2018

Real-Time Object Recognition using a Multi-Framed Temporal Approach

Corinne Alini
calini@bowdoin.edu

Real-Time Object Recognition using a Multi-Framed Temporal Approach

An Honors Paper for the Department of Computer Science

By Corinne Taylor Alini

Bowdoin College, 2018

Contents

Acknowledgements

I would first and foremost like to thank my advisor, Eric Chown, for his valuable advice and insight that helped guide me throughout this project. I would also like to thank Bill Silver for his mentorship, instruction, and guidance. As teachers and mentors, their dedication and support of this project has proven to be invaluable. I am indebted to my colleagues for their aid and would especially like to thank Phil Koch and James Little. I am grateful for my non-computer science friends and family who reviewed this paper and endlessly listened to and aided with my presentation. Lastly, I would like to thank my dogs, Ginny and Madison, for inspiring this project with their obsession with chasing a moving ball.

Abstract

Computer Vision involves the extraction of data from images that are analyzed in order to provide information crucial to many modern technologies. Object recognition has proven to be a difficult task and programming reliable object recognition remains elusive. Image processing is computationally intensive and this issue is amplified on mobile platforms with processor restrictions. The real-time constraints demanded by robotic soccer in RoboCup competition serve as an ideal format to test programming that seeks to overcome these challenges. This paper presents a method for ball recognition by analyzing the movement of the ball. Major findings include enhanced ball discrimination by replacing the analysis of static images with absolute change in brightness in conjunction with the classification of apparent motion change.

List of Figures

List of Tables

List of Algorithms

1 Introduction

One of today's most important areas in Computer Science involved achieving human-level capabilities and one of the most challenging is developing algorithms for computer vision. Computer vision has applications in many modern technologies such as industrial robotics, autonomous vehicles and mobile. Computer vision is essential for identification, guidance, gauging, and inspection. For example, self-driving cars use machine vision to help with navigation, and applications like Snapchat use machine vision to analyze images and to produce filter features.

Robots provide a good example of the challenges involved in adapting vision algorithms to use in real world applications. With their mobile platforms, robots often have a limited processing power, yet the image processing required for computer vision is computationally intensive. Computer vision that performs as well as human vision may remain elusive, but improving efficiency and performance is not.

This paper examines the problem of real-time robotic vision for the Northern Bites RoboCup Standard Platform League (SPL) team. RoboCup is an international robotics competition in which different teams use the Aldebaran NAO robot in soccer matches. Using RoboCup as the platform for research, this paper will focus on real-time object recognition using a unique multi-framed approach. The goal of the paper is to determine if using motion for object detection is a viable option. This paper strives to perform object recognition to find a soccer ball in motion on the field.

Section 2 discusses the problem and Section 3 discusses the motivation behind the approach. Section 4 examines the static and dynamic applications of computer vision as related work. Section 5 provides an overview of the research presented by this paper. Section 6 examines the image differential portion of the new algorithm, Section 7 describes classification schema, and Section 8 describes its threshold. Section 9 focuses on the spot filter and detector. Results are presented in Section 10 with implications for future work and conclusions in Sections 11 and 12.

1.1 Background

1.1.1 Camera

The NAO robots include two identical SOC Image Sensor cameras located on the forehead (top camera) and mouth (bottom camera). Both cameras output images at 1280x960 resolution. The Northern Bites vision system requests 640x480 images from the top camera and 320x240 images from the lower camera. These images are captured at a maximum rate of 30 frames per second and are stored in the YUV 422 image format.

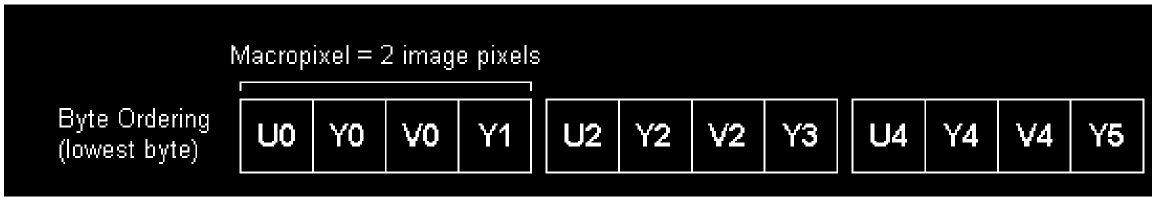

Figure 1.1: This is a representation of the YUV 422 image format [19].

1.1.2 Image Format

When the camera from the mobile agent takes an image, the resulting measurements are stored in the YUV 422 image format. The YUV 422 image format is a 2-dimensional array with each pixel formed by three channels: Y, U, and V. The Y component represents the luma or brightness and the U and V gives the chroma, or color. Without the U and V component, the image would be simply black-and-white. The 422 portion indicates the number of bites per channel. In a 422 image, there are 2 bytes of each U and V for every 4 bytes of Y. The YUV 422 image format is depicted in Figure 1.1. The y value is used with its neighboring u and v in order to create a colored pixel value.

1.1.3 Processor

The processing power of the NAO robot is limited when compared to the capabilities of other platforms. The processor runs at 1.6 GHz meaning that it can only execute 1.6 billion instructions a second. For an application that allocates 50% of its computing time to vision and since both cameras get 30 frames a second, this means that the processing power limits the application to only approximately 13 million instructions per image. Since an image can vary from 640x480 to 320x240, this results in a maximum range of 44 to 173 instructions per pixel. Thus, there are serious limitations to the types of algorithms that can be used in such vision system.

As a basis for comparison, the iPhone 5 has a dual-core processor that runs at 1.3 GHz. This means that each processor can run 1.3 billion instructions per second, and therefore a total of 2.6 billion instructions per second. If NAO robots had this processing power, then the maximum range of instructions would increase to 141 and 564 instructions per pixel. An extra GHz allows the algorithm to be slower and therefore have more accurate prediction property. The constraints inherited on Aldebaran's NAO robotic platform require the visual algorithm to be extremely computationally efficient.

2 The Problem

While computer vision has been improved by the advent of Deep Learning Neural Networks, real-time robotic vision remains far from solved. A robot must perform visual processing in $\frac{1}{30}$th of a second per frame, finding the necessary objects in the scene. The robot not only has to locate the soccer ball, but also has to identify goal posts and other robots. In addition, the robot cannot commit all of its processing power to vision, but must also walk, coordinate with other robots, and perform many other behaviors in order to play soccer. The robot has a very limited amount of time to spend on analyzing the image. In addition, since the SPL league dictates that the hardware of the robot cannot be changed, the robot only has one processor to use. This severely limits the amount of time a robot has to find the ball.

In addition to these time constraints, the robot has to be extremely accurate with its identifications. Any mistakes in identification are devastating in RoboCup and can cost games. *False positive* identification happens when the robot sees a ball where one does not exist. This false positive results in the robot moving in the wrong direction as well as communicating a false positive location to teammates who then also might move to the wrong location. If a robot's identification is a*false negative*, it does not identify the ball where there is a ball. False negatives can result in a robot continually looking for a ball rather than moving the ball towards the goal.

The identification of a ball is especially problematic for NAO robots due to similar color and shape between these robots and soccer balls. Both soccer balls and robots have dark area scatters across a white background as seen in Figure 2.3. Furthermore, the goal posts, lines, and random objects in the background are also white. Color cannot be used as a search parameter for reliable object recognition. While shape is often used for object identification, the robotic feet, robotic heads, and soccer balls all have a very similar curved shape. Robots often fall so their curved heads as well as

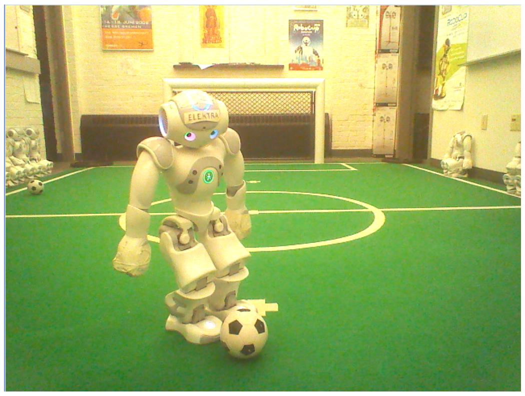
Figure 2.1: This image is an example of the ball next to the robot's foot.

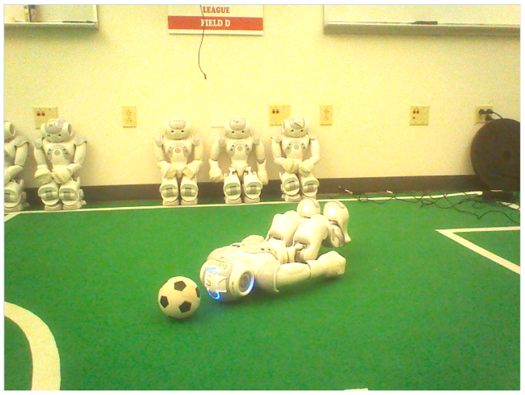
Figure 2.2: This image is an example of the ball next to the robot's head.

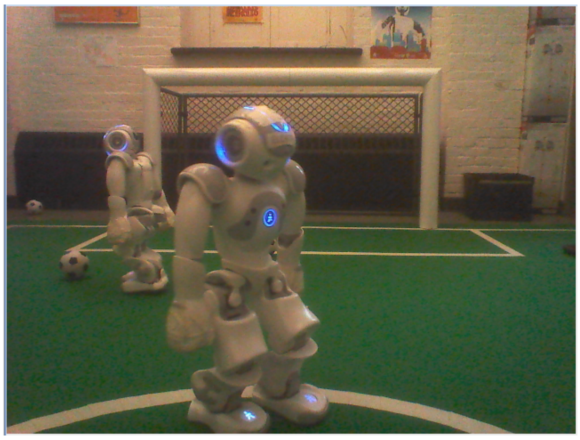

Figure 2.3: This is a frame of a scene with multiple moving robots and a moving ball. This image is an example of a more complicated scene that a robot may see in a given game.

their curved feet are located on the ground in same location where other robots would be searching for the ball. In Figure 2.2, the robot's head is very close to the soccer ball and in Figure 2.1 the robot's feet are close to the ball. Other search parameters need to be identified since only utilizing color and shape in searches yields many false positive ball recognitions.

3 Motivation and Inspiration

If a human was asked to identify the images in Figure 3.1, they would easily describe a soccer ball roughly in the middle of the field. They would properly identify that the cross in the middle of the field was not the soccer ball.

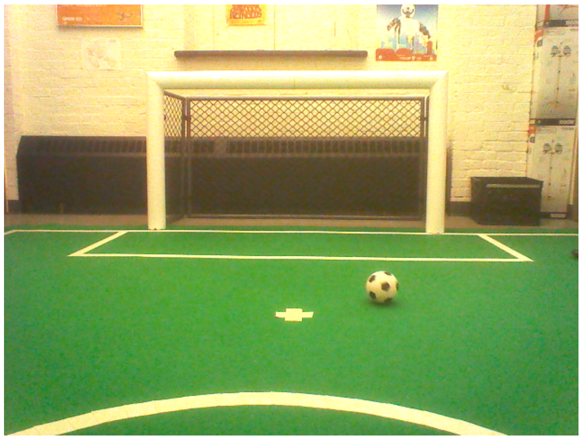

Figure 3.1: This is a frame of a ball rolling across the soccer field. This is an example of a rather simple scene that presents a challenge in differentiating between similar shapes of the cross on the field and the ball.

Humans can accurately identify an object in 350 milliseconds [6]. A person would not be able to identify the soccer ball if they were given the 33 milliseconds or $\frac{1}{30}$th of a second [6]. They would not successfully be able to recognize the soccer ball with the constraints of the NAO robot in RoboCup. However, humans need less than 100 ms per image if the images are presented sequentially [6]. A human would need less time to find the soccer ball if presented with a sequence of images.

If the scenes get more complicated with more images like in Figure 2.3, even a human would need more time to find the soccer ball. The fact that the scene is poorly lit and there are other moving objects makes the task even more difficult. Given that humans are more robust at object recognition than computers, computers would need an extended length of time to find the soccer ball, time that does not exist for a real-time application to a soccer match.

The motivation for this project is to explore the human characteristic of improved visual recognition efficiency with sequential imagery and investigate if sequential imagery can be applied to machine vision and exploited to improve object recognition. This project uses human object detection as inspiration to ameliorate the speed of computer object detection.

4 Related Work

For inspiration and in preparation for this research, both static (single-framed) and dynamic (multi-framed) object detection methods were examined. Single-framed approaches have the advantage that each identification is mutually exclusive: a false positive identification made in a previous frame does not affect the next identification. They also are more versatile as they can identify both moving and still objects. However, this approach does not maintain any previous information, which can slow down the time to find the object. Dynamic object detection provides additional data about the object such as its movement that static approaches cannot provide. Knowing where the object was in the previous frame narrows search parameters and can speed up the process of object identification. This narrowing of search parameters is a critical step in increasing speed, yet any misidentification at this step can be devastating. False identifications in previous frames can negatively affect current identifications as the frames are not mutually exclusive. The dynamic approach is also limited in that it cannot identify stationary objects.

4.1 Static Object Classification

The majority of the RoboCup SPL league has adopted either a machine learning or heuristic approach to identifying objects. The Northern Bites team utilizes a heuristic approach, meaning the robots find candidate regions and then filter them to further classification.

4.1.1 Northern Bites Ball Detection System

A difficult problem in robotics is how to visually identify complex objects. Ball detection has proven to be very challenging aspect of robotic soccer. When the SPL league switched to black and white balls, the recognition of these balls could no longer rely upon color identification. Most objects in the robotic visual image have the same color scheme like robots and goalposts. The uneven lighting on the field masks black-and-white soccer balls. Shadows cast by three-dimensional objects like robots, goal posts and referees further impede accurate ball detection.

The Northern Bites ball detection system runs the YUV image though a constant-time spot filter that detects spots. A spot is defined as an area with a large difference in gray-levels between that area and its surrounding region. The algorithm calculates the color contrast between the inner and outer region to determine candidate regions. The spot detector algorithm outputs a list of black spots and a list of white spots, which serve as candidate regions for the heuristic analysis. (see Section 9 for further details regarding the spot filter and spot detector).

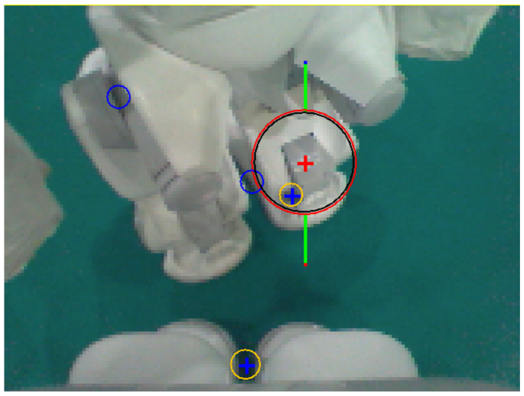

Figure 4.1: The current ball detection system falsely classified the back of a robot's leg as a soccer ball [3].

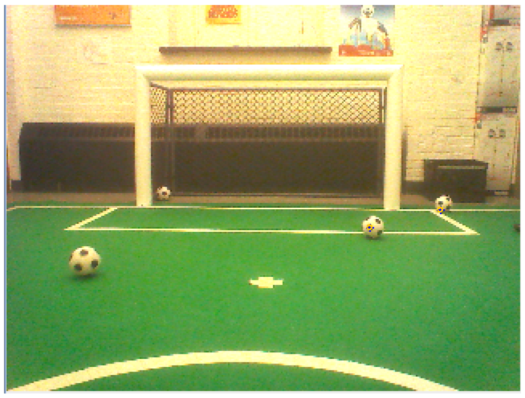

Figure 4.2: The current ball detection system was unable to identify any of the balls in this image.

The ball detection algorithm then removes all the white spot candidates above the horizon. It also removes any candidates that are identified as too small. The next step is to remove robots by quantifying the amount of white below the spot and eliminating if there is excessive white below the spot. Finally, it counts the number of black spots inside and considers the candidate a spot if there are enough black spots within the larger white spot.

4.1.2 Problems with Current Ball Detection System

The current ball detection system makes many mistakes in identifications and struggles to identify soccer balls in particular scenarios. Even with the extensive heuristic system, the robots often misidentify robot heads, jerseys, and knees as soccer balls. The misidentification of a robot knee is seen in Figure 4.1. More importantly, the ball detection system struggles with recognizing moving balls as seen in Figure 4.2, which has proved fatal for Bowdoin's goalie. Since the ball is moving constantly in an efficiently played game of soccer, the robots are unable to find the ball a significant portion of the time.

The ball detection system of the Bowdoin RoboCup team analyzes a single image to find the ball. All apparent motion is eliminated in this process of examining one frame. Since the movement of a ball is much different from robots, referees, and all other motion a robot would see, the ability to analyze motion would greatly reduce processing time and improve ball detection accuracy. Knowing information about the balls previous location allows the algorithm to focus its search on particular regions of the image instead of the whole image.

4.1.3 Machine Learning Approach

A few RoboCup teams in the league have adopted machine learning techniques like Convolutional Neural Networks (CNN) to find the ball [2, 9, 14]. CNN is a supervised machine learning technique that is similar to a regular neural network with the assumption that the inputs are images. In 2016, SQPR from Sapienza University of Rome described their approach to ball detection using a CNN which was independent of color variation and illumination [2]. They combined the CNN with image region segmentation to reduce the search space. They trained against a positive and negative training set to train their classifier.

University of Texas Austin used a hybrid approach that combined machine learning with a heuristics approach to create an effective ball detection system as described in [13]. They use a region-of-interest filtering pipeline approach with a Deep Neural network to create a fast object detection system that was able to detect the ball's position with no prior knowledge. This means that their algorithm does not know anything about the previous whereabouts of the ball to find it. Their algorithm initially applies a high-level heuristic search to quickly identify ball candidates. From there, it uses more rigorous filters to remove false positives.

The main issue with a purely machine learning approach is that many machine learning algorithms require thousands to millions of labeled samples to produce a robust and accurate system. Samples need be collected and labeled manually in a laborious and slow process. In addition, the machine learning approach itself requires time. The neural network needs an inference time that is within the real-time constraint of 33 milliseconds. While algorithms do exist that are sufficiently efficient for RoboCup implementation, the platform severely limits which neural neural network can be used. UChile tested a variety of networks and eliminated most including Alexnet and Darknet-CIFAR10, which had inference times of 7400 milliseconds, 4400 milliseconds respectively [5]. In fact, UChile only found two neural networks with acceptable inference times for the RoboCup platform ($\sim$ 1 milliseconds) [5].

4.1.4 Heuristics Approach

Another approach to ball recognition is to create candidate regions and then weed out candidates.

The UChile Robotics Team from Universidad de Chile developed an algorithm that begins by finding white regions in the image that are neither a white goal post nor a field line [11, 12]. White regions are identified using scan lines. For each white region, an interior point, a starting point for a vertical and a starting point for a horizontal scan are computed. The scan stops when there is a change in color. The white regions are then send through classification which runs the region through numerous filters like “is the ball near the robot” and “is the current ball spot is inside an already detected ball” [11, 12].

B-Human from University of Bremen and the German Research Center for Artificial Intelligence (DFKI) developed a multi-stepped heuristics approach to detect the ball [16]. They first search for ball candidates using scan lines of different densities. They then find the directions of the gradients through the creation of a contrast-normalized Sobel image. From there, they look for the ball contour in the search space and then take those with the highest response and check the ball pattern.

While computationally more efficient, the heuristics method is not general purpose. This method is developed for particular tasks and is not applicable to general object detection. Finally, this method requires more trial and error. It does not always lend itself to an obvious simple solution and it can take years to develop a usable algorithm.

4.2 Dynamic Object Classification

Object recognition using motion often requires a multi-framed approach. The algorithm needs to examine the progression of time to determine which pixels correspond to movement. While many of the current approaches are too slow for this problem, they provide the general inspiration for this approach.

4.2.1 Optical Flow

Currently, motion detection in computer vision is often estimated using optical flow. Optical flow is defined as "the apparent motion of brightness patterns in the image" [17, 4]. It is based upon finding the apparent motion in a sequence of images. In Figure 4.3, the vector field demonstrates how the box moves between each frame [1]. The 2D vector field demonstrates the directional change in motion.

The idea is to fuse a 2D vector field to map the displacement of an object between frames. The goal is to estimate the pixel motion from image $I(x, y, t)$ to $I(x+\Delta x, y+\Delta y, t+\Delta t)$.

If there is a pixel with intensity $I(x, y, t)$ in a frame, x and y are the coordinates of the object, t is the time interval, and I is the intensity of brightness. In the next frame, the object moved by Δx and Δy taken after Δt time. Assuming that pixel intensities are translated from one frame to the next for the gray-scale image,

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

This equation is called the *Brightness Constancy Constraint Equation.* To solve this equation, take the Taylor Series Expansion of I, which results in:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + H.O.T$$

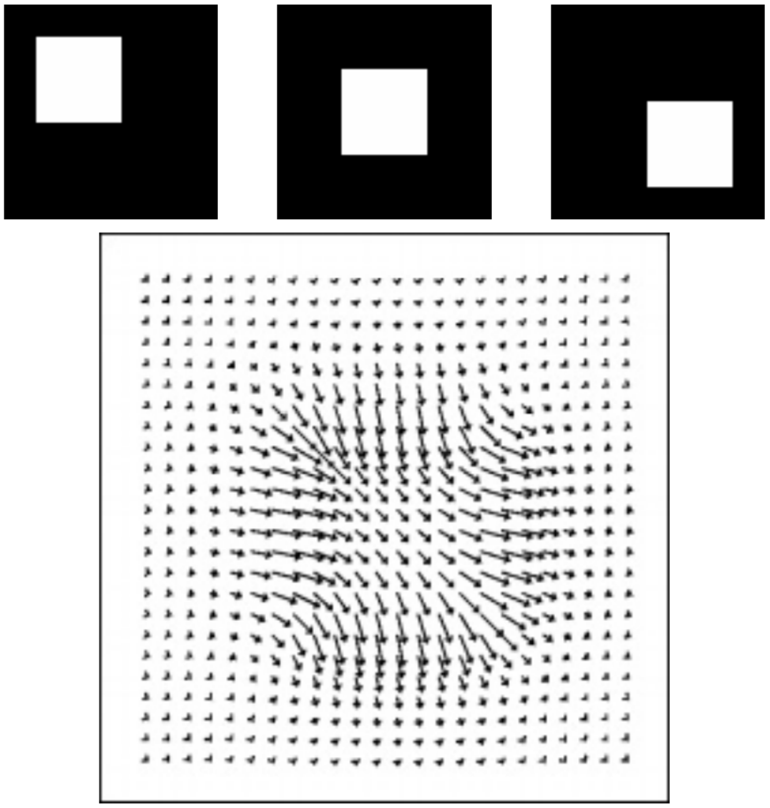

Figure 4.3: This is an example of how optical flow uses a vector field to find the apparent change in motion [1].

From here, it simplifies to:

$$0 = \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

So, using the short hand $I_v = \frac{\partial I}{\partial v}$ it simplifies to,

$$\begin{aligned} I_x\Delta x + I_y\Delta y + I_t &= 0 \\ \nabla I \cdot (\Delta x, \Delta y) &= 0 \end{aligned}$$

Thus, the component of the flow perpendicular to the gradient is unknown. Since there are two unknown variables, Δx and Δy, the equation cannot be solved in its current state. Thus, the optical flow algorithm needs additional information to estimate the actual flow. This is called the *Aperture Problem* [17, 4, 10]. Methods have been developed to add additional constraints to estimate optical flow. One such method includes the *Horn-Schunck* method, which introduces a smoothness constraint [10]. It begins with the assumption that there is smoothness in the flow over the whole image. This allows for an estimation of the velocity field that minimizes the equation above for each pixel in the image. This iterative implementation allows for the estimation of optical flow that has been shown to be both insensitive to brightness levels and noise.

While this method seems appropriate to multi-framed object recognition, it is ineffective for real-time applications. As described with the *Horn-Schunck* method, the iterative relaxations require multiple passes over the image and multiple computations per pixel. In such a computationally limited environment, this method requires a large number of operations per pixel to achieve acceptable estimation accuracy. As previously explained, the processor only has about 173 instructions per pixel. Therefore, optical flow is theoretically interesting, but is too computationally intensive and cannot handle this type of moving object recognition efficiently. For example, Talukder et al. created a real-time optical flow algorithm for mobile platform. Their algorithm runs at 6.8 frames per second on a 320x240 image on a 2.1 GHz platform [18]. This algorithm is still too slow for the RoboCup time constraints even though it is very efficient and runs in real-time.

5 Overview

The goal of this work is to explore a new method for soccer ball detection. The current object detection method does not incorporate motion, which could be used to improve accuracy of ball detection. The paper describes a new method designed to improve the current ball detection system that incorporates ball motion in the processing by analyzing multiple frames at a time.

Motion can be detected by analyzing the ball's edges. When the RoboCup league utilized a solid orange colored ball, motion would be detected by color change in the image at the ball edges where the color in the image went from orange to the field color green and green to orange. However, a black and white soccer ball has the added advantage that the interior pattern has apparent motion while the ball is rolling. Going from black to white creates high contrast from image to image, much more than contrast in the image of a robot. The algorithm uses this theory as the foundation for the algorithm.

There are 3 major steps to this algorithm:

1. Measuring Change.
2. Discriminating apparent changes in motion from object motion.
3. Discriminating between different moving objects.

The algorithm measures change by using an absolute value differential image method that is not only time efficient, but also robustly recognizes spherical objects with constant motion. As explained above, the algorithms used for RoboCup require quick and efficient processing in order to run. The multi-framed ball detection system works by first finding the change in brightness. This is done by taking the difference of the Y-components of two images. The difference image demonstrates what parts have changed from one image to the next.

The next step is to find the objects that are moving and to attenuate the effects of other types of apparent motion. This is done through the use of a threshold - a limit that indicates at what pixels are deemed noise.

From there, the algorithm runs the image in the spot detector and filter to produce a list of candidate spots. This method looks at the overall image to identify objects that are similar to the ball.

5.1 Apparent Motion

Motion vision is done by examining frame by frame changes in brightness over time. Therefore, anything that causes a change in brightness is perceived as motion.

When doing image processing with motion there are four factors that produce changes in gray-levels:

1. Object
2. Light
3. Camera Movement
4. Noise

Variation in any of these factors will cause an apparent change in motion in the image. For example, if the only movement in the scene is an object like a ball or robot, its movement will be depicted in the image as change in gray levels. Also, camera movement will also induce apparent motion in the resulting image. When a person takes an image with a camera while the camera is moving, the resulting image is blurry. If looking for the motion of the image, it would be difficult for a computer to differentiate this case from one in which everything in the image is moving. Finally, changing light will also induce apparent motion. Given that motion is identified by a change in brightness of a pixel, if the lighting changes from dark to light or vice versa then that pixels brightness will change accordingly. This will be viewed as a change in motion when nothing actually moved. Shadows, camera flashes, and other changes in ambient light may be interpreted as motion since finding changes in brightness is used to identifying motion.

Image noise affects the perceived motion of an image as well, which will be discussed in the threshold section.

6 Image Differential

The first step of the algorithm is to confirm that every frame taken by the robot is consistently passed to the code. This is important, as missing frames cause chronological gaps that create incorrect identification due to inconsistencies with the time intervals between frames.

Examining the number of frames passed from the robot to the code tested whether or not the robot receives all the frames. If the robot were not passing every frame then the code would see less than 60 frames a second (30 for each the top and bottom camera). The code did receive 60 images a second and thus confirmed that the robot does pass all 60 frames to the code.

The next step is to determine what is moving in the image. In order to improve speed and space complexities, this was achieved by taking the difference between two images as shown in Algorithm 1. A y-image is an image composed solely of the y components of the original image (also called gray-levels). Y-images were chosen over subtracting the whole image for their simplicity and reduced complexity.

Algorithm 1 The algorithm for creating the differential image.

```
1: procedure DIFFERENTIAL IMAGE
2:     $prev\_image \leftarrow$ y-values of the previous image in the sequence
3:     $curr\_image \leftarrow$ y-values of the current image in the sequence
4:     $differential\_image \leftarrow$ Initialize the differential image
5:     if $prev\_image.width() == curr\_image.width()$ then
6:         if $prev\_image.height() == curr\_image.height()$ then
7:             for each $w$ in $width$ do
8:                 for each $h$ in $height$ do
9:                     $differential\_image \leftarrow |prev\_image[w][h] - curr\_image[w][h]|$.
```

To find the difference image, the algorithm begins by confirming that the images are of the same size. Since the robots two cameras produce different sized images, sequential images must come from the same camera or the size differential would be perceived as motion.

After confirming that the images are the same size and of the same camera, the algorithm creates a new image to store the difference image. Then it iterates through each pixel and sets the difference image to the absolute value of the difference between the y-values of the images. This results in a black and white image that shows the apparent motion of the image through the depiction of the change in brightness as seen in Figure 6.2.

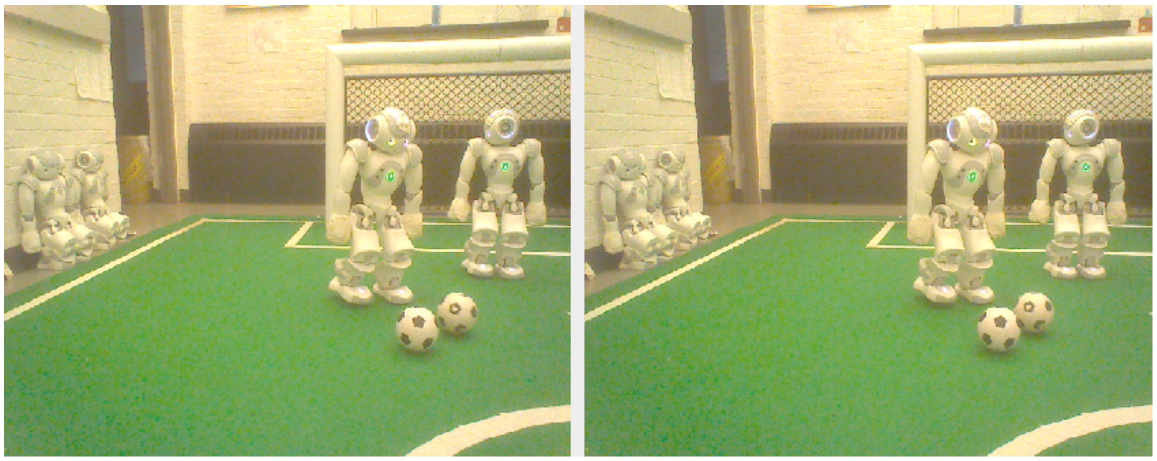

Figure 6.1: This is an example of a sequential scene used to test the absolute differential method. Note that everything in the image except the leftmost ball is moving.

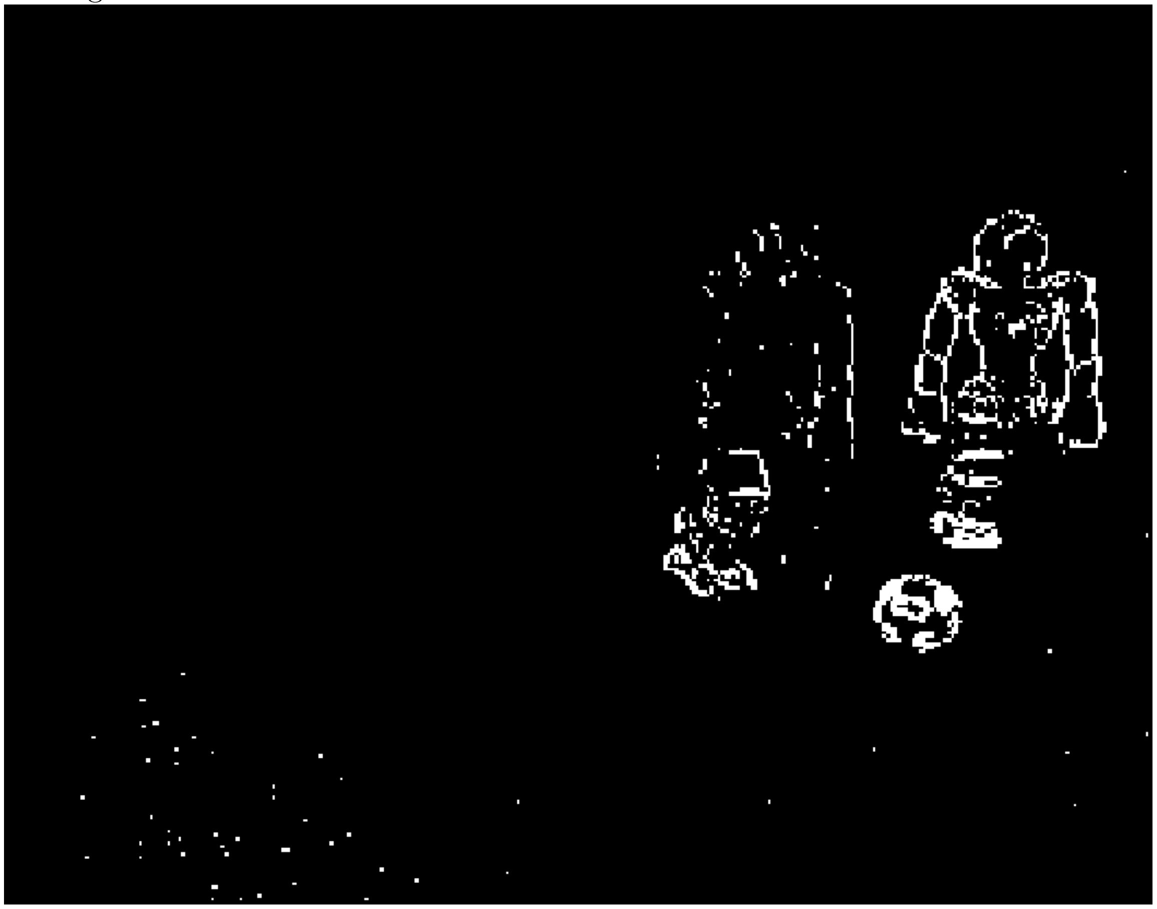

Figure 6.2: This is a depiction of the difference image from two images (with a binary threshold for visualization).

7 Classification

The next step is to determine how to use the absolute differential image to find the ball. Our approach uses two steps to perform classification:

1. Amplify the pixels that are part of the ball and attenuate the other pixels that were due to apparent motion.
2. Discriminating the ball from the robot. At a given time, there can be between 0 or 9 robots in the image.

Once given the raw measurements as the absolute differential image, the algorithm needs to make a distinction between what information is important and what is not. The method modifies each pixel by ideally reducing the brightness of the pixels that are not the ball and increasing the brightness of the pixels that are the ball. This is done by looking individual pixels using thresholding.

Once the desired information is isolated from extraneous information, the next step is to identify the soccer ball and return a result. The algorithm uses two theories of operation about the relationship between gray-levels and the ball:

1. A moving ball has very high spatial density. This is predicated upon the idea that a pixel's neighbors could help determine if that pixel is part of the ball. The algorithm uses a spot filter and spot detector to find areas in the absolute differential image where there is a lot of black surrounding a white spot of roughly the size of a soccer ball.
2. Higher absolute differences are more likely to be a moving ball. Regions of the image that have a ball should be brighter from our thresholding than regions with robots. Furthermore, pixels below a certain point will convey almost no information to help our search for the ball. These low points could be due to noise or robot motion. Therefore, our method should not include or at least attenuate these pixels.

Using these two ideas, different thresholding methods are applied to continue ball classification

8 Thresholding

With the differential image, the algorithm needs to remove ambient noise created from the uncertainty in the process of taking a picture. A camera takes an image by making measurements of brightness, which results in some uncertainty. This ambient noise could be wrongly viewed as motion between the images

Such noise is uncorrelated, as this uncertainty does not relate to a particular region of the image or time. This lack of correlation is fortunate as the noise can potentially be removed from the image by introducing a threshold.

The following subsections present the three thresholding methods that were tested: binary, linear, and quadratic thresholding.

8.1 Binary Thresholding

The first method tested was a step function. Step functions change values such that the result can either be true or false depending on its relationship to the threshold. In the case of the differential image, this means that if the pixel value is less than some threshold, the value was set to 1023 or white, otherwise it is set to zero or black as seen in Figure 8.1. This threshold is a *binary threshold* as it follows the step function logic.

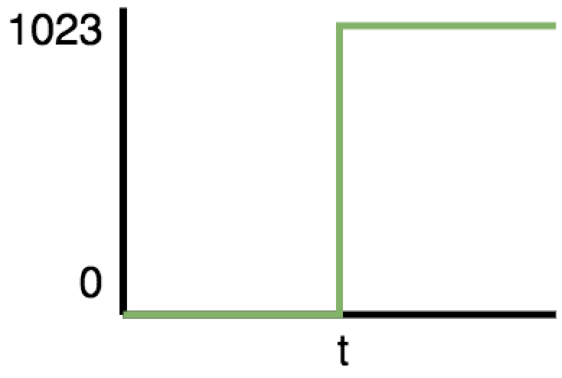

Figure 8.1: This graph demonstrates how the binary threshold sets the value. t is the binary threshold. If the pixel value of the differential image is below this threshold t, the value was set to zero (black), otherwise the value was set to 1023 (white).

The threshold was calculated by first finding the standard deviation of a given differential image. Thus, the noise is removed by finding which pixel values are outliers. This was done by having the robot taking images of scenes at different lighting conditions with no movement. The lighting conditions tested can be seen in

Figures 8.4, 8.5, and 8.6. The threshold is impacted on different lighting conditions. Since the lighting conditions are not always constant at competitions, both extreme settings were used to determine a threshold that would work for all types of situations. From these images, the difference image was computed and the standard deviations of gray levels (y-values) were computed.

8.1.1 Results

Overall, this thresholding method allowed for an easy way to eliminate pixels believed to be noise. Due to its simplistic nature, using this threshold is a quick and efficient operation that has little to no impact on the running time of the algorithm. This method was applied using both static and dynamic thresholds. However, two major concerns appeared regarding choosing the threshold and discriminating robots from balls. These issues led other options being explored.

Static Binary Threshold

Static threshold method involved finding the average standard deviation between multiple, diverse images. Out of 20 different scenes analyzed, the average of the standard deviations was 51.57. The problem with using a static threshold became apparent when changing lighting between different environments. RoboCup does not have a constant brightness for their lighting, so this method did not prove effective, as it did not allow for automatic adjustments for different lighting scenarios.

The standard deviation varied too much with varying lighting conditions and it was too hard to robustly discriminate the robot from soccer ball using this method.

Dynamic Binary Threshold

Through dynamic threshold testing, 3 standard deviations were found to be the most effective dynamic binary threshold to eliminate the noise. This method effectively removed the noise in a series of images allowing for the movement to be clearly seen.

Different lighting conditions were now introduced into the experiment using a thresholding method to remove robots from the scene. Thinking that the difference between the black and white spots of a soccer ball is more drastic than the dark and light parts of a robot, this was tested to try an eliminate robots. This was done by finding the standard deviation of scenes involving the movement of just robots and just soccer balls. Testing illustrated that there is no fixed threshold that allows for discrimination between balls and robots as demonstrated in Figure 8.3. Therefore, the dynamic binary threshold prevented accurate discrimination between balls and robots for the spot filter and spot detector.

Overall, binary thresholds were effective at eliminating noise from the image, but were unable to discriminate balls from robots. The major issue with the binary

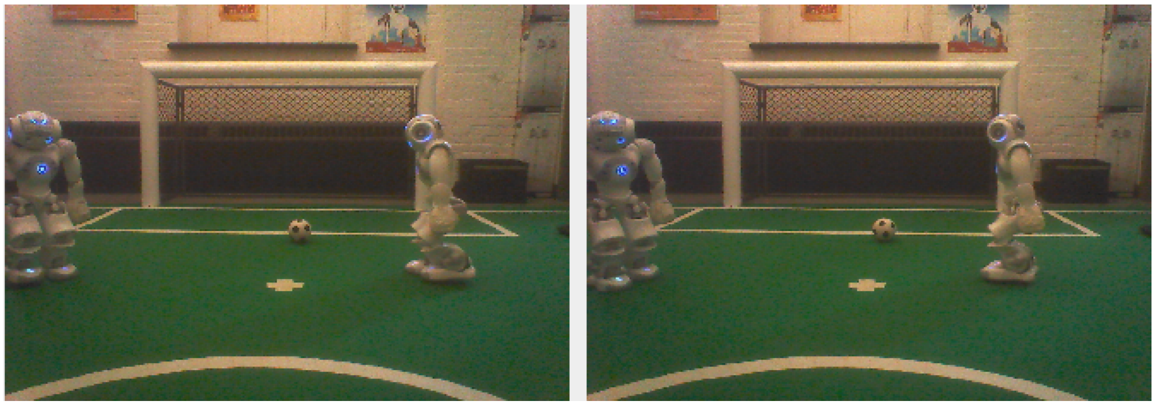

Figure 8.2: This is an example of a sequential scene used to test binary thresholding. Note that both the robots and the ball are moving.

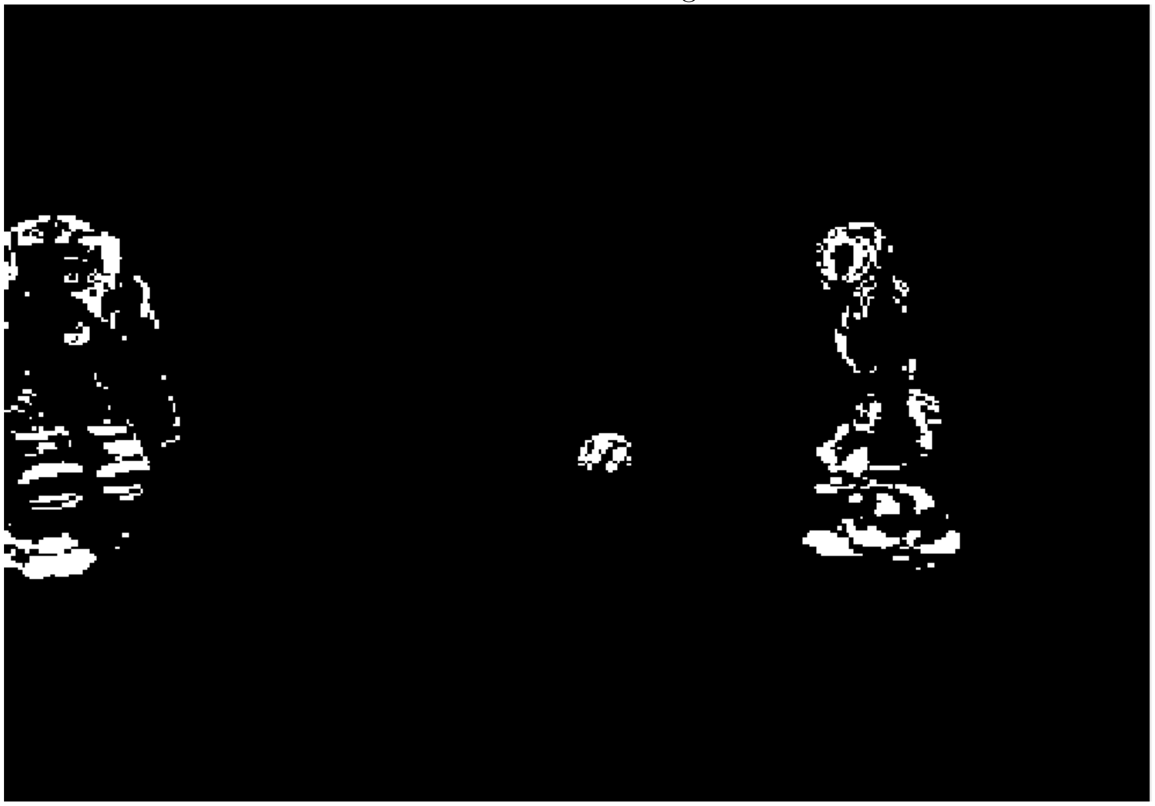

Figure 8.3: This is a sample image of the output produced from binary thresholding.

threshold is that it removes all discernible differences between the soccer ball and robot. Consequently, the binary threshold did not provide an effective method for finding just the soccer ball.

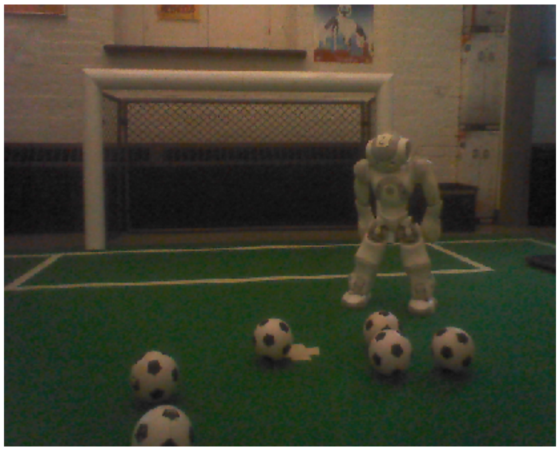
Figure 8.4: This is an example of an image used to test the binary threshold with minimum lighting.

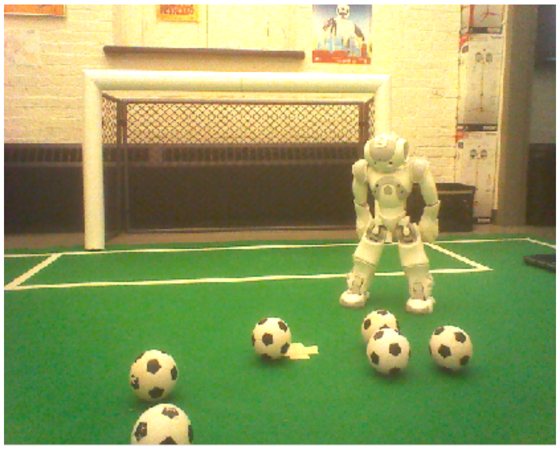
Figure 8.5: This image is an example of an image used to test the binary threshold with average lighting.

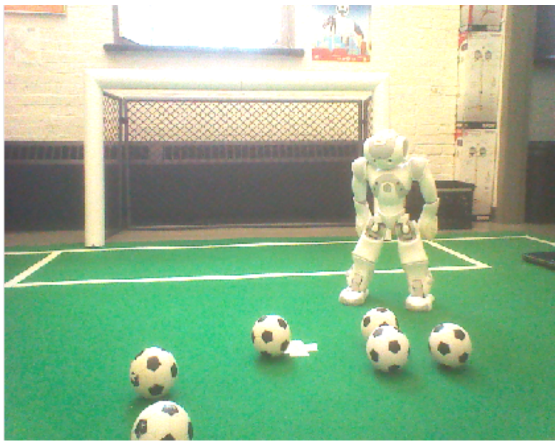
Figure 8.6: This is an example of an image used to test the binary threshold with natural or extreme lighting.

8.2 Fuzzy Threshold

The other method for thresholding explored was *Fuzzy Logic*. Fuzzy logic allows for a conversion from raw measurement to a value that indicates relative confidence. Instead of being either true or false, the value can be partially true or partially false. This concept applies to the differential image each pixel will not be black or white, but can take on intermediate values. The raw measurements of the differential image are the absolute gray level difference between successive frames.

8.2.1 Linear Threshold

The first type of fuzzy threshold applied was a linear threshold. This thresholding method allowed for small changes between similar pixel values and allowed for a more gradient approach.

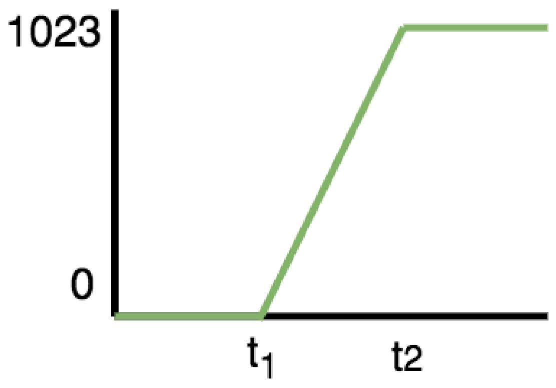

Figure 8.7: This graph demonstrates how the linear threshold sets the value. t_1 is the lower threshold and t_2 is the upper threshold. If the pixel value of the differential image is below t_1,the value is set to zero (black). If the value was greater than t_2 the value is set to 1023 (white). If the pixel was between the two values a fuzzy threshold is applied.

A basic linear threshold is shown in Figure 8.7. The value is set to the smallest value (0) if the value is less than t_1, and is set to the largest value (1023) if it is greater than t_2. In between t_1 and t_2 then it would be scaled to a value between the largest and smallest value according to its value. It is linear because the scaling is done using the equation for a line.

The algorithm described in Algorithm 2 demonstrates how the linear threshold was applied to each pixel. If the pixel was less than t_1, the lower threshold then it was set to 0. This was done to remove the lower outliers that were most likely to be noise.

If it was between these two thresholds then fuzzy threshold is applied:

$$confidence = slope * max(raw\_input - low\_threshold, 0).$$

This equation has two unknowns, the lower threshold and the slope. Different value combinations for lower threshold and slope were tested. The testing found that the standard deviation of the absolute differential image produced the most accurate results. Slope testing results can be found in Section 10.

After the equation was applied, if the confidence was greater than 1023 then it was set to 1023. This was to confirm that that no pixel could be larger than the max pixel value.

Algorithm 2 The algorithm for applying the linear threshold.

```
1: procedure LINEAR THRESHOLDING ON A DIFFERENTIAL IMAGE
2:     Given: a gray-level $raw\_input$ from the absolute differential image, lower
   threshold, and slope.
3:     if $raw\_input < low\_threshold$ then
4:         $confidence = 0$
5:     else
6:         $confidence = slope * max(raw\_input - low\_threshold, 0)$
7:     if $confidence > high\_threshold$ then
8:         $confidence = 1023$
```

8.2.2 Quadratic Threshold

A quadratic threshold was also explored and applied to each pixel. The graph of a general quadratic threshold can be seen in Figure 8.8.

This method also attenuates low values and amplifies high ones. This thresholding method is based on the belief that higher values should always correspond to a higher confidence that this pixel is part of the ball. This is due to the high contrast of the ball.

The algorithm applies the following equation to each pixel:

$$confidence = raw\_input * raw\_input/some\_value.$$

First, the problem with this equation is that division is an expensive operation. Given the time constraints presented earlier in the paper, the algorithm needs to be as fast as possible. In C++, Shift operations are significantly faster than divides. Therefore, if a way could be devised to use a shift instead of a divide then the efficiency of the algorithm would be improved.

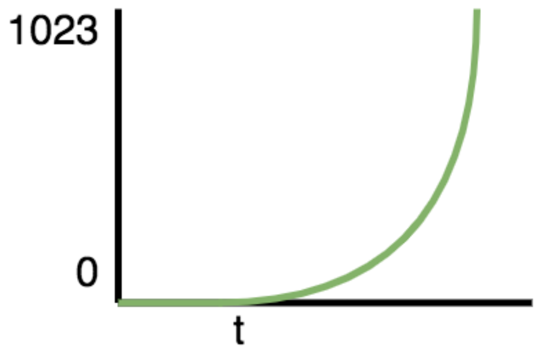

Figure 8.8: This graph demonstrates how the quadratic threshold sets the value. t is the quadratic threshold. If the pixel value of the differential image was below this threshold t, then the value was set to zero (black). otherwise it was set to $raw\_input * raw\_input << value$.

Utilizing the fact that shifting by some x is the same as dividing by 2^x to improve our algorithm:

$$confidence = raw\_input * raw\_input >> x.$$

The tradeoff for shifting is that it can only be divided by a power of 2. This significantly limits the precision when compared to dividing.

The next step was to determine an best value for x in order to find the maximum number of balls. Since there were only 10 values from which to choose ($2^{10} = 1024$, our max value), x was tested on multiple images from different scenes in order to find the best value.

9 Spot Filter and Spot Detector

After thresholding, the resulting differential image is left with differences that are likely due to motion. This motion should be a result of object motion. Thus, the algorithm takes advantage of the second theory of operation: a moving ball has very high spatial density. The algorithm uses a spot filter and spot detector to find candidate regions where the average of inner region is less than the average of the outer region. The next step is to run the differential image through the spot filter. The spot filter and spot detector are run twice to find both white and black spots.

9.1 Spot Filter

After finding the differential image, the algorithm runs it through a spot filter. The spot filter is an algorithm that detects places in the image where central region of values significantly greater or smaller than surrounding. Since a soccer ball is white with black spots, then the spot detector would ideally find the difference between the central region of the soccer ball and its surroundings. It would see that the value of the brightness of white, which has a $YUV = (1, 0, 0)$ is different than the value of brightness of black which has a $YUV = (0, 0, 0)$.

In particular, the system uses a type of spot filter called a constant-time boxcar filter. The boxcar filter finds the difference in brightness by running two squares, one in the center and one that surrounds that square, along the image as seen in Figure 9.1 For each iteration, the filter determines the difference in average of center and surround to produce candidate bright and dark spots. This means that for each grouping of center and surround, the y-values are averaged for each square. Then these averages are subtracted to produce a difference in average. The spot filter determines that this area is a potential candidate for the ball if the difference in average is larger than a set threshold.

9.2 Spot Detector

Once the spot filter produces a different candidate, the spot detector then takes the different candidates and tries to narrow down the candidates. The goal of the spot detector is to remove noise by using peak detection to find the most likely candidates. Peak detection is the process of finding local maxima in the histogram.

The spot detector begins by taking the array of candidates and organizing the data into a histogram to perform peak detection. The spot detector first goes through looking for peaks and rejects those that are too green. Since the field is green, peaks

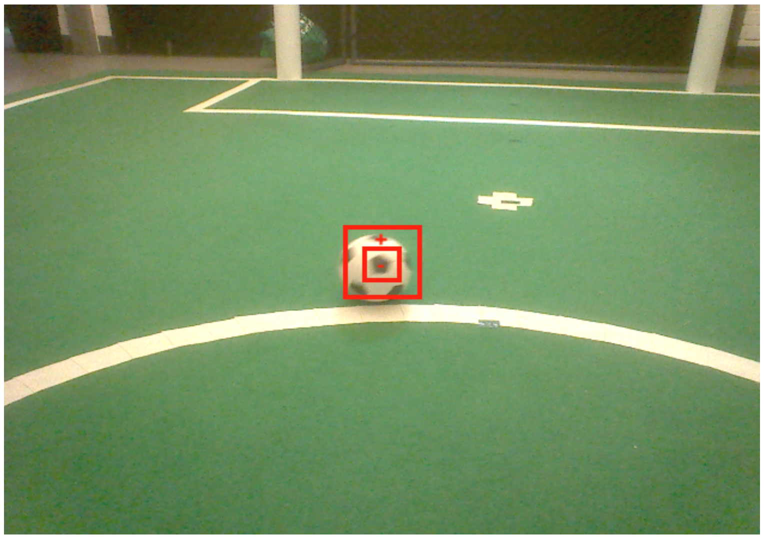

Figure 9.1: Representation of how the constant-time filter would locate ball.

that are too green indicate that the candidate is the field rather than the ball so they are therefore rejected.

Afterwards, the weaker overlapping peaks are rejected to produce only tallest peaks in a particular area. This should take $O(n^2)$, but the natural sorting of the list allows the operation to be fast.

Since the differential image is only compromised of gray-levels, the spot detector is skipped in differential mode.

10 Results

The ball detection system must be able to reliably identify the ball in multiple real-time scenarios. The algorithm's run-time and accuracy was tested on four different types of scenes:

1. Only balls moving.
2. Only robots moving.
3. Both robots and balls moving.
4. Balls moving near moving robots.

The first three are crucial for identification as they are the most common scenarios. Testing was also performed in scenes with balls moving near moving robots to challenge the system. Each case was tested on a set of 20 randomly selected images. The number of positive identifications, false negatives, and false positives were recorded for each scenario. False positives occurred due to the perspective projection from the spot detector. Since the ball is assumed to be on the ground, something else not on the ground with similar characteristics, say a robot's arm, would be considered a candidate for the ball by the spot detector.

Each image was run through quadratic thresholding with a shift of 7 ($QUAD\_7$) and quadratic thresholding with a shift of 6 ($QUAD\_6$) and linear thresholding with a slope of 2 (LIN_2) and linear thresholding with a slope of 5 (LIN_5). These were chosen as they were the most robust in the initial testing and provided an example of the tradeoffs.

		Linear 5	**Linear 2**	**Quadratic 7**	**Quadratic 6**
Only Ball	Success Rate	95%	90%	90%	95%
	Avg False Positives	0	0	0	0
Only Robot	Success Rate	20%	100%	100%	50%
	Avg False Positives	2.55	0.35	0.5	1.75
Robots and Balls	Success Rate	100%	85%	85%	90%
	Avg False Positives	1.85	0.35	0.45	0.95
Ball near Robot	Success Rate	80%	90%	80%	80%
	Avg False Positives	2.5	0.8	0.8	1.55

Table 10.1: The success rate and average number of false positives.

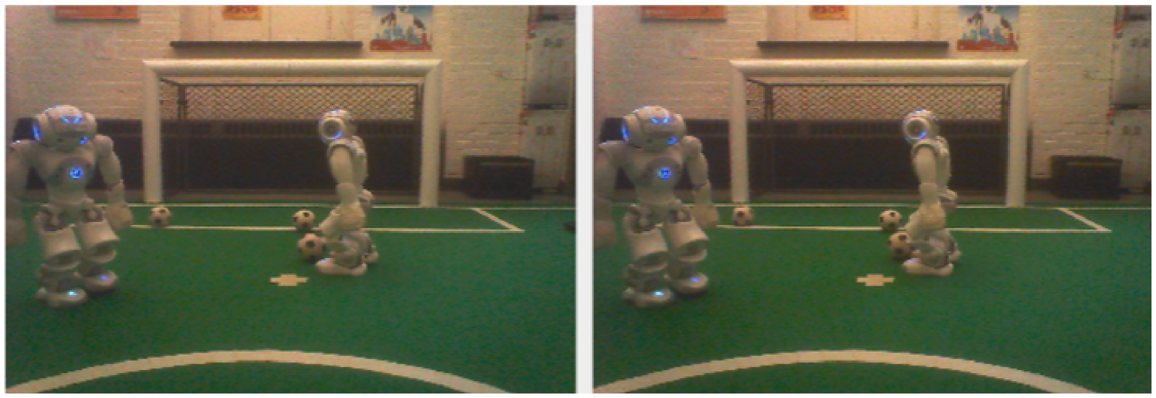

Figure 10.1: This is an example of the original frames. Everything but the ball in the middle is moving.

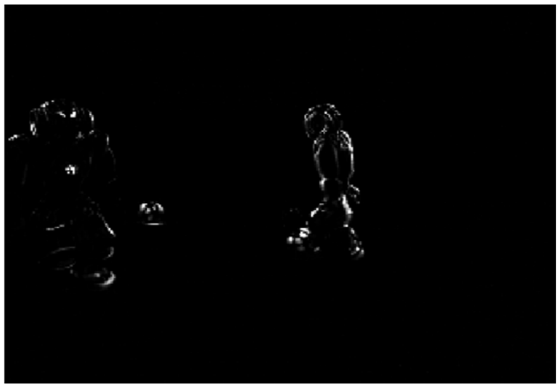

Figure 10.2: This is a sample image of the output produced from *LIN_5*.

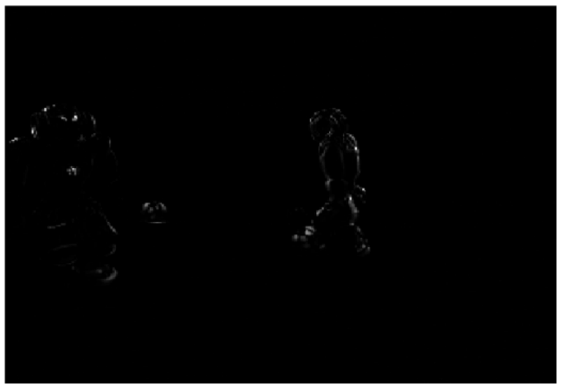

Figure 10.3: This is a sample image of the output produced from *LIN_2*.

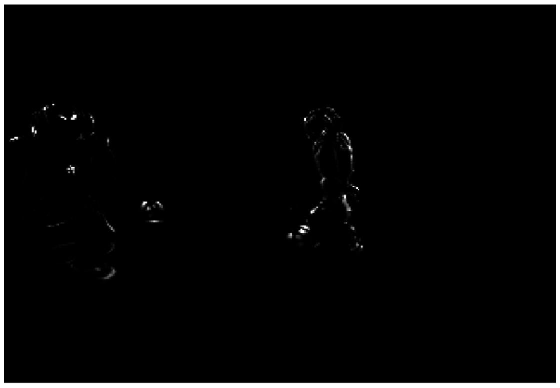

Figure 10.4: This is a sample image of the output produced from *QUAD_6*.

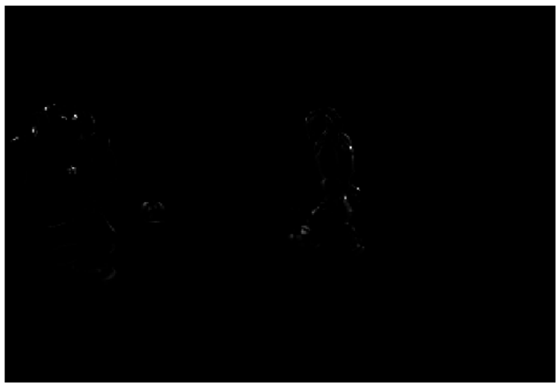

Figure 10.5: This is a sample image of the output produced from *QUAD_7*.

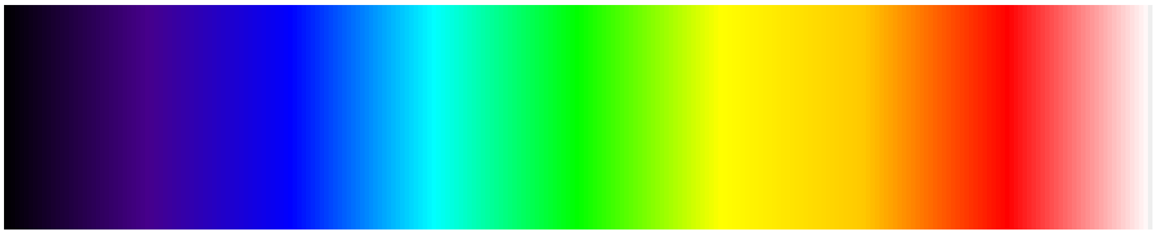

Figure 10.6: This is the heatmap used in the following images. The heatmap goes from left to right with black being the lowest pixel value and white the highest.

Figures 10.1, 10.2, 10.3, 10.5, and 10.4 are samples of the actual output from the algorithm. The colored images used in the results section are altered from gray-scale images to better illustrate the brightness levels. The heatmap used is shown in Figure 10.6.

10.1 Only Moving Balls

These cases tested the thresholding methods on the simplest scenario. The only thing on the field that was moving was the ball thus any movement must be the ball. This scenario was chosen to confirm that the algorithm could perform the most trivial classification with accuracy and robustness.

As demonstrated in Figures 10.8, 10.9, 10.11, and 10.10, the ball was successfully identified with no difficulty. There were no false identifications on the cross or field lines. The *LIN_2*'s result seen in Figure 10.9 had the smallest values compared to the others. The other method's soccer balls had a lot of pixels that were at the maximum value. These cases illustrated that *LIN_5*, *QUAD_7*, and *QUAD_6* had a lot of pixels that were at the maximum value. This could create issues when examining other cases but these bright pixels helped identification in this situation.

When looking at the overall performance demonstrated in Table 10.1, the *LIN_5* and *QUAD_6* performed minimally better than their counterparts. The different between the results came from scenarios where the ball was at a much further distance. Further, none of the thresholding methods caused false positive identifications with this scenario. This is significant as it means all of the thresholding algorithms were able to discriminate object motion from noise.

10.2 Only Moving Robots

The algorithm was tested on scenes with only moving robots to test how the different slope and shift values affect false positives. The only objects moving are the two robots in example 10.12. Since one robot is further than the other, their brightness values are lower. Thus, this scene is a great example to test how the thresholding methods do with robots at varying differences.

With the only moving robots case, a success is defined as a lack of identification. The scene was considered a failure even if there was only one false positive. Both *LIN_2* and *QUAD_7* successfully do not identify the robots as soccer balls as seen in Figures 10.14 and 10.16. *LIN_5* and *QUAD_6* incorrectly identified the robots as candidates to be the ball; however, *QUAD_6* had fewer false positives. The hypothesis for this is that *LIN_5* and *QUAD_6* cause too many pixels to be the maximum value and do not create a distinction between robots and balls. Making many non-ball values the maximum value eliminates the first theory of operation. If many moving pixels are the highest absolute difference, the balls cannot have a higher absolute difference.

These results create further insight into the effects of different shifts and slopes. With linear slopes, these results would suggest that lower-valued slope correspond to fewer candidates. Since the graph of a linear threshold with a steeper slope is more vertical, a linear threshold with a steep slope is more similar to a binary threshold than its shallower sloped counterpart. Thus, the steeper slope is creating a greater distinction between smaller and larger values of Y. This is in line with the theory of operation as higher absolute differences are more likely to be the ball, so the lower differences can be discarded. The same idea applies to quadratic shifts. A larger shift should correspond to lower pixels becoming smaller than they would be with a smaller shift. Thus, this also is in line with the same theory of operation that higher valued pixels are more likely to be the ball. Since the theory of operation holds, there is a tradeoff between throwing away more false positives and having a higher probability of finding the ball. The hypothesis is that the increase in false positives is due to the increase in maximum values. This is particularly visible when looking at the success rates and average number of false positives in Table 10.1. In the 20 different scenes presented, *LIN_2* and *QUAD_7* did not have any false positives. This implies that these methods are good at not incorrectly identifying robots as balls. However, *LIN_5* and *QUAD_6* had very few instances where it was able to successfully not identify a robot as a ball.

10.3 Moving Robots, Moving Balls and Moving Robots near Moving Balls

These scenarios test whether or not the algorithm can discriminate balls from robots when they are in the same image. It tests both theories of operation. If a moving ball has a spatial density then thresholding should attenuate the wrong pixels, so the spot filter can discriminate the robots from the ball. If higher absolute differences are more likely to be a ball, then the ball should have a higher absolute difference than the robots.

10.3.1 Moving Robots and Moving Balls

LIN_5 was successful in finding the soccer ball, but it found many false positives. As shown in example 10.18, the ball was successfully identified in spot 4, but there were four other false positives. This method was able to find all the balls in the 20 different examples of moving robots in the same frame as a moving ball. As seen in Table 10.1, this method was able to find all the balls but it produced an average of 1.85 false positives per image. Thus, this method did the poorest job at discriminating balls from robots even though it was more likely to find the ball. This is believed to be due to the overall increase in maximum values. This increase does not create a distinction between the robot and ball as all the pixels are increasing, not just the balls. Thus, the higher absolute difference might not correspond to a ball in this scenario.

QUAD_6 was more successful as it had less false positives, but it did not find the ball as frequently. Even though the success rate went down, it did only average 0.95 false positives per image as seen in Table 10.1. This is concurrent with Figure 10.20 as the number of false positives is half of *LIN_5*.

QUAD_7 has less false positives than the previous two cases. It was more successful at discriminating the robots from the balls. When looking at Figure 10.21, the difference in magnitude is more consistent to what is expected from the second theory of operation. The result of 20 tests had a very low average of false positives since this result was closer to our theory of operation seen in Table 10.1. However, this resulted in a lower success rate.

The final method, *LIN_2*'s result was closest to what was expected from the theories of operation. Even though this case did not have as many bright pixels as the other cases, it did a better job at distinguishing the ball from the robot. Also, it has the biggest distinction between robot and ball pixels, which follows the second theory of operation: higher absolute differences are more likely to be a moving ball. Figure 10.19 has the highest concentration of absolute difference in brightness at the ball spots compared to the surrounding figures. The results of *LIN_2* were most in line with the theories of operation.

10.3.2 Moving Robots near Moving Balls

The distinction between the previous case and this case was made because scenes with a moving ball next to a robot are inherently harder for the ball detection system. It further tests how well the system discriminates robots from balls as they are directly next to each other.

Like in the previous case, *LIN_2*'s result is closest to what is expected from the theories of operation. It did not exceed the maximum pixel value and it separated the robot pixels from the ball pixels. Also, the tradeoff between false positive discrimination and shift/thresholding number is present in this case. This case, however, does have *LIN_2* and *QUAD_7* having false positives.

This scenario particularly demonstrated the differences between *LIN_2* and *QUAD_7* with regards to the success rate. *LIN_2* was more successful than any of the other methods. It had the highest success rate and the smallest average number of false positives as seen in Table 10.1.

10.4 Run-Time Analysis

In general, the total run-time is dominated by either the memory access time or the computations. The algorithm is broken down into three parts for the run-time analysis: taking the differential image calculation, thresholding, and running the image through the spot filter.

Currently, the absolute difference code is not optimized nor is adapted to run on the robot. The results of this algorithm run on a computer with a 2.2 GHz processor is in Table 10.2. Both the spot filter and absolute differential image were run 30 times on various images from different scenes. Even though the 2.2 GHz processor is faster than the robot's 1.6GHz processor, an estimation of the optimized run-time can be obtained.

There are seven basic operations needed to create the absolute differential image:

Algorithm 3 The machine instructions to compute the absolute difference for two pixels

```
1: procedure ABSOLUTE DIFFERENTIAL IMAGE CALCULATION FOR ONE PIXEL
2:     Fetch pixels from first source image
3:     Fetch pixels from second source image
4:     Subtract pixels from second source image
5:     Absolute value
6:     Store pixels to destination image
7:     Decrement loop counter
8:     if counter > 0 then
9:         jump back to beginning of loop
```

The fetch, subtract, absolute value, and store can be processed in parallel. The robot's CPU can run 8 16-bit pixels in parallel. Six operations can be done at once thus the algorithm does $\frac{6}{8} = 0.75$ instructions per pixel. Since the robot has a 1.6GHz CPU, it can do 1.6 billion instructions per second. If the calculation roughly estimates to one clock tick per instruction then for the 640x480 image with roughly 300,000 pixels, the expected computational time is 144 microseconds.

	Average Running Time	**Standard Error**
Differential Image	213.73μs	2.54
Spot Detector	334.76μs	9.93

Table 10.2: The average CPU running time and standard error.

In comparison, Table 10.2 demonstrates the computational time difference between the spot detector and absolute image differential. Since the absolute image differential method is much simpler than the already-optimized spot detector, the running time is drastically quicker even before the algorithm and implementation are optimized.

However, memory access time needs to also be accounted for to find the total time. This algorithm will get excellent cache performance and be running at close to the full memory bandwidth of the robot since the memory accesses are sequential. There is a total of three memory accesses that occur: reading the pixel from the first source image,reading the pixel from the second source image, and writing to the destination. Therefore, the amount of memory that needs to be transferred is (640x480) $*$ 2 bytes $*$ 3 operations $= 1,843,200$ bytes. Assuming that the robot's memory bandwidth is $330\frac{MB}{sec}$ then this algorithm would need 6 milliseconds. Further improvements can be made by only considering those pixels below the horizon and reusing memory access from other parts of the vision algorithm.

Since 144 microseconds is less than 6 milliseconds, the total time is the memory access time. This is a reasonable runtime for this target platform, as 6 milliseconds gives plenty of buffer time for the other necessary operations.

10.5 Summary

When selecting between linear and quadratic threshold methods, there is a tradeoff between the number of false positives and successful identifications. In the linear threshold method, the slope value is inversely proportional to the number of false positives and is proportional to the number of successful identification. In the quadratic threshold method, the shift value is proportional to the number of successful identification and is reversely proportional to the number of false positives. However, there is a threshold such that the number of successful identifications drastically decreases once the shift or slope passes with each of these methods.

LIN_2 was best at following the theories of operation. It did not push the pixel values beyond the pixel limit, which allowed it to distinguish the robot from ball pixels This is why it had the higher success rates with the lowest average false positives per image. The result shows promise, but there is still not an acceptable success rate. Thus, a third theory of operation would be needed to improve this ratio.

This algorithm was able to perform at the speeds needed for deployment in embedded platforms. The simplicity of the algorithm allowed for an efficient runtime.

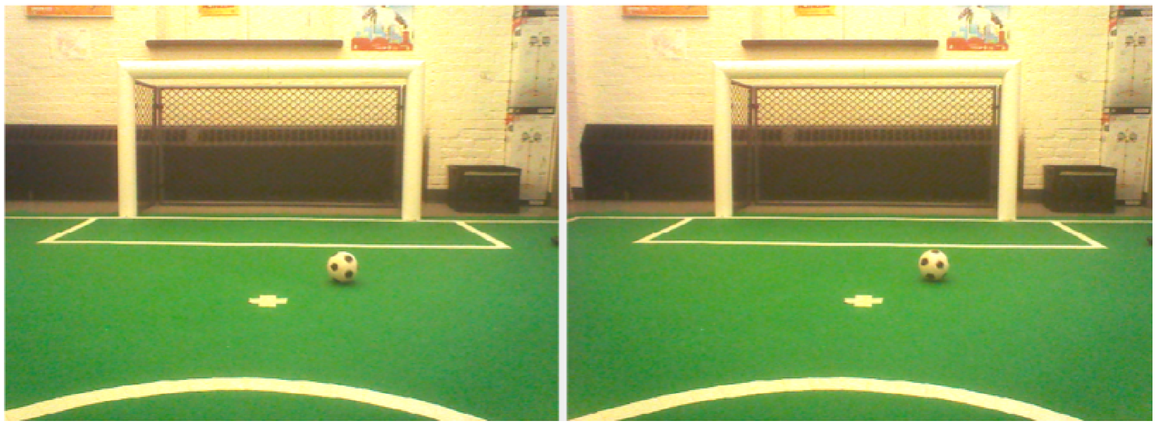

Figure 10.7: This is an example of a sequential scene used to test the algorithm on scenes with only moving balls. Note that only the ball is moving.

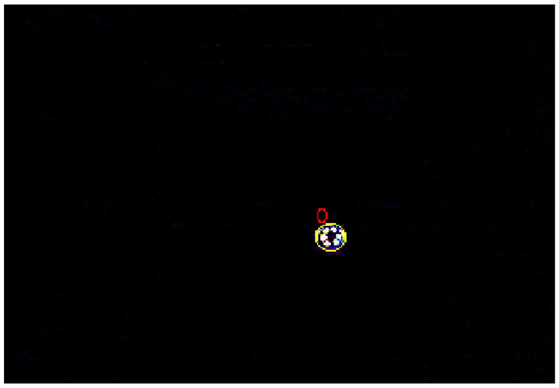

Figure 10.8: This is the output of *LIN_5*.

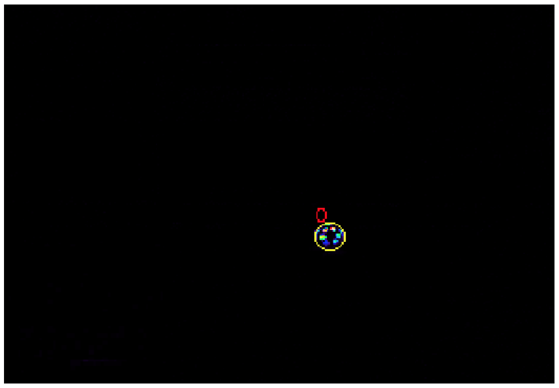

Figure 10.9: This is the output of *LIN_2*.

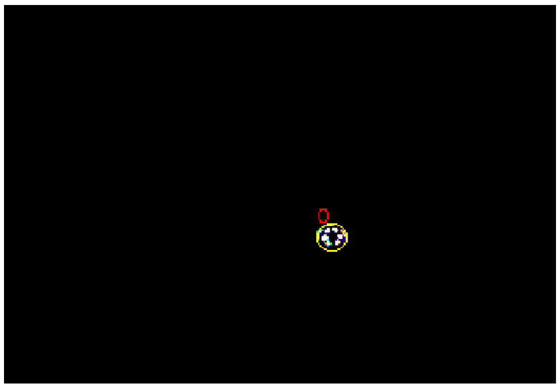

Figure 10.10: This is the output of *QUAD_6*.

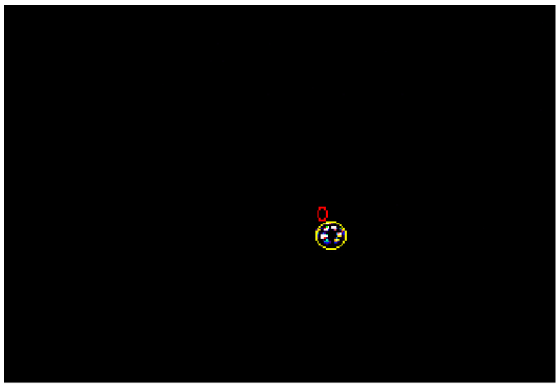

Figure 10.11: This is the output of *QUAD_7*.

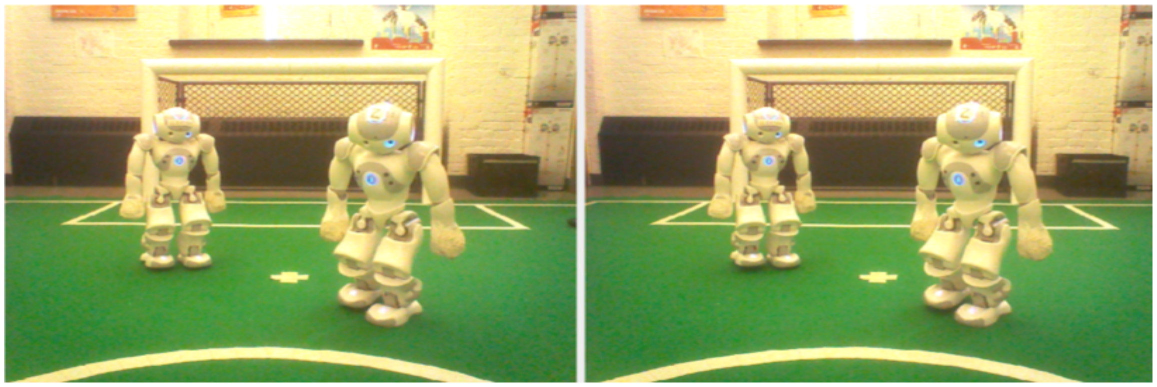
Figure 10.12: This is an example of a sequential scene with only moving robots. Note that both robots are moving.

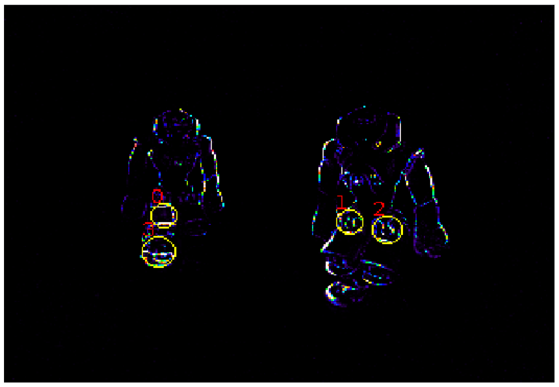

Figure 10.13: This is the output of *LIN_5*. The algorithm unsuccessfully identified the moving robots as balls.

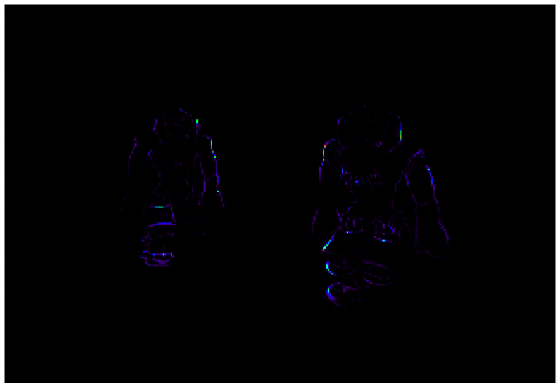
Figure 10.14: This is the output of *LIN_2*. The algorithm did not successfully discriminate moving robots from moving balls.

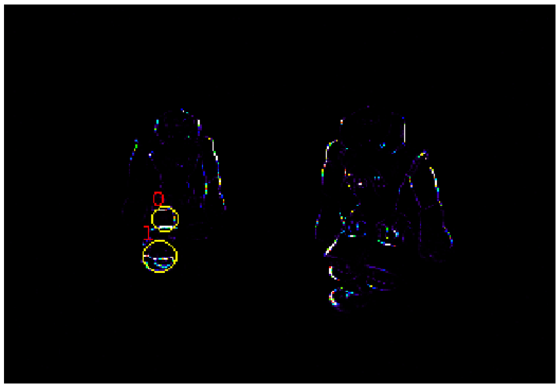

Figure 10.15: This is the output of *QUAD_6*. The algorithm did not successfully discriminate moving robots from moving balls.

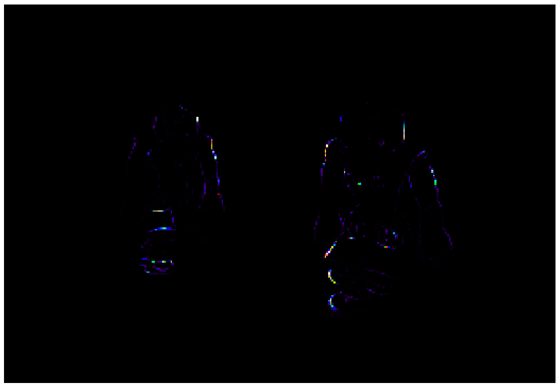
Figure 10.16: This is the output of *QUAD_7*. The algorithm successfully did not identify the moving robots as balls.

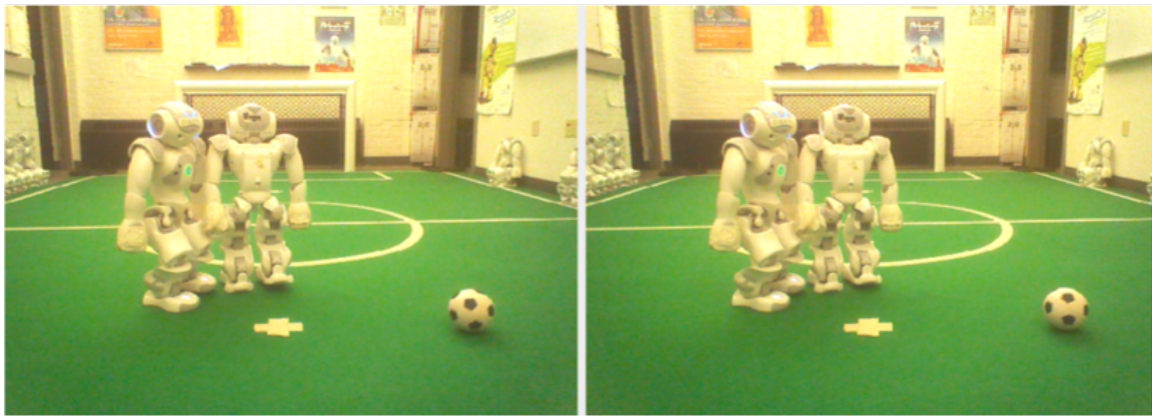

Figure 10.17: This is an example of a sequential scene with both robots and balls moving.

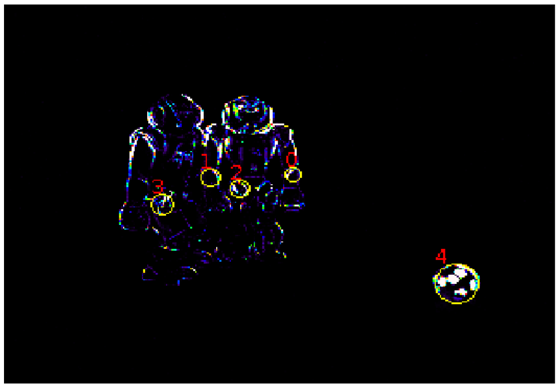

Figure 10.18: This is the output of *LIN_5*.

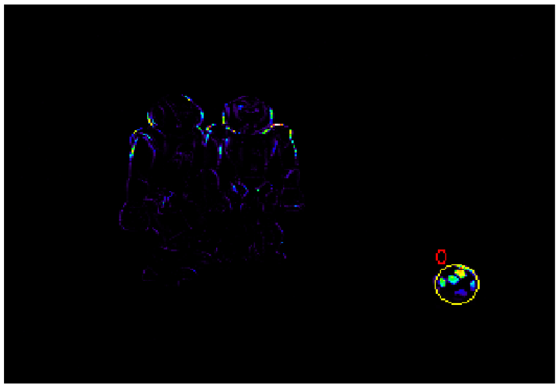

Figure 10.19: This is the output of *LIN_2*.

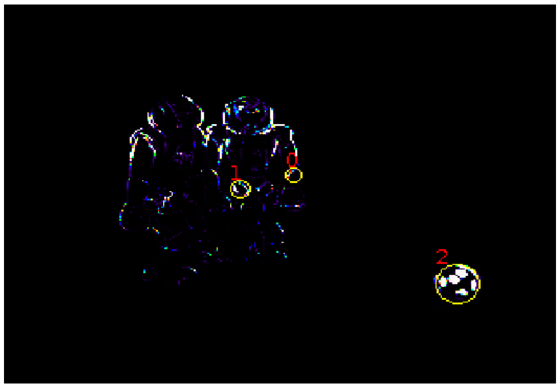

Figure 10.20: This is the output of *QUAD_6*.

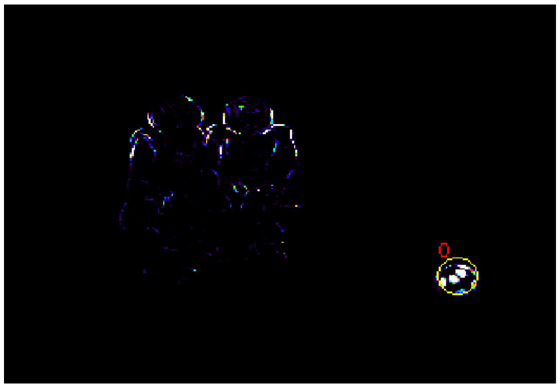

Figure 10.21: This is the output of *QUAD_7*.

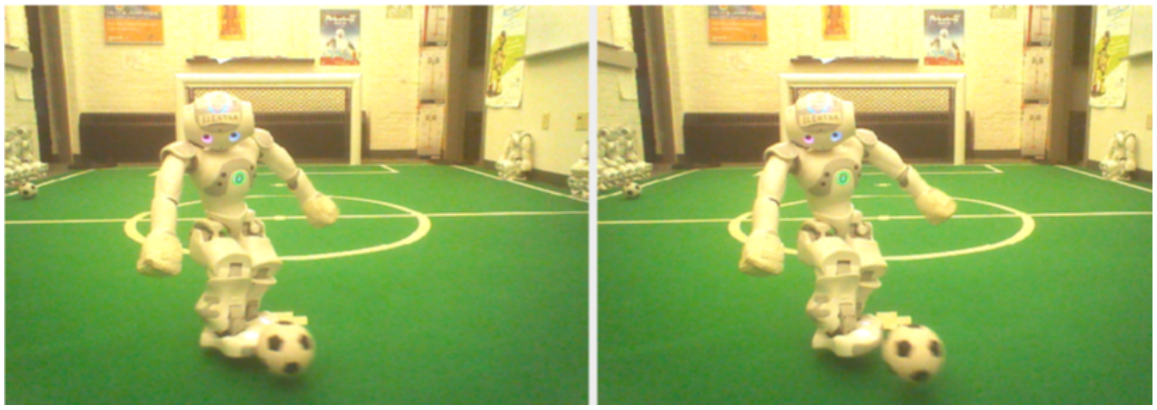

Figure 10.22: This is an example of a sequential scene with a moving ball next to a moving robot.

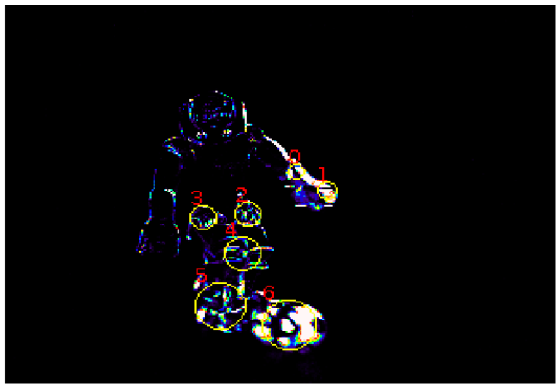

Figure 10.23: This is the output of *LIN_5*. The algorithm found the soccer ball in spot #6.

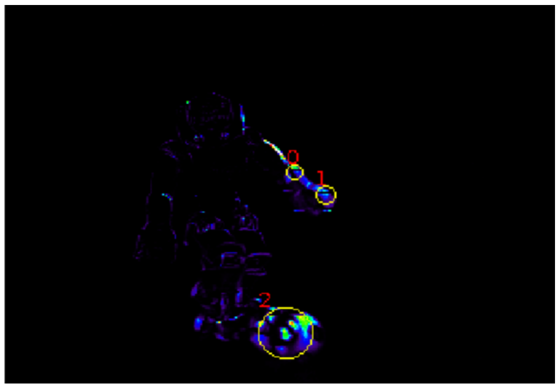

Figure 10.24: This is the output of *LIN_2*.

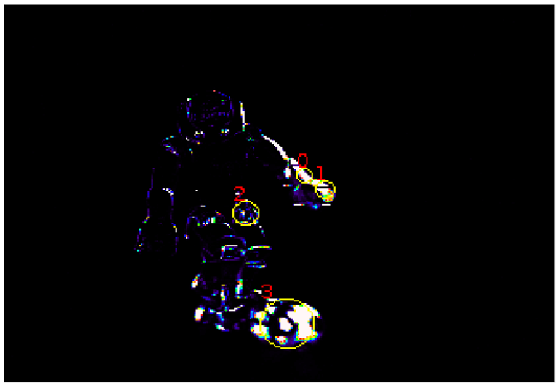

Figure 10.25: This is the output of *QUAD_6*. The algorithm found the soccer ball in spot #3.

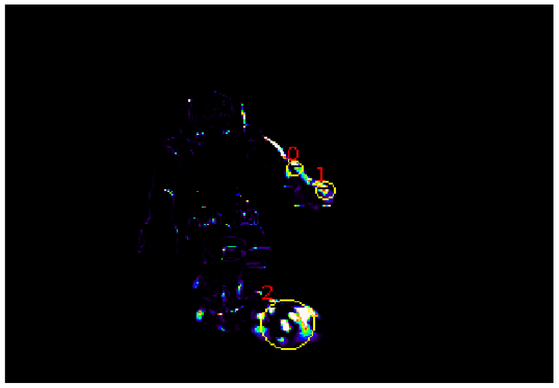

Figure 10.26: This is the output of *QUAD_7*.

11 Future Work

As demonstrated, the theories of operations do a competent job discriminating robots from balls. Further work can be done to improve run-time, accuracy and robustness. First, the algorithm needs to be optimized before robotic implantation. Second, the addition of a third theory of operation could further ameliorate discrimination with this method. Finally, further work could be done to eliminate false positives created from camera motion.

11.1 Run-Time Improvements

As discussed in Section (real-time), this new algorithm is efficient with significant room for more optimization. Additional work is necessary before utilization to provide sufficient processing time for other required visual operations. Converting the algorithm from C++ to assembly would gain significant processing time that approaches parameters required for implantation.

11.2 History of the Ball

Currently, the algorithm treats a pair of images as mutual exclusive from another pair of images. This method works well as it allows each scene to be analyzed without any negative influence by another scene. However, there are further improvements that need to be done to make the algorithm more robust. A third theory of operation could be a ball's previous location to provide information about its current location. Given that the ball does not often jump in a relatively short period of time under one second, taking advantage of previous knowledge regarding the ball's location could remove some false positives and speed up the run-time.

If the algorithm found the ball in the previous differential image, the runtime could be improved by beginning the search in the area the ball was previously found. If there is previous information regarding the ball's location, the algorithm could be the search in current image in that area as demonstrated in Figure 11.1. If a candidate is found in that region, then the algorithm would not have to continue. This would drastically speed up the search, as the algorithm would not have to go through the entire image each time.

This theory of operation would also reduce false positives. Using the history of the ball as a filter could catch cases where the robots movements create false positives. In Figures 10.26 and 10.24, the false positive is created by the bright lighting on the robot's arm thus the false positive only appeared when the robot raised its arm. Knowing that the ball was in the lower half of the field would remove this false

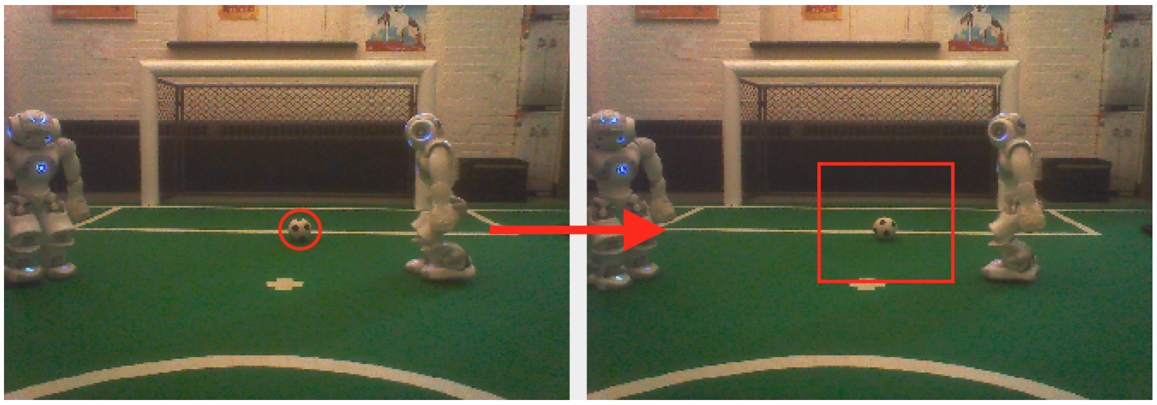

Figure 11.1: If the ball was found in the previous image on the left, then the algorithm could begin the search in a smaller region of the image as demonstrated by the red box on the right image.

positive, as the algorithm would not look in this part of the field unless it could not find a ball in the lower half. Therefore, the ball would be found first and that false positive would be excluded.

This method would also have additional benefits outside of ball recognition. It would provide the behavior system with a way to track the ball. Tracking the ball could provide useful information to predict and anticipate future ball movements. For example, if the robot last found the ball on the left side of the image, it could pan to the left to find the ball. This could help as it helps the robot find a lost ball much quicker.

11.3 Camera movement

A robot searches for the ball by walking around and panning its head. This causes the robot's camera to move, which makes the images blurrier. The initial assumption made in this paper was that the camera is not moving. In reality, the robot is walking around and panning its head; thus the camera does move in the games. The algorithm needs to take into account that the robot's head can be moving. The algorithm described in this paper currently finds false positives in the lines when given a sequence of images where the camera is moving. When examining Figure 11.3, the change in line location causes it to look like two white lines with a dark interior. This shares unfortunate characteristics with the ball, which also white with a dark interior. The lines are viewed as possible ball candidates.

One possible method for removing false positives on field lines is to use the line detector that is already implemented. The algorithm could run the line detector on both images and then remove any candidate that falls on the line. This method would not add any additional overhead since the field line detector is already run on each image. However, it would not be perfect solution, as it would discard any balls on the field lines.

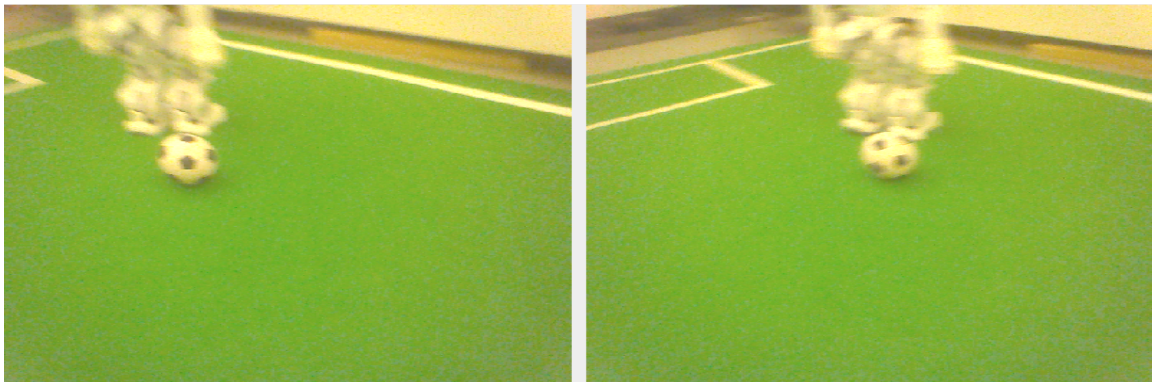

Figure 11.2: This is an example of a sequential scene with the camera moving (robot was walking), a moving ball next to a moving robot.

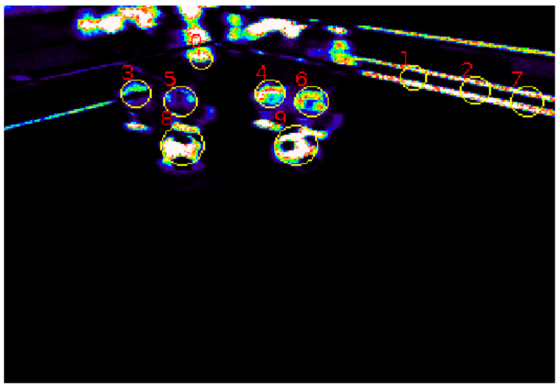

Figure 11.3: This is the absolute differential image with spots. Note that there are false positives on the field lines.

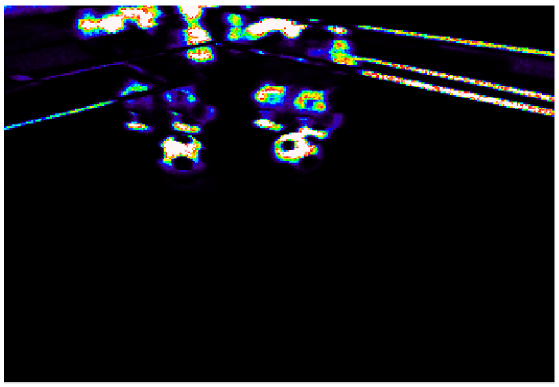

Figure 11.4: This is the absolute differential image with spots. Note the brightness of the ball.

12 Conclusion

Current methods of ball detection find the ball with reasonable accuracy, but are still flawed. The time constraints of the platform and general complexity of the problem has prevented these object recognition algorithms from performing as well as a human toddler. Using human vision as inspiration, this paper explored and tested a theory that motion could be a viable method for ball detection. It used two theories of operations to perform ball recognition: higher absolute differences of brightness values are more likely to be a moving ball and a moving ball has very high spatial density in the image. This paper discovered that there is a tradeoff between throwing away more false positives and having a higher probability of finding the ball. It also discovered that *LIN_2* followed the theories of operation closest. The other thresholding methods tested pushed the pixel values beyond the limit, which eliminated the benefits of the first theory of operation. While it did well, *LIN_2*'s accuracy was not quite accurate enough for effective performance in an actual RoboCup game. Thus, a third theory of operation should be introduced to improve accuracy. Even without a third theory of operations, this method improved reliability and provides a foundation for future sequential algorithms. Overall, this paper concludes that incorporating motion detection is a viable algorithm option in constrained platforms and could improve object recognition.

Bibliography

[1] Gilles Aubert, Rachid Deriche, and Pierre Kornprobst. Computing optical flow via variational techniques. *SIAM Journal on Applied Mathematics*, 60(1):156–182, 1999. `http://slipguru.disi.unige.it/readinggroup/papers_vis/aubert99flow.pdf`.

[2] Domenico Bloisi, Francesco Del Duchetto, Tiziano Manoni, and Vincenzo Suriani. Machine learning for realistic ball detection in robocup spl. 2017. `https://arxiv.org/pdf/1707.03628.pdf`.

[3] Eric Chown, William Silver, and Konstantine Mushegian. Visual soccer ball detection in robocup spl. 2016.

[4] Jason Corso. Motion and optical flow. `https://web.eecs.umich.edu/~jjcorso/t/598F14/files/lecture_1015_motion.pdf`, 2014.

[5] Nicolás Cruz, Kenzo Lobos-Tsunekawa, and Javier Ruiz-del Solar. Using convolutional neural networks in robots with limited computational resources: Detecting nao robots while playing soccer. *arXiv preprint arXiv:1706.06702*, 2017.

[6] James J DiCarlo, Davide Zoccolan, and Nicole C Rust. How does the brain solve visual object recognition? *Neuron*, 73(3):415–434, 2012. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3306444/#R68`.

[7] Aldebaran Documentation. Motherboard. `http://doc.aldebaran.com/2-1/family/robots/motherboard_robot.html`, 2011.

[8] David Fleet and Yair Weiss. Optical flow estimation. In *Handbook of mathematical models in computer vision*, pages 237–257. Springer, 2006. `http://www.cs.toronto.edu/~fleet/research/Papers/flowChapter05.pdf`.

[9] Dipl-Inf Andreas Fürtig, Jonathan Cyriax Brast, Sina Ditzel, Hans-Joachim Hammer, Timm Hess, Kyle Rinfreschi, Jens-Michael Siegl, Stefanie Steiner, Felix Weiglhofer, and Philipp Wörner. Team description for robocup 2017, 2017.

[10] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. `http://image.diku.dk/imagecanon/material/HornSchunckOptical_Flow.pdf`.

[11] K Lobos, G Azócar, N Cruz, R Pérez, P Miranda, F Leiva, C Celemın, and J Ruiz-del Solar. Uchile nao ball perceptor. 2016. `https://github.com/uchile-robotics/nao-ball-perceptor-2016/wiki#algorithm-in-a-nutshell`.

[12] K Lobos, G Azócar, N Cruz, R Pérez, P Miranda, F Leiva, C Celemın, and J Ruiz-del Solar. Uchile robotics team team description for robocup 2017. 2017. https://www.robocup2017.org/file/symposium/soccer_std_plf/Team_Description_Paper_2017.pdf.

[13] Jacob Menashe, Katie Genter Josh Kelle, Josiah Hanna, Elad Liebman, Sanmit Narvekar, Ruohan Zhang, and Peter Stone. Fast and precise black and white ball detection for robocup soccer. 2017. http://www.cs.utexas.edu/users/pstone/Papers/bib2html-links/LNAI17-jmenashe.pdf.

[14] Simon O'Keeffe and Rudi Villing. A benchmark data set and evaluation of deep learning architectures for ball detection in the robocup spl. 2017.

[15] Nicolas Pinto, David D Cox, and James J DiCarlo. Why is real-world visual object recognition hard? *PLoS computational biology*, 4(1):e27, 2008. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2211529/.

[16] Thomas Rofer, Tim Laue, Jesse Richter-Klug, and Felix Thielke. B-human team description for robocup 2016. http://www.robocup2016.org/media/symposium/Team-Description-Papers/StandardPlatform/RoboCup_2016_SPL_TDP_B-Human.pdf, 2016.

[17] Sumedha Singla. Using optical flow to find direction of motion. http://www.cs.utah.edu/~ssingla/CV/Project/OpticalFlow.html, 2015.

[18] Ashit Talukder, S Goldberg, Larry Matthies, and Adnan Ansar. Real-time detection of moving objects in a dynamic scene from moving robotic vehicles. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1308–1313. IEEE, 2003.

[19] Dave Wilson. Uyvy yuv pixel format. http://www.fourcc.org/pixel-format/yuv-uyvy/, 2011.