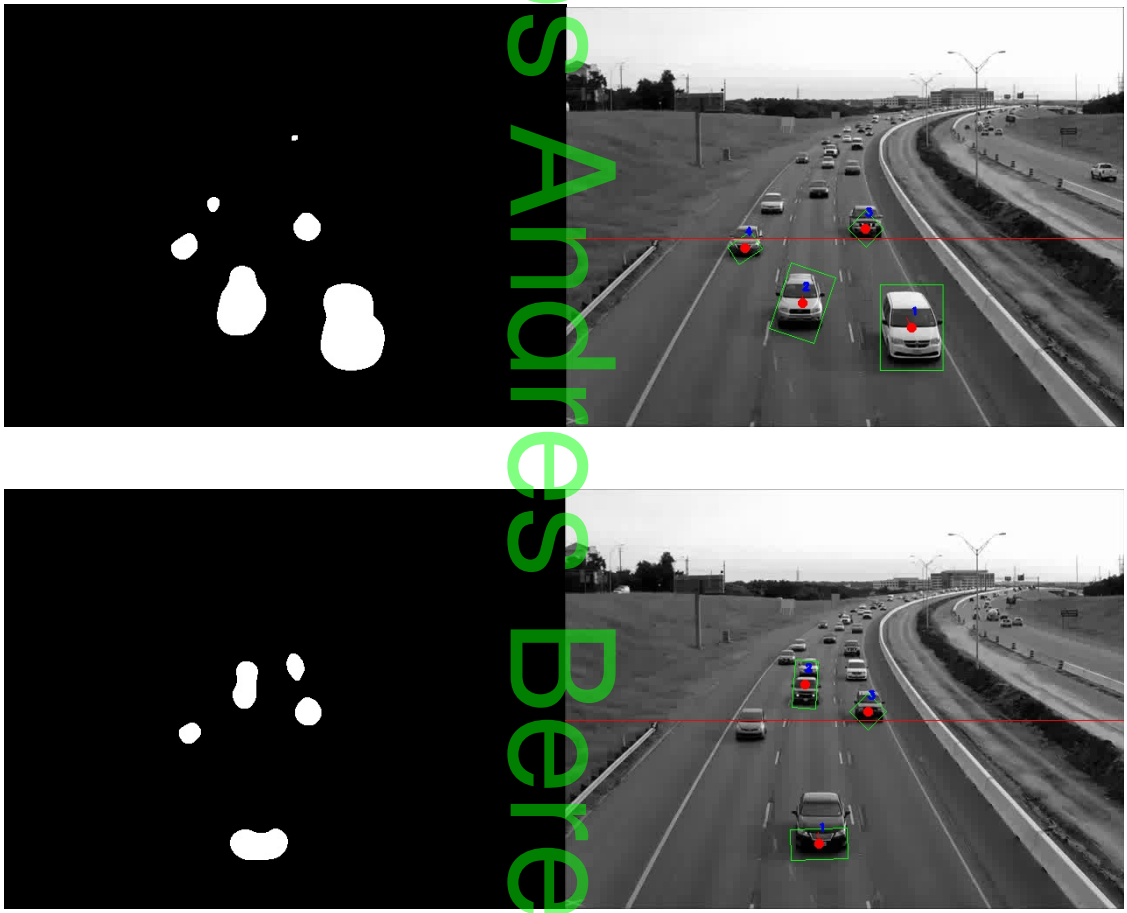
As mentioned before, the system is far from finished, and still has weaknesses that affect its accuracy at counting vehicles. For one, occlusions vastly affect counting. Because of the way contours for each vehicle are found, when one moving object is covering up part of another from the point of view of the camera, the program will assume it is just one object it is dealing with, instead of two separate ones.



This is a case of occlusion where many objects are detected as one. The image to the left shows what the computer is using to find contours. Connected regions are considered one. To the left, we can see that the cars in the middle do not seem to be occluding each other, however, they occluded each other in the background difference.

The next issue is that paint color in cars can also confuse the counter. Darker vehicles cause more problems because for the camera, their colors are not so different from that of the road, and therefore it has a harder time detecting that something has moved from the background image. This causes darker vehicles to not be detected correctly. Sometimes the

computer notices some changes but they are not big enough until the object is closer to the camera and already past the counting line before being considered as a car. On other occasions, the contour's area detected by the computer is not big enough to trigger the threshold to be count as a moving object, so it passes completely unnoticed.



Up: A frame showing the contours cars and to its right is the frame with bounding boxes for those contours. It can be seen that the contours are big and the vehicles have clear colors.
Down: To the left we see contours and to the right we can see the cars that produced them. The contours here are noticeably smaller than the ones in the row above. This corresponds with the vehicles having dark colors. If the car's color is similar to the color of the road, it is harder for the program to detect a change in the background, thus recognizing smaller contours.

Finally, another big problem for the system right now is illumination. Maintaining a running average of the background helps with slow illumination changes such as transitions from day to night or vice versa. However, sudden illumination changes can still affect the accuracy of the program. For instance, shadows cast by the moving cars are often different enough from the background to activate the threshold for motion detection. If the vehicle is far from others, this will most likely only cause the contour for that object to be bigger. On the other hand, if the vehicle is next to others, this often leads to those two or more objects to be detected as one because the shadow connects them both. This is similar to what happens when occlusion is involved.

6. DISCUSSION

In conclusion, the traffic counting system developed here shows that computer vision techniques can provide a solution to some tasks that could be expensive otherwise. The system was developed in under two months, with no previous knowledge of computer vision. This is a testament of how tools like OpenCV provide a very approachable way to solve problems of this sort easily.

There is more work to be done, especially regarding performance on an embedded device, and on reducing the high dependence on correct parameter setting from the user. Even with a short time to polish accuracy, the program demonstrated potential as a possible counting system, at least under specific conditions.

7. ACKNOWLEDGMENTS

The project could not have been possible without help from other people. Funding for this summer research experience came from the URCPP funds at Berea College. Besides, Dr. Matt Jadud provided me with the freedom to explore this area of computer vision and AI which interests me, as well as advice on some topics. Additional thanks to Dr. Scott Heggen and Dr. Mario Nakazawa for their availability when I needed to consult for different topics.

BIBLIOGRAPHY

1. Paul L. Rosin, “Thresholding for Change Detection”, (1998), CiteSeerx database. http://users.cs.cf.ac.uk/Paul.Rosin/resources/papers/thresh-ICCV.pdf.
2. Mahesh C. Pawaskar, N. S. Narkhede, Saurabh S. Athalye, “Detection of Moving Object Based on Background Subtraction,” *International Journal of Emerging Trends and Technology in Computer Science (IJETTCS)* vol. 3, no. 3 (Jun. 2014): 215-218, http://www.ijettcs.org/Volume3Issue3/IJETTCS-2014-06-24-126.pdf.
3. S. Susrutha Babu, S. Suparshya Babu, Dr. Habibulla Khan, M. Kalpana Chowdary, “Implementation of Running Average Background Subtraction Algorithm in FPGA for Image Processing Applications,” *International Journal of Computer Applications (0975-8887*), vol. 73, no. 21 (July 2013): 41-46, http://research.ijcaonline.org/volume73/number21/pxc3890259.pdf.