MCGILL UNIVERSITY

Multi-Sensor Robot and Vision

Design Project 2

Christopher McGirr 260399623

Yulric Sequeira 260404113

Design Project Supervisor: Professor Joseph Vybihal

Design Project Group 15

11/4/2014

Table of Contents

List of Tables

List of Figures

1. Abstract

The main goal of this semester's design project was to implement a simple motion detection algorithm that can detection areas of motion and track where that motion is so that the robot may avoid that motion. The challenges that we faced were the speed of the computation of our algorithm.

We attempted to offload the image processing over Bluetooth to another computer, but found the data throughput too small for any real-time application of our algorithm. Then we develop an optimized version of our algorithm in C and minimized the computation time by eliminating the number of pixels the difference filter has to check in the image and the complexity of the motion quadrant detection algorithm.

What we have achieved is a reasonably quick algorithm that can react to movement, albeit with a delayed response. Though the range of our motion detection algorithm far exceeds what the onboard ultrasonic sensors are capable of which is an advantage of our algorithm.

2. Introduction

The human brain is a computationally powerful machine that is able to process visual data on the fly with the ability to detect patterns and objects. The hope is to one day translate this ability to digital systems to grant autonomous robotic systems more environmental data for decision making. Robotic vision can be applied to a number of situations such as space exploration for when a human control of a rover is not possible or for navigation and control of machines used today. The possibilities for this application is endless.

The challenge becomes implementing vision algorithms on embedded systems that are portable, lightweight, energy efficient and cost effective. In our second semester of the design project we have tasked ourselves to implement our simple motion detection algorithm on an embedded computer of a two-track robot. The goal is for the robot to detect motion of objects so that is can avoid these objects that are in its path. The difficulty is that the embedded processor of the robot has limited computational power compared to a consumer personal computer and so two options are left for how our algorithm will be implemented. First, is to use the existing Bluetooth communication on the robot to stream the webcam images off the robot for processing and have the results sent back to the robot. The challenge is to have enough images sent per second and processed in a short amount of time for the result to be relevant to

the robot's current situation. Secondly, an optimized motion detection algorithm can be created to be run on the robot taking into account the robot's limited processing power.

3. Background

The motion detection algorithm that we have created must run on a small embedded computer. The PhidgetSBC2 is the board that we will be using for running our algorithm. The board is a Linux based system running a lightweight version of Debian customized for ARM processors. The ARM processor is set to a maximum clock speed of 400MHz which compared to modern personal computers is quite slow. The available memory of the board is 64MB SDRAM for running applications and 512MB Flash Memory for storage.

This limits the complexity of our algorithm to be run on the machine. The basic idea behind our motion detection algorithm is to compare two consecutive frames from a webcam pixel by pixel. To simplify and reduce computation further the algorithm will be run using grayscale images rather than a colour image. Rather than representing each pixel as three 8bit values of Red, Green, Blue we will use only one 8bit value representing the intensity. This helps to reduce image size and computation time.

Once we have the difference of two consecutive frames we can begin searching this difference for regions of change and encapsulate these regions as an area of motion. With all the regions of motion we can begin tracking the motion as it moves in the field of view of the robot. This information can be used by the robot for decision making when finding a path for example through a room. The algorithm may be simple, but the difficulty comes when implementing the algorithm so that it can function in real time. Another option is to off load the image processing to another computer using the robot's Bluetooth connection.

The Bluetooth communication will be implemented using the Iogear Bluetooth 4.0 USB Micro Adapter. The advantage of using a Bluetooth USB adapter is that the low level protocols for handling communication will be handle by underlying drivers. The driver which will be used on the robot is the BlueZ kernel modules [1]. The driver supports multiple communication protocols such as RFCOMM, L2CAP, SCO and many more. The protocol which we will be using for our project is the RFCOMM. This protocol provides reliability in communication much in the same way TCP handles communication with the use of handshaking and verification that a message has been sent. It is very much a synchronous communication protocol. This ensures the data is sent and received. If a piece of data is not received in a limited time, the protocol will terminate the connection and return an error. The only major difference RFCOMM has with

TCP is that the ports are limited to 30 rather than 65535 [2]. However, this is not a major concern for us as we will be using a single connection between a computer and the robot.

Once we have the Bluetooth connection established the next challenge is to retrieve an image from the webcam. Unlike, the previous semester we will be utilizing a different image processing library that can handle image capture from a webcam. OpenCV or Open Source Computer Vision is a C/C++ library that handles image processing [3]. Now this library is equipped with many functions that can interpret and analysis images and video. However, for this project we will focus on the functions that handle the image capture. The OpenCV library uses V4L2 to communicate with the webcam which is a generic driver used in most Linux systems to handle cameras. This eliminates coding we would have to accomplish on our part handling the image capture protocol. In addition, OpenCV provides structures for containing the image that provide image meta data such as the size, height, width and pointer to that data in memory which will allow us to easily access the image data. With all of these components we can now begin the implementation of our design.

4. Design and Implementation

Bluetooth Image Transfer

The image transfer over Bluetooth is accomplished using two applications on two separate systems. One is the robot and the other is a Linux computer that will compile and display the transferred image. The difficulty is creating an algorithm that efficient sends the image data over Bluetooth in a short amount of time.

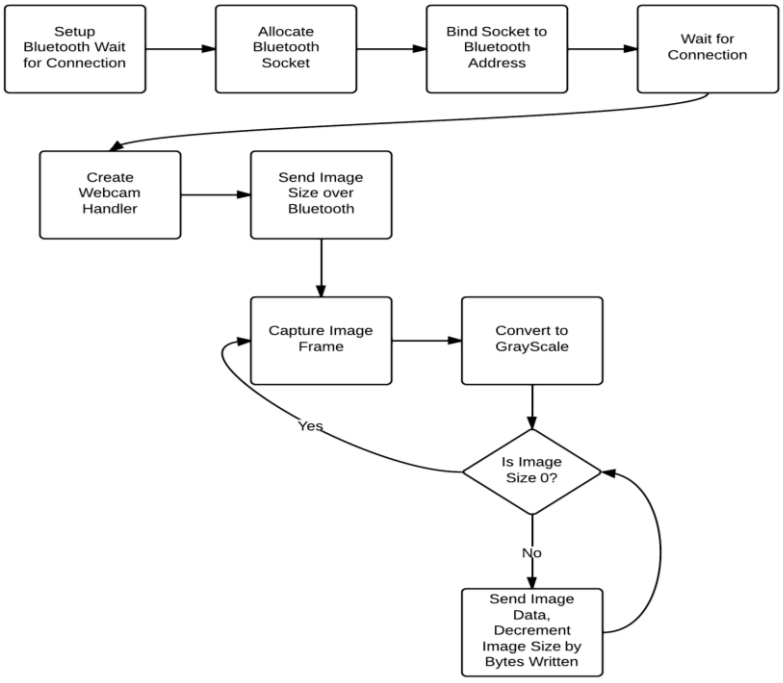

Figure 1: Bluetooth Communication on Robot

Figure 1 demonstrates how the robot will be sending the image data over Bluetooth. First we create a Bluetooth socket and wait for a connection from a computer before continuing. Once the connection has been established we capture a frame from the webcam and convert the frame to grayscale to reduce the image size. Once that is done we send the image over Bluetooth using socket commands. We continue writing to the socket until the all the data has been sent. The socket command for the write function returns the number of bytes written so this value is used to keep track of how much of the image is written.

A similar process is done on the PC side of the code. Figure 2 demonstrates the program flow of execution. First we establish the Bluetooth socket and then we attempt to connect to the robot. Note the robot will have to be waiting for a connection or else the code will throw an exception and terminate. Once a connection has been established we receive the image size and step width of the image to be used for debug to make sure the correct image size is being sent. Then we begin receiving the image itself by reading as much from the socket as possible and decrementing the image size until we reach zero meaning the transfer is complete. With the data stored in memory we pass the pointer to the blank image that was created earlier and change that images pointer to the recent image data. We minimized the handling of the imaged data by only dealing with the image's pointer rather than transferring the data to the blank image once we received it.

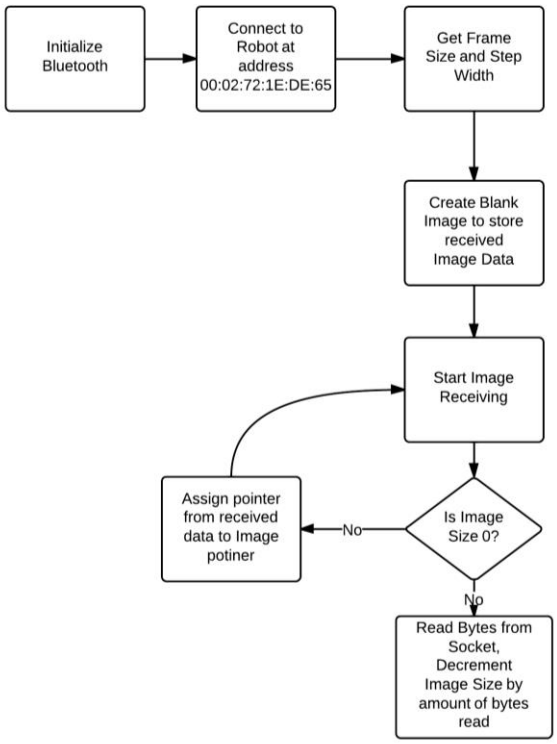

Figure 2: PC Bluetooth Image Transfer Diagram

The server and client code were both written in C to maximize efficiency when running on both the robot and the PC. The use of generic sockets to handle the Bluetooth communication means that the code is relatively simple as we simply push the data to the socket and the communication protocol handles breaking the data into packets before being sent.

Optimized Motion Detection Algorithm

The choice to switch programming languages from Java to C this semesters comes from the fact that the detection algorithm will likely have to run quickly and efficiently on the PC or robot. Unlike last semester were Java was used due to its ease of use, this semester our detection algorithm will be written in C to allow for more efficient use of memory and processor power with less of the overhead that accompanies the Java Programming language.

The algorithm that was created in the previous semester was then simply translated into C with some minor modifications to speed up the computation time of the algorithm. First we changed the way in which the image is stored. Rather than have to deal with the hundreds of thousands of pixels that result from a standard 640x480 image we will break the image into major quadrants rather than individual pixels. For this project the image size was essentially reduced to 40x30 quadrants with each quadrant representing a 16x16 square of pixels from the original

image. This change affected the way in which our difference filter operated. The difference filter is responsible for calculating the difference between two consecutive frames. If a pixel from one frame to another has change intensity outside of a certain threshold we indicate that pixel as motion or the color white as shown in Figure 3. Now we had to modify our difference algorithm to scan the image in 16x16 quadrants. We set a quadrant high, meaning motion has occurred, only if half of the total number of pixels within that quadrant changed intensity within some threshold. This method of difference filtering helps to reduce much of the noise of an image as one image's intensity changes from another.

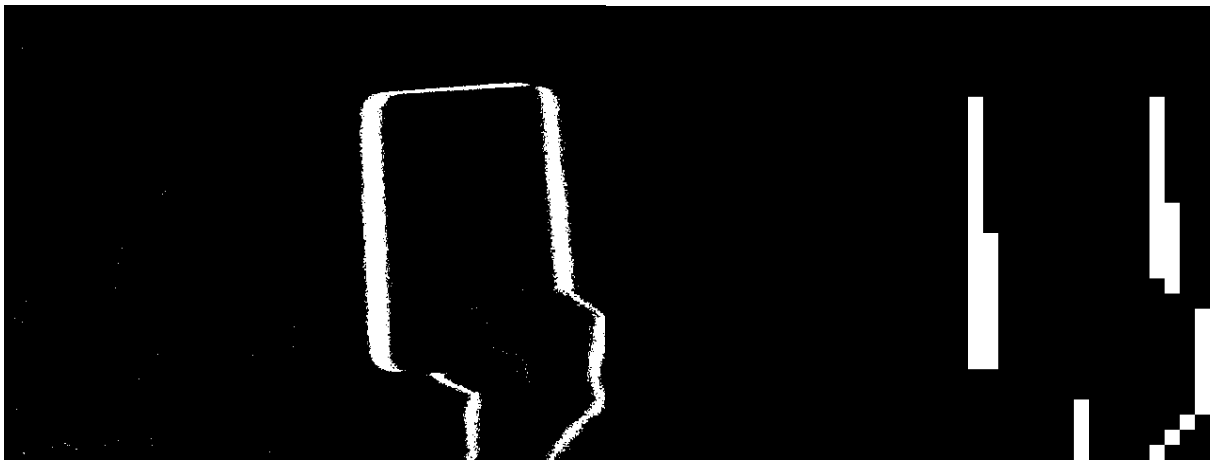

Figure 3: Comparing Raw Difference Filter with Threshold (LEFT) and Optimized Difference Filter (RIGHT)

As Figure 3 demonstrates the optimized filter does remove a lot of the noise between frames. However, it does simplify the image quite a bit, but this will help when motion detection algorithm is implemented as it will not have to search 640x480 pixels rather 40x30 quadrants for regions of motion reducing the memory footprint.

Now that we have a simplified difference of two consecutive frames demonstrating motion we can begin encapsulating these regions of motions so that we can track them. Now our motion quadrant algorithm searches the image for large regions of motion indicated by white quadrants as shown in Figure 3. Figure 5 demonstrates the logical flow of our motion detection algorithm. The algorithm finds areas of motion and encapsulates these regions using a path finding algorithm and stores the X, Y coordinate as well as the width and height of this region in a structure. Now unlike Java, C does not have dynamic arrays in the standard library and to reduce computation and memory a standard size of 20 motion regions are stored at any given time. Of course an image may have more than twenty regions of motion so only the largest of the regions are stored.

The basic idea behind the motion detection algorithm is that the algorithm scans the image, in this case the optimized difference image of 40x30 quadrants, and checks to see if these quadrants contain motion. If they do it begins a path finding algorithm checking the perimeter

of this quadrant. If motion is detected adjacent to this quadrant the motion region is expand in the corresponding direction updating both the X,Y coordinate of the motion region and the width and height. Once there are no more adjacent regions of motion we terminate the search and encapsulate this region. Then we check to see if this region that has been found is larger than any we have found so far. If it is we store it in the structure that contains previous regions of motion. If not we simply discard this region.

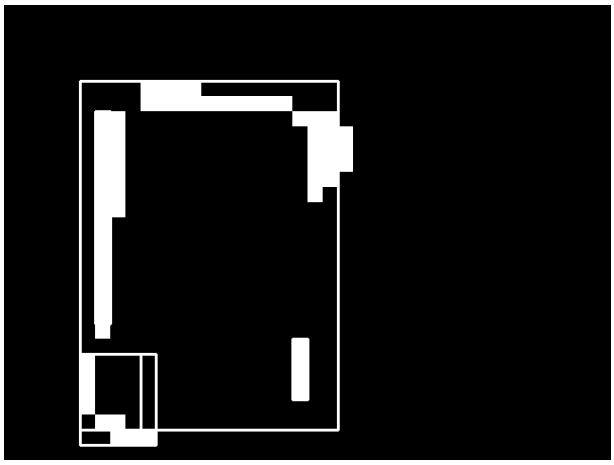

Figure 4: Image showing Encapsulation of Motion Regions from Difference Filter

This helps to reduce further noise if small regions of motion are occurring near large areas of motion as the robot will most likely be concern with the large regions of motion. Now that a list of motion regions have been stored we can begin to track the motion from one instance to another.

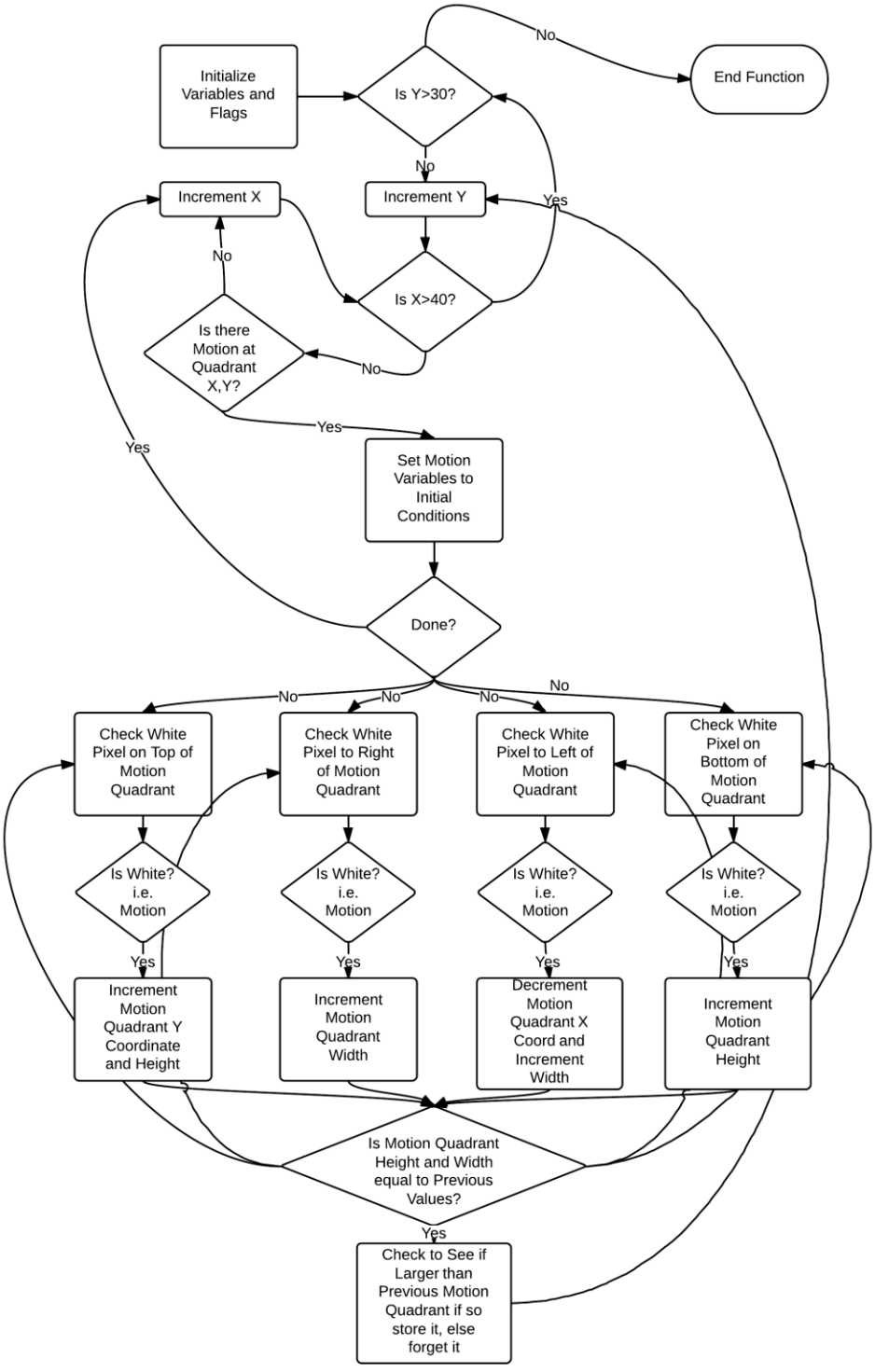

Figure 5: Overview of the Motion Detection Algorithm

However, the situation becomes complex if we attempt to track more than one region of motion as we cannot easily distinguish one object from another. We simply have information that points to regions of motion. To simplify the problem further and reduce the computation needed we combine all the regions of motion into one large region. This is accomplished simply by searching through the list of motion regions and finding the smallest and larges X, Y coordinates and then denoting a single motion of region using these values as its vertices.

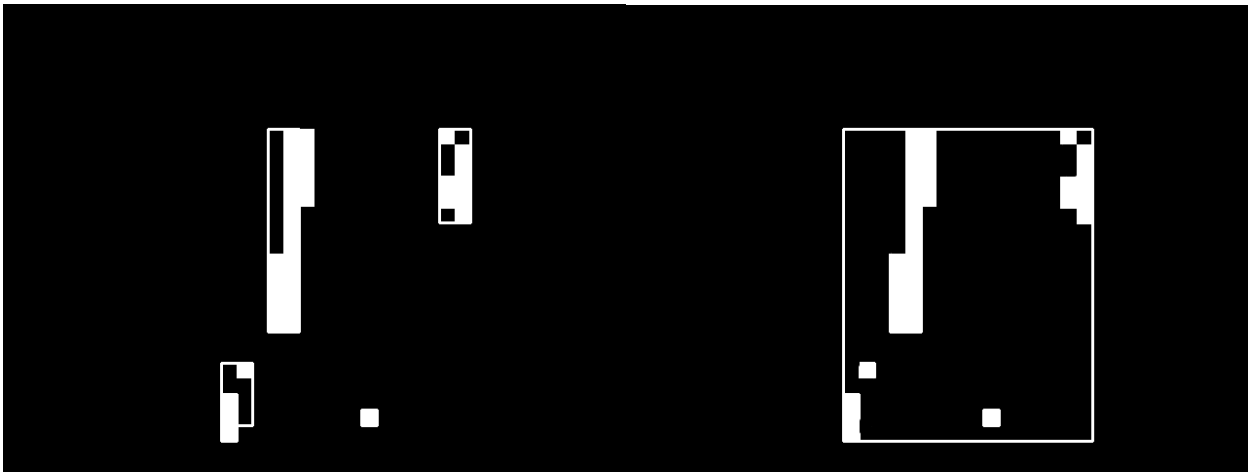

Figure 6: Demonstrating how we combine Regions of Motion into one Large Region. Right is combined Quadrants, Left is individual Quadrants.

As Figure 6 demonstrates the regions of motion are encapsulated into one large region of motion that is outline in the white lines. With a single region of motion we can now easily track where exactly that motion is in the image and indicate to the robot where that motion is travelling.

We accomplish this by separating the area of the image into three distinct regions: Left, Center and Right. The bounds of the region are given as quadrant 0 to 14 along the X axis as Left, 15 to 25 as Center and 26 to 40 Right. We then input the motion region into our function and determine the center of the motion and determine in which region the motion falls, Left, Right or Center. Note we only consider the X axis or one dimensional case so we cannot give information as to whether the object is moving closer or farther away from the robot by considering the Y axis. We choose to do this as to minimize the calculation needed and to create as simple as possible algorithm to be run on the robot. If we did consider this we could estimate the distance an object from the robot.

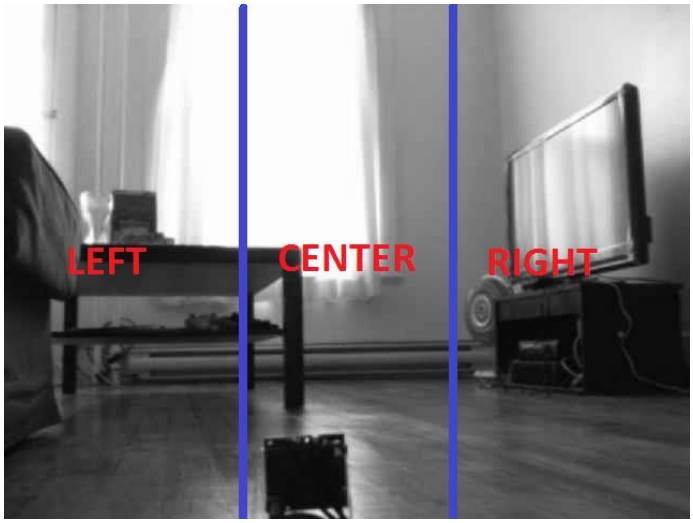

Figure 7: Example Image of how the Image is divided into Three Major Regions of Motion

We can see in Figure 7 how the motion is identified from the perspective of the robot. If a region of motion occurs in the Left region of the image the robot will move right to avoid, and if it occurs in the Right it will move Left. If a motion occurs in the center of its field of view the robot will move backwards to avoid. With all the components we can finally construct an algorithm for the robot to avoid regions of motion.

Figure 8 demonstrates the complete motion detection algorithm that will be used on the robot. The algorithm works by checking if motion has occurred. If motion has occurred the algorithm checks in which region and moves the robot accordingly. Once the robot moves a reset flag is set high to indicate the robot has moved to avoid an object that it has detected as motion. The reset flag helps to stabilize the image intensity as the webcam has an auto balance feature that stabilizes the images brightness when lighting conditions change. To avoid interference and false positive detection of motion we let the webcam adjust to the new brightness when the robot rotates to avoid motion. Once the rotation is complete and the image has stabilized the robot begins once again detecting if motion has occurred in the three major regions.

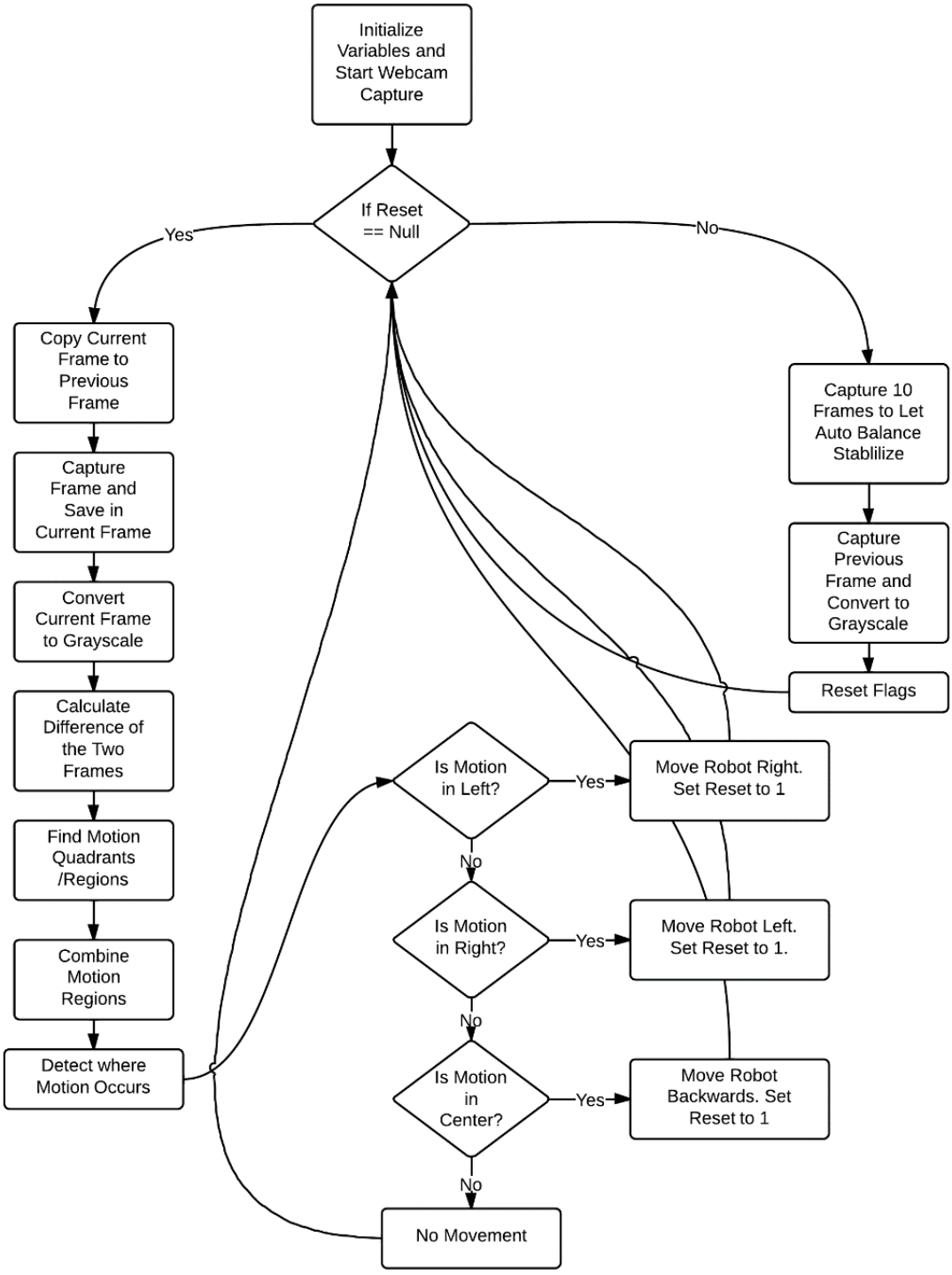

Figure 8: Full Motion Detection Algorithm

5. Results and Testing

Bluetooth Image Transfer

The testing of sending images over Bluetooth was done by simply sending images of varying sizes over Bluetooth and timing how long it took for the transfer to complete. Table 1 demonstrates the varying image sizes chosen, the time taken and the data transfer rate. The sizes were chosen to maximize the horizontal pixels in the images sent as to maximize the robot's horizontal field of view if the image transfer could be done successfully.

Table 1: Bluetooth Transfer Rate for Different Image Sizes

Image Size (WxH)	Raw Image Size in Bytes	Frames Per Second	Bluetooth Data Rate(KB/s)
640*480	307200	0.59	181.25
640*240	153600	1.18	181.25
640*120	76800	2.32	178.18

As we can see the transfer rate did not yield desirable results. Ideally we would want at least five images a second to be able to produce reasonable results for the robot. The maximum amount of frames per second that we were able to achieve was 2.32 FPS. This result is not ideal as the vertical amount of data being sent is greatly reduced. However, it may be sufficient for detecting movement for the robot in this narrow band.

Theoretical the maximum data throughput of the Bluetooth V4.0 that we are using is 375KB/s [4]. However we see that we are achieving only 48% of that value. Now this may be due to the choice of protocol as we have chosen to use RFCOMM which was the protocol used by previous groups on this robot. Now RFCOMM is a serial protocol that emulates RS-232 and is similar to TCP which can lead to slower transfer rates. A solution would be to use another protocol and that is the L2CAP protocol for communication which mimics a UDP connection [2]. UDP or User Datagram Protocol is a simple transmission model with minimum protocol mechanisms. There is no handshaking, guarantee of delivery or error checking. This protocol can lead to higher transfer rates, but the lack of synchronization could lead to problems. However, due to the limited time, this avenue for increasing the transfer rate was not explored, but it can be a viable option for succeeding groups that could allow for successful image streaming from the robot over Bluetooth.

Another point is that the images were being sent over Bluetooth as raw, uncompressed images. An improvement that could be made is compress the images using a JPEG or MPEG compression which can greatly reduce the amount of data that needs to be sent. However, this introduces more computation on the robot's part and decompression once the image is received. Again this avenue of improvement was not explored due to time constraints.

Optimized Motion Detection Algorithm

The motion detection algorithm that we have created was tested on both the robot and a PC to compare the results of the computation time. Table 2 demonstrates the time taken to compute the difference of two consecutive frames and detect motion in the difference. We see even with the optimization that the difference algorithm is taken much longer than the motion detection algorithm on the robot. When compared to a PC which takes 20 to 40ms to compute the difference and detect motion. Compared to the PC we see that the code is not suitable for the robot and is a little too slow for detecting quick motion.

Table 2: Computation Time for Motion Detection Algorithm

Algorithm	Computation Time (seconds)
Optimized Difference Filter on Robot	0.96
Motion Detection on Robot	0.000001
Full Difference and Motion Detection on PC	0.02-0.04

The major computation time is calculating the difference of two images. In the initial optimization, the difference filter scans through the whole image checking the intensities of all the pixels. In order to reduce the computation we can skip some of the pixels to reduce the amount of time needed to calculate the difference and so create a more responsive motion detection algorithm.

Table 3: Computation Time for Varying Difference Filter Settings

Difference Image Search Percentage of 640x480	Amount of Pixels Searched	Computation Time (seconds)
100%	307200	0.96
50%	153600	0.62
25%	76800	0.45

As Table 3 shows when we decrease the amount of pixels the difference filter has to search we reduce the computation time slightly. Even at a quarter of the amount of pixels searched we see that we reduced the computation time by a half compared to the full search. Of course here we lose a lot of information of the image, but since the algorithm is tasked with detecting largest area of motion on the image this can be deemed acceptable.

Another interesting point that we found when testing our algorithm is that when the robot moves due to its response to movement the balance of the image changes due to different lighting conditions. So to avoid false positives being given when there is no movement the webcam was given time to allow it to adjust to the new lighting condition when the robot rotates or moves.

The advantage of using a webcam for motion detection rather than ultrasonic sensors is the range the webcam can view an object. Now the robot's motion detection range was tested simply by crossing the robot's field of view at varying distances. From testing we found that the maximum distance our motion detection algorithm can pick up a person is at a range of 7.5 metres. The debugging information showed that at that range one or two quadrants of 16x16 pixel size were be shown as areas of motion. Of course this testing was carried out with one single moving object or person in this case. If more than one object is moving within the robot's field of view the robot will give false results as to where motion is occurring.

7. Impact on Society and Environment

The impact that robotic vision can have on society in a general and broad sense is large. Allowing a robot or computer the ability to detect motion or objects, opens up the capabilities and tools a robot may have to analysis and observe its surroundings.

Robotic vision opens the possibility for more sophisticated autonomous robots that could be used in a variety of applications from space exploration to humanoid robots. Although the problem with robotic vision is how the robot can know what or where an object is from a two dimensional image. The challenge becomes how can an image be parsed and in order to extract the useful information such as where objects are, which direction are those objects moving and even identify those objects.

From this design project we can see that it takes quite a lot of processing power to parse a whole image pixel by pixel. For robotic vision to work on small embedded computers that would often be found on mobile robots we need algorithms and hardware that are specialized for the sole purpose of image processing and image analysis.

8. Teamwork

The division of labour this semester was split even this semester. Having Yulric create the motion detection algorithm in Java allowed Christopher to convert that code to C and optimize it to run on the robot's small embedded computer. We meet each other regularly to keep one another up to date on the progress of the project.

Once again weekly meetings were held with Professor Vybihal along with his other Design Project groups. We kept Professor Vybihal up to date on our progress throughout the semester and the problems that we had encountered during the semester such as the drawbacks of the Bluetooth image transfer.

9. Conclusion

The motion detection algorithm that we have created was able to run on the embedded computer of the two-track robot. However, from the results and testing we see that a standard webcam image contains hundreds of thousands of points of data and to process all these points in a short amount of time is difficult on such a low power system.

However, the system itself is not necessarily a poor embedded system, but with an OS such as Linux running on top of this embedded computer it becomes hard to utilized 100% of the computation power of the processor. Overhead costs and limited access to the underlying hardware on the processor such as timers and peripherals due to the OS can limited how much computation power can be extracted. Although it has to be noted that having an OS running on top of the embedded computer does make it easier to interface and program for. However, the embedded Linux computer that we used for the project was never intended for computation intensive algorithms, but more simple operations and so it is possible to implement such an algorithm on specialized hardware.

In addition, the possibility of sending the image over Bluetooth can be done given more time. There are multiple ways of accomplishing image streaming over Bluetooth by either changing the communication protocol or compressing the image. One solution would be to use a different communication method such as Wi-Fi to accomplish image streaming because of its higher data throughput.

10. References

[1] "BlueZ Project," 2010. [Online]. Available: http://www.bluez.org/.

[2] A. Huang, "An Introduction to Bluetooth Programming," Cambridge University Press, 2008. [Online]. Available: http://people.csail.mit.edu/albert/bluez-intro/index.html.

[3] "OpenCV," Itseez, 2014. [Online]. Available: http://opencv.org/.

[4] "Bluetooth 4.0 USB Micro Adapter," IOGear, 2014. [Online]. Available: http://www.iogear.com/product/GBU521/.