Stereoscopic Vision for Autonomous Navigation

Final Report

By:
John Hessling

Advisor:
Dr. Huggins

May 14, 2010

I. Introduction

There are many forms of navigational aids that are used in autonomous navigation. There are sonar sensors that can measure distance to the nearest object in a cone of direction. Then, there are laser sensors that are more precise, yet have a smaller cone of direction. However, these are much more expensive, and cannot distinguish between objects. Then there is mono imaging that consists of one camera. This will provide a clear view of objects in the path, but does not provide information regarding depth. Therefore, to combine the best traits of all of these methods, two cameras are used and the pictures merged to provide detailed information. This is called Stereoscopic Vision. This system has been used on mars rovers and experimental cars. This project created an integrated, self-contained stereoscopic imaging system.

Goals

This is a basic goals list for the project but ultimately a fully functional self contained unit with precise object distance detection in the field of view.

- Field of view greater than 17.5 degrees each side of center for a total of 35 degrees.
- Field of view will be from 1 meter to 10 meters
- Communication to client unit
- Detect multiple objects in the field of view
- Distance accuracy shall be 5 cm per meter distance
- Three separate run modes: Calibrate, Test, and Navigate mode

II. Functional Description and Requirements

The system will consist of only two digital cameras mounted on an adjustable rig. This rig must be capable of making directional adjustments during calibration and then maintain position for the duration of use. There will also be a desktop/laptop computer for the processing power needed. Since the implementation will not be mobile a desktop computer will be used for this project. However, the final project will be a self-contained unit that will require a built-in processing unit. The basic hardware will be of a setup of the two mounted cameras feeding information through a frame grabbing device into a computer as shown in figure 1. The output will be sent through the usb connection for normal run mode. While in calibration mode or in testing mode, communication will be from keyboard and to the screen. Testing mode will still send a copy of the information through the usb connection but only for use in testing.

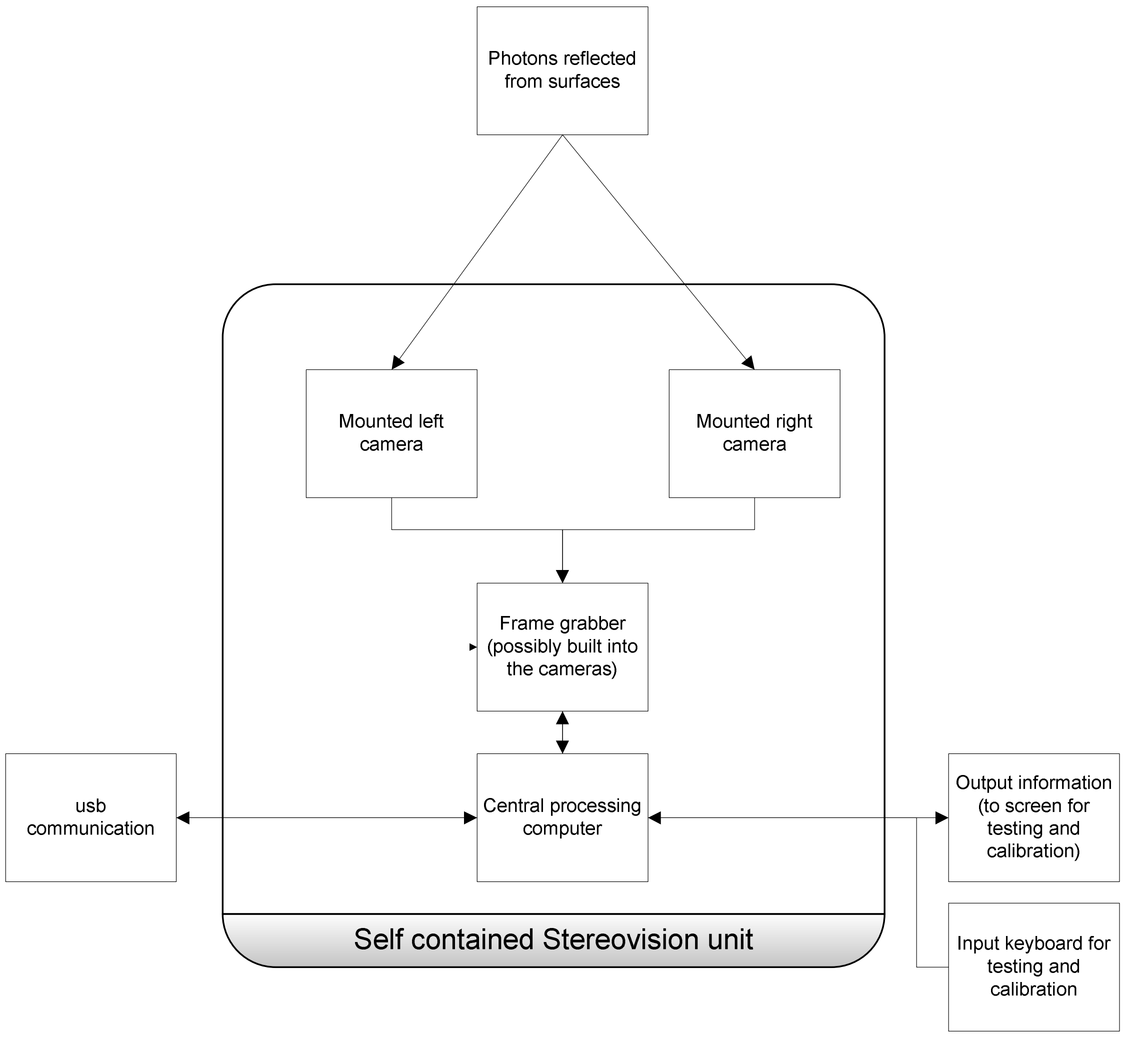

Photons reflected from surfaces
Mounted left camera
Mounted right camera
Frame grabber (possibly built into the cameras)
usb communication
Central processing computer
Output information (to screen for testing and calibration)
Input keyboard for testing and calibration
Self contained Stereovision unit

Figure 1

The system will consist of three different modes: Camera mount calibration, Testing, Navigate mode. This is illustrated in figure 2.

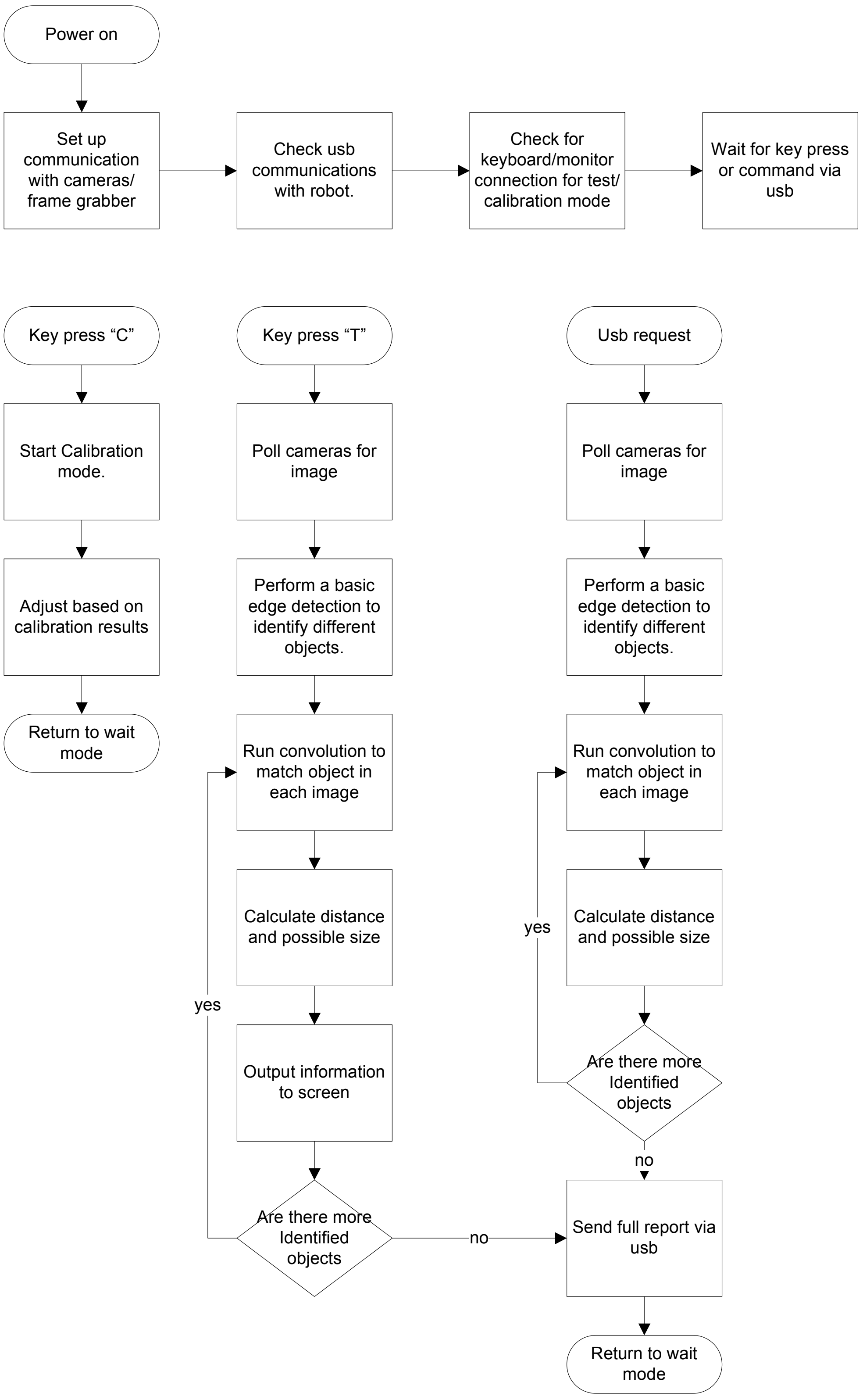

Power on
Set up communication with cameras/ frame grabber
Check usb communications with robot.
Check for keyboard/monitor connection for test/ calibration mode
Wait for key press or command via usb
Key press “C”
Start Calibration mode.
Adjust based on calibration results
Return to wait mode
Key press “T”
Poll cameras for image
Perform a basic edge detection to identify different objects.
Run convolution to match object in each image
Calculate distance and possible size
Output information to screen
Are there more Identified objects
yes
no
Usb request
Poll cameras for image
Perform a basic edge detection to identify different objects.
Run convolution to match object in each image
Calculate distance and possible size
Are there more Identified objects
yes
no
Send full report via usb
Return to wait mode

Figure 2

III. Theory of SI

For this project we are using a geometric approach to calculate object's distance. The model for this geometric approach operates under the assumption that the cameras are "Ideal Pinhole" in nature. This implies that the cameras operate as the figure 3. An Ideal pinhole camera has no lens. Therefore, it has no distortion. This means that a pixel relates directly to an angle. The cameras that are being used do have lenses, but the manufacturer has software that is part of the operating driver which adjusts the image to compensate for the lens distortion. The software also flips the image horizontally and vertically. Part of the camera pixel ration calibration was to verify that the images received from the camera were adjusted correctly to reflect the ideal pinhole model.

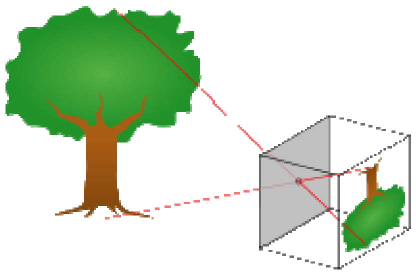


figure 3

Camera Pixel Ratio Calibration

To find the ratio that relates pixels to angle, I used an experimental approach. First, I used one camera. The camera was aimed perpendicular to a flat wall. The centerline of the image was marked on the wall. Then, the visible edges both left and right were marked. Measurements were taken as shown below. The following equations are used to calculate the full angle of view for the camera.

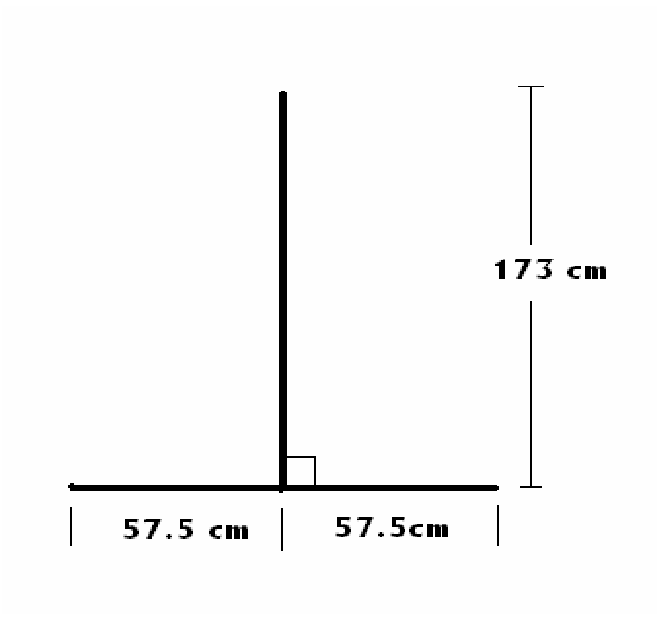


figure 4

Angle = $\tan^{-1}(57.5/173) + \tan^{-1}(57.5/173)$

This gave an angle of 38.31°. The image from the camera is 352 pixels wide so the angle to pixel ratio is 0.10753. Since MATLAB calculates in radians, this was converted to 0.001877 radians per pixel. This process was repeated to verify both cameras would have the same angle to pixel ratio. The other camera yielded a ratio of 0.001885. The process was then run for both cameras at a distance of 100 cm from the wall. The results were always 0.0019 +/- .00005. The error was within 2.7%. That error includes my human error while taking measurements.

Then I tested using angles that were not all the way at the edge of the image. These results are listed in the table below, but they also show the same ratio.

Figure 5

Ratio Calculations

	Distance away	Lateral distance	number of pixels	Calculated ratio	Percent different from .0019
Left Camera	173	119	352	0.0018768	1.22
	100	71.75	352	0.0019515	2.71
	281	196	352	0.0019012	0.06
	100	36	184	0.0019358	1.88
	100	18	92	0.0019513	2.7
Right Camera	170.75	118	352	0.0018849	0.79
	101.25	72	352	0.0019355	1.87
	281.5	197	352	0.0019071	0.37
	101.25	36	181	0.0019441	2.32
	101.25	18	92	0.0019273	1.44

This verifies that the assumption that the corrected images can be treated as “pinhole models”. This ratio was put into the program as a constant with a value of 0.0019. The validity of this constant only holds true if the focus on the camera is not changed. When the camera focus is adjusted it does so by changing distance from the lens to the photo panel. This not only changes the focus, but also changes the angle to pixel ratio. To minimize the focus issue, both cameras were focused correctly for an object placed four meters away. The closer objects and the farther objects are therefore slightly out of focus. However the edge detection used on these images is still able to find the edges, even with a slight blur.

Distance calculation

Once an edge has been detected in one image and has been correlated to the corresponding edge in the other image, the system can calculate relative position of the edge. There are three different geometric models and equation sets used based on the relative location of the edge. Recall that the pixel to angle constant is 0.0019 for the cameras I used, but the equations below have been generalized so I will use (ac) as this constant. I will use (W) as the image width in pixels. (Lp) and (Rp) will be the location of the edge in the left and right image respectively.

Left of center on both images:

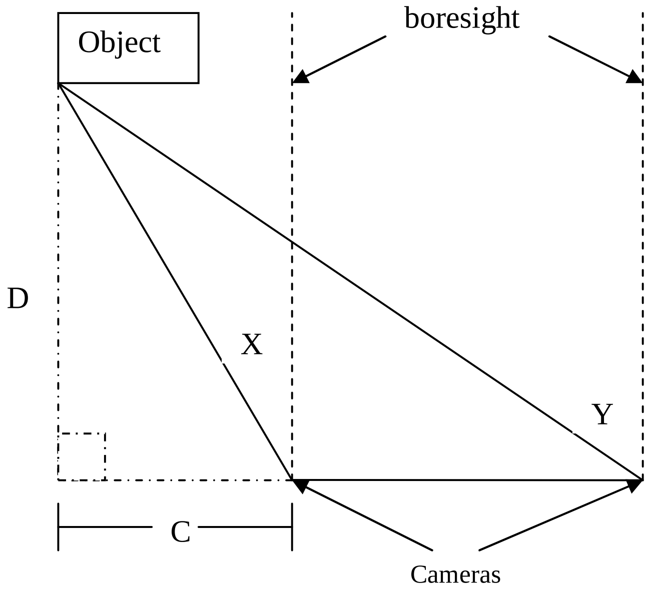


Figure 6

Derivation

We begin with knowing angle X and Y.

The complement of those angles is L and R

Triangle left

L = tan (D/C)

$\text{Tan}^{-1}(L) = D/C$

$D = \tan^{-1}(L)*C$

Triangle right

R = tan (D/(C+30))

$\text{Tan}^{-1}(R) = D/(C+30)$

$D = \tan^{-1}(R) * (C+30)$

Combine equations and solve for C

$\text{Tan}^{-1}(L)*C = \tan^{-1}(R)*(C+30)$

$C*(\tan^{-1}(L) - \tan^{-1}(R)) = 30* \tan^{-1}(R)$

$C = (30* \tan^{-1}(R))/ (\tan^{-1}(L) - \tan^{-1}(R))$

Put the value of C back into either of the triangle equations

Left image angle (L) = Π/2 - (W/2 – Lp) * ac

Right image angle (R) = Π/2 - (W/2 – Rp) * ac

Lateral position (LP) = $- ((\tan^{-1}(R) * 30)/ (\tan^{-1}(L) - \tan^{-1}(R)) + 15)$

Distance = $\tan^{-1}(R) * (C + 30)$

Right of center on both images:

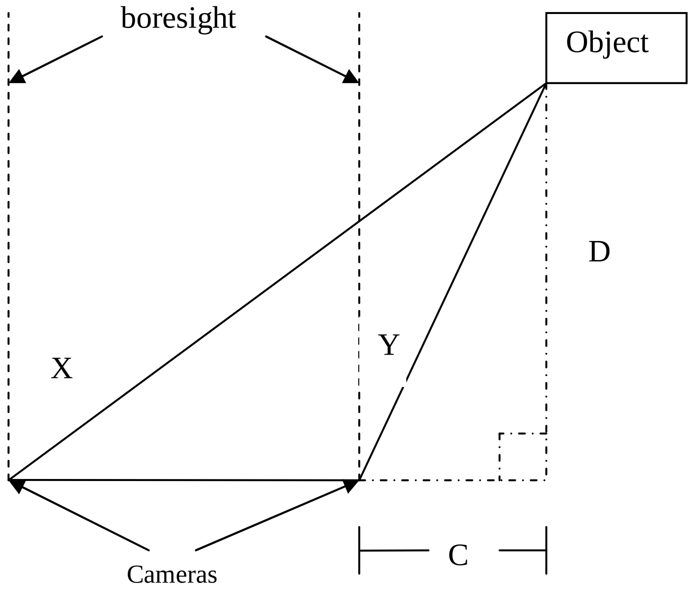


Figure 7

Derivation
We begin with knowing angle X and Y.
The complement of those angles is L and R
Triangle left
L = tan (D/(C+30))
Tan^{-1}(L) = D/(C+30)
D = $\tan^{-1}$(L)* (C+30)

Triangle right
R = tan (D/C)
Tan^{-1}(R) = D/C)
D = $\tan^{-1}$(R) * C)

Combine equations and solve for C
Tan^{-1}(L)* (C+30) = $\tan^{-1}$(R)*(C)
C*($\tan^{-1}$(R) - $\tan^{-1}$(L)) = 30* $\tan^{-1}$(L)
C = (30* $\tan^{-1}$(L))/ ($\tan^{-1}$(R) - $\tan^{-1}$(L))

Put the value of C back into either of the triangle equations
Left image angle (L) = Π/2 - (Lp - W/2) * ac
Right image angle (R) = Π/2 - (Rp - W/2) * ac
Lateral position (LP) = - ($\tan^{-1}$(L) * 30)/ ($\tan^{-1}$(R)- $\tan^{-1}$(L)) + 15
Distance = $\tan^{-1}$(R) * C

Right of center on left image and left of center or right image:

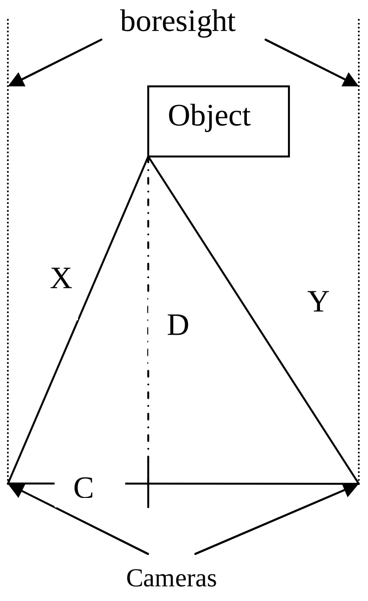


Figure 8

Derivation

We begin with knowing angle X and Y.
The complement of those angles is L and R
Triangle left
L = tan (D/C)
$\mathrm{Tan}^{-1}(L) = D/C$
$D = \tan^{-1}(L)* C$

Triangle right
R = tan (D/ (30-C))
$\mathrm{Tan}^{-1}(R) = D/ (30-C))$
$D = \tan^{-1}(R) * (30-C))$

Combine equations and solve for C
$\mathrm{Tan}^{-1}(L)* C = \tan^{-1}(R)* (30-C)$
$C*(\tan^{-1}(R) + \tan^{-1}(L)) = 30* \tan^{-1}(R)$
$C = (30* \tan^{-1}(R))/ (\tan^{-1}(R) + \tan^{-1}(L))$

Put the value of C back into either of the triangle equations
Left image angle (L) = Π/2 - (Lp - W/2) * ac
Right image angle (R) = Π/2 - (W/2 – Rp) * ac
Lateral position (LP) = $(\tan^{-1}(R) * 30)/ (\tan^{-1}(L) + \tan^{-1}(R)) + 15$
Distance = $\tan^{-1}(L) * C$

IV. System implementation

Hardware

The hardware setup for this project was simple in nature. There are two mounts on a wooden platform using mounting brackets from Jameco model number 15-B004. Mounted on these brackets are two Logitech webcams model number 861078-0030. The webcams were chosen because they are inexpensive and were already available in-house. There is according to Logitech's website software, built into the webcam's driver, which limits the effect of lens distortion. This software is called "RightLight technology". According to the manufacturer, this software adjusts the images from the camera to compensate for both the lens distortion and brightness. This adjustment to the images means that I can use the images as if they were from a pinhole model camera.

Figure 9

Image capture

There is a built in procedure for image grabbing in the image processing toolbox for MATLAB. The first step is to create an object variable by using the "videoinput" command.

```
obj1 = videoinput('winvideo', 1);
```

After the object has been created there is a "preview (obj1);" command that will open a new window and continuously display the video feed. For most of my code I issue the preview command to give the cameras a time to normalize. The "stoppreview (obj1);" will end the preview and close the preview window. To actually capture the image and save a copy we use the "getsnapshot" command.

```
imagel = getsnapshot(obj1);
```

This saves a copy of the image from the camera and saves it into "imagel" variable.

Edge detection

The use of an edge detection algorithm is optional for stereoscopic vision. However, there are benefits to this implementation. The processing time to correlate a two dimensional binary array is much faster than that of a three dimensional double array. For this main reason, I chose to employ the use of edge detection before correlating. Originally, I had written an algorithm for edge detection. My program was met with limited success. There was no way to change the measure by which an edge was defined. I then reverted to the "edge" function located in the image processing toolbox. This function takes a grayscale image, and using one of six possible algorithms, returns a two dimensional binary array. The six possible algorithms are: sobel, canny, prewitt, roberts, log, zerocross. The algorithm is specified during the function call. I chose to use the "canny" algorithm, based simply on results and reading information online. The gray scaled image and the edge threshold are also specified during the function call. The threshold allows the user to specify to the function that is to be considered an edge. Since

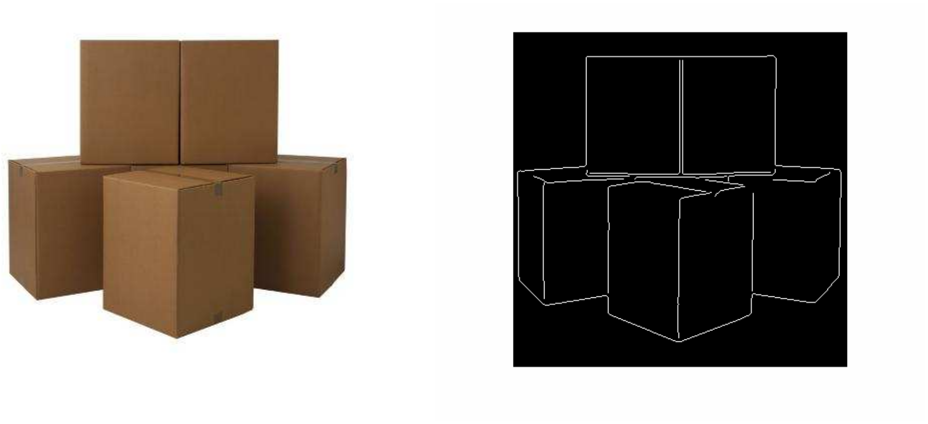


Figure 10

the "edge" function only accepts grayscale images, I first had to run the "rgb2gray" function from the image processing toolbox.

Correlation technique

The correlation procedure begins with the images that have been through the edge detection. They have also been adjusted as per the correction coefficients. There is a built-in correlation function in MATLAB that can correlate 2 dimensional arrays. I use the edged left image and find the edge that is lowest in the foreground.

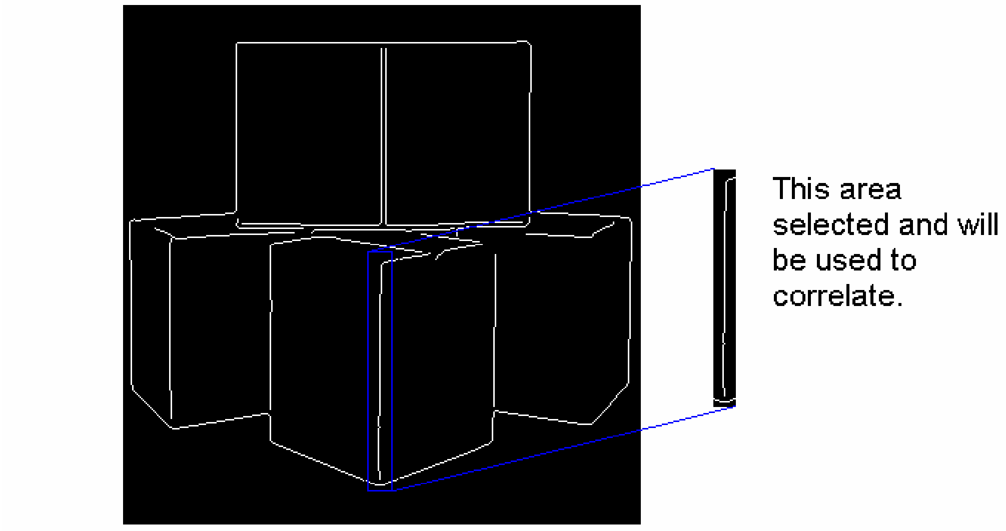


Figure 11
This will be the edge that extends the lowest in the image. I then select the vertical part of that edge plus the surrounding 5 pixels and set it as its own two dimensional array. I then

use the number of pixel up from the bottom on the original image that the edge was found and the total height of the edge to set as parameters for what part of the second image to run the correlation.

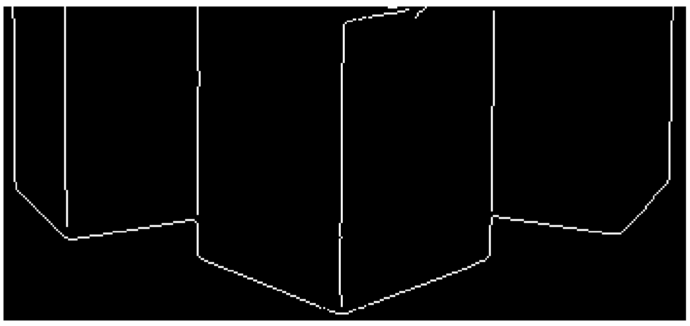


Figure 12

The returned array is scanned for the highest value. This value is the location of the edge on the right image. These location values will then be used to calculate distance to the edge. The edge that was just correlated is then removed on both images. This is to avoid a false positive during the next edge correlation.

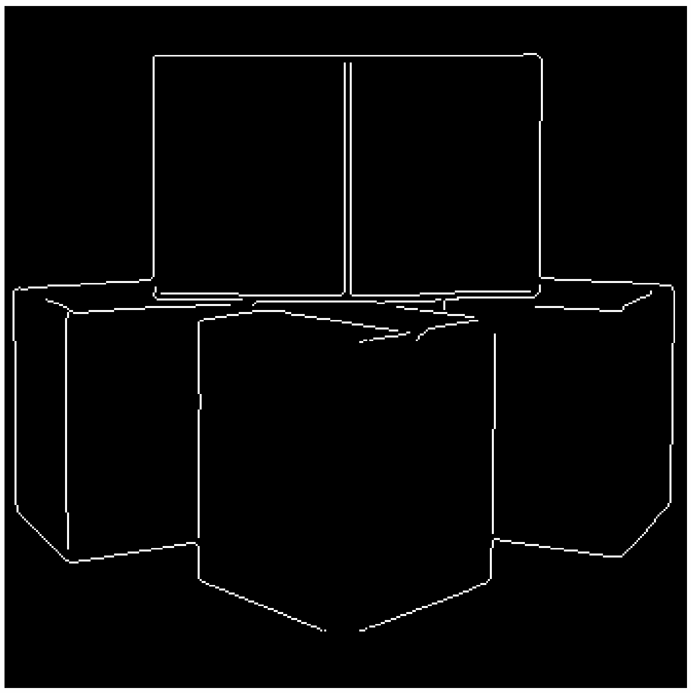


Figure 13

Camera Alignment Mode

The calibration sequence in this project is the most important step to ensure accuracy. The distance calculations are based on the assumption that the cameras are aimed strait and level. The calibration tool consisted of a pair of crosshairs set 30cm apart and at the same height as the cameras. The calibration GUI displays a live feed from the

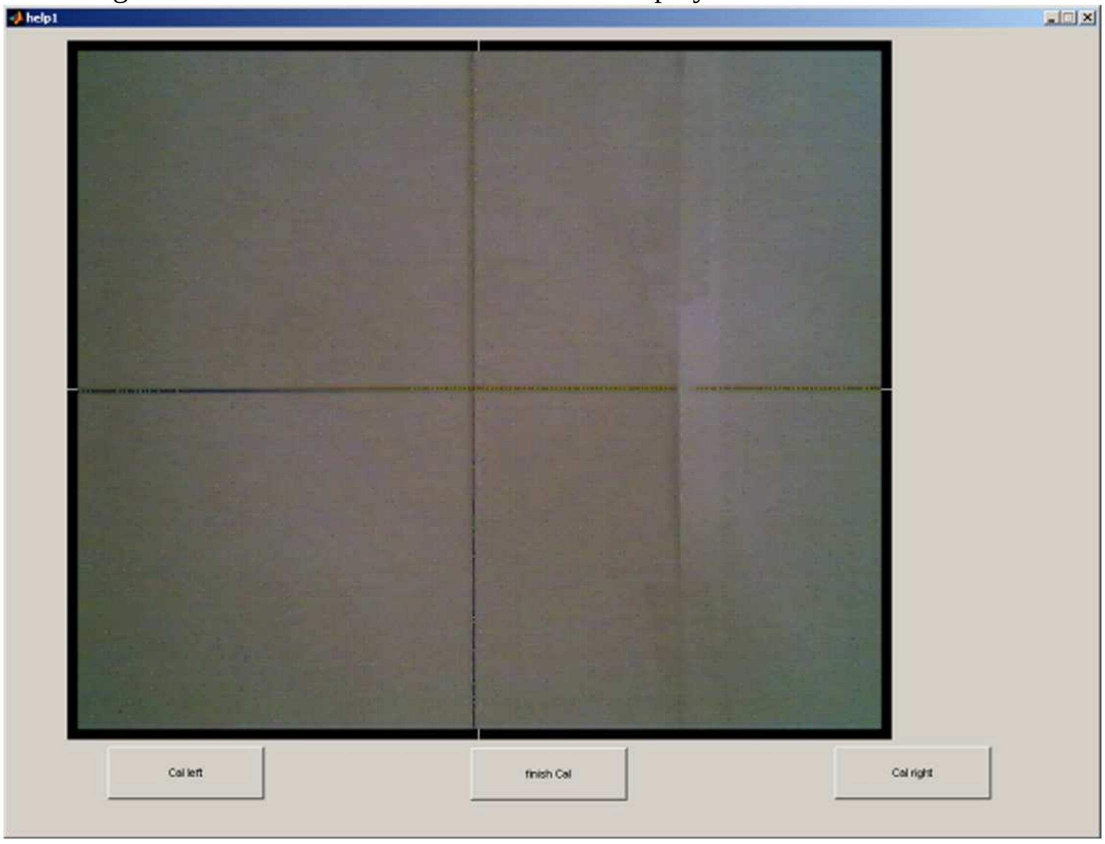


Figure 14

cameras, one at a time, allowing the user to line up the crosshairs for a rough adjustment. The system does not require the crosshairs to be lined up perfectly. After adjusting both cameras, the program captures the image of the crosshairs from both cameras. This information is then used to calculate three coefficients for each camera. These coefficients adjust the image in three ways: vertical, horizontal, and rotational. These coefficients are saved to be used prior to the edge detection algorithm in the testing and navigation modes. There is an inherent flaw in this procedure, as I am assuming that the crosshairs are directly allied with the cameras, and that the calibration tool is perfectly

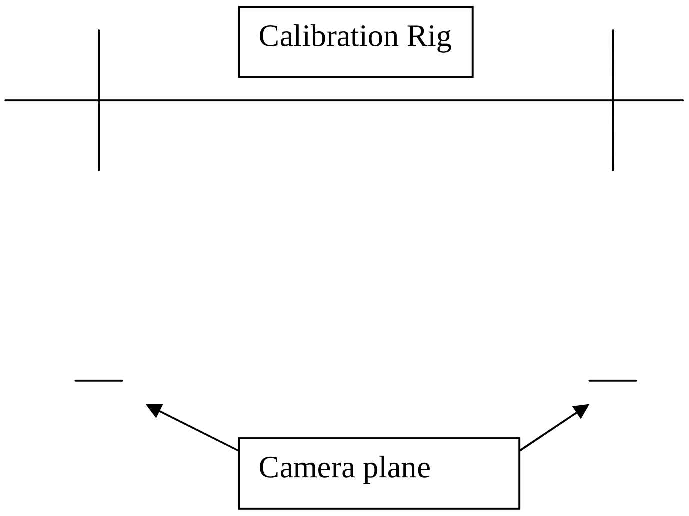


Figure 15

parallel with the camera mounting rig. The table below shows the relationship of the angle off parallel combined with the distance the calibration rig and gives the effect of error in pixels. The effect the 8 pixels off alignment would have on an object directly in front of the system and 8 meters away would be an error of 5 meters. The listings in red would have over a 100% error for an object 8 meters away.

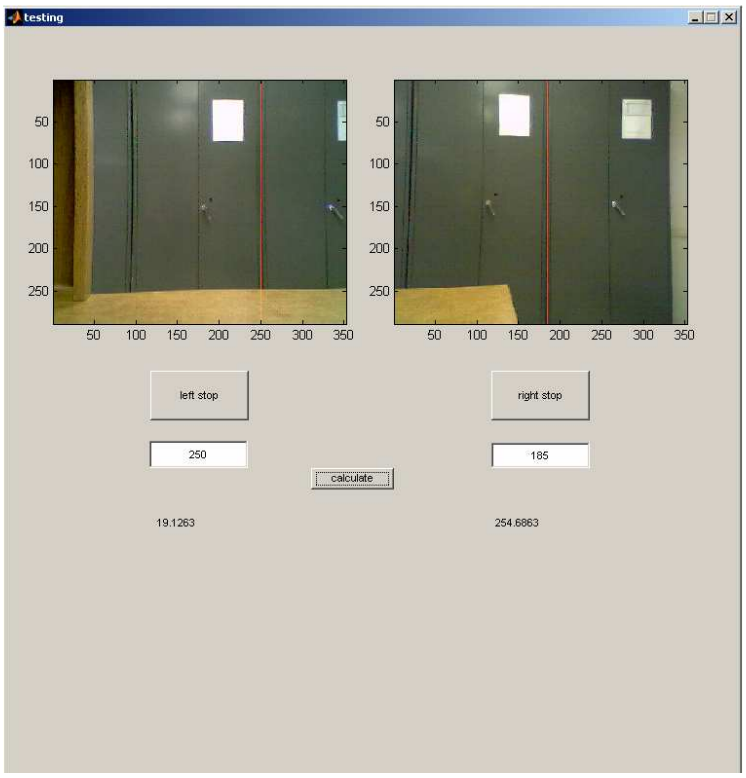
Figure 16

		Mounting rig distance		
		30	20	10
angle off parallel mounting rig	5	2	3	6
	10	8	12	24
	15	18	27	54
	20	32	48	96

This has a net effect of making the system cross-eyed. These two factors combined give a high probability of failure.

Testing GUI

The testing GUI was divided into two parts: Testing-A and Testing-B. The purpose of Testing-A GUI is to check the distance calculation capability of the system. This GUI is fairly user dependant. A preview of each camera is shown. This gives the cameras a chance to come online and adjust the brightness of the image. Then, the user pushes the “left stop” and “right stop” buttons to freeze the images. The user must then input the approximate x-coordinate of the edge they want calculated for each image. The user’s input doesn’t need to be exact as the system will look for the closest edge to the value inputted. Then the user selects the “calculate” button. The system accesses the adjustment coefficients that were saved during the most recent calibration. It uses these to correct the images that were just collected. The system then runs the edge detection function on each



250

185

19.1263

254.6863

image. Based on the results from the edge detection and the user inputted coordinates, it selects the closest edge. It displays the images with a vertical line overlaid at the x-coordinate that it found. The distance calculations are made and the results are placed below. The number on the left is the lateral distance in centimeters and the number on the right is the distance in centimeters. This GUI only calculates one edge at a time but the user can select edges multiple times using the same images. The Testing-B mode bypasses all of the user inputs except the calculate button. Upon startup the GUI shows the two raw video streams to the screen. Once the videos have adjusted for the brightness, the user can select the calculate

Figure 17

button. The software then captures an image from each camera. Then it runs the edge detection and the correlation algorithm. Based on the location of the correlated edges it makes distance calculations and displays them in a text box located at the bottom of the screen. It also saves this text information in a file for later retrieval.

Navigation mode

The final mode for the project does not have an interactive GUI. This is designed to be accessed via usb or comport from another computer. The requesting computer would send a command to the system requesting information. The system run an equivalent to Testing-B mode without the user selecting calculate. The resulting text file that contains the information about the objects is then sent back to the requesting unit. To do this I am using a simple dos ".bat" file and windows "hyper terminal". I have not worked on the usb interface.

V. Experimental Results

The results have been mixed for a few reasons. The portion of the project that has created the most error is the calibration procedure. There have been calibration runs that seemed to produce error rates of less than 3% over the entire distance range of 1 to 10 meters.

Such examples include an error of 4cm measuring a distance of 196cm and an error of 22cm measuring a distance of 874cm. While these were valid results it seems to be the exception rather than the rule. They have been shown to not be easily repeatable and therefore nonscientific. The simple fact is if the calibration tool is not aligned perfectly and perfectly parallel then each camera can be off by a few pixels. With each camera off only three pixels this can create an error of about 100cm for an object at a distance of 800cm. This corresponds to an error of 12.5%. This possible error is not acceptable. There is also a problem with the mounting system in that the cameras are not held in the same position when the rig is moved. The mounts allow for too much movement. This means that the unreliable calibration procedure must be repeated every time the rig moves. A more robust and stable mounting system would limit this, but to what extent I can only speculate. The second problem involves the correlation procedure. The procedure for correlation starts with locating the closes edge in the foreground, which is the edge that appears lowest in the image. The problem is lowest edge in one image may not correspond to the lowest edge in the other image.

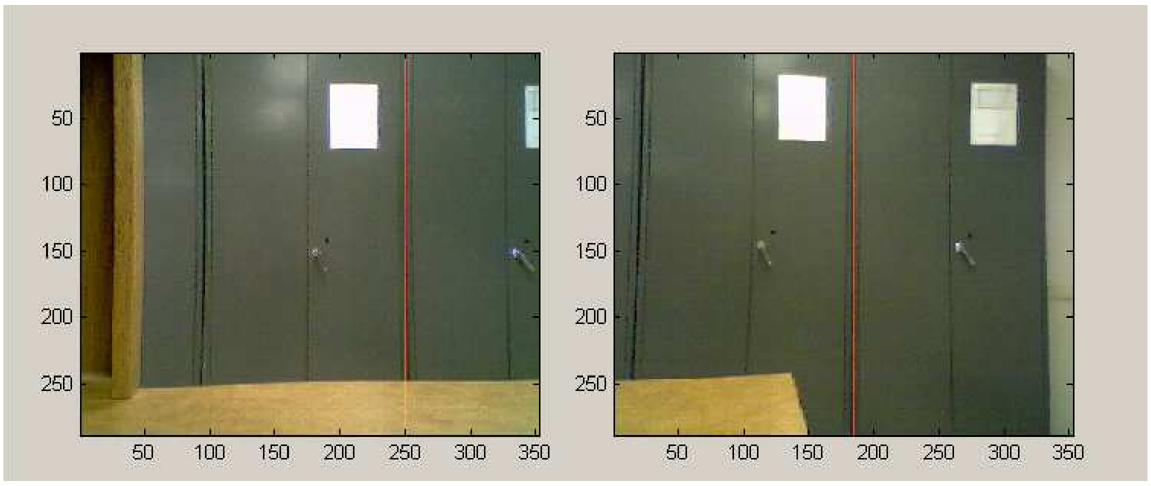


Figure 18

The image above is example that would create an error in the correlation process. Notice that the first edge seen in the right picture is the bottom of the cabinet toward the right side of the image. This edge does not appear in the image on the left. This flaw can be avoided by clever programming, but I did not encounter this issue early enough to make corrections.

VI. Future work

There are several things that could be done as future work on this project. The first would be creating a more robust but still adjustable mounting system that would allow the cameras to be adjusted but then locked down so they cannot move during transport. The second thing is to change the calibration rig. This would make the results easier to repeat. The calibration tool probably should be mounted in a fixed position relative to the cameras instead of trying to reposition the crosshairs my eyeing its position. The third thing to make the system deal with the correlation problem encountered. The final addition would be to complete the self-contained-ability of the unit by creating the usb interface.

VII. Conclusion

The simple conclusion is this product is not ready for any use at this time. There are too many issues with accuracy and the object correlation that need to be corrected first. I do feel that we are on the correct track and upon completion of the future work this system would be a viable option for an autonomous navigation aid.

VIII. Patents and Standards

There are only a few patents that affect this project. The first are patents that have to do with the Logitech cameras used. There are several patents on the webcams themselves and a patent on the "RightLight" software that adjusts the images from the camera. There are also patents held by NASA for the system used on the mars rover. This limited my ability to investigate the algorithms they used to identify objects and calculate distances.

IX. Bibliography

[1] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1/2/3):7-42, April-June 2002.

[2] Microsoft Research Technical Report MSR-TR-2001-81, November 2001. http://vision.middlebury.edu/stereo/taxonomy-IJCV.pdf

[3] http://www.terryblackburn.us/WildIdeas/stereographic_theory.htm#History

[4] http://www.mathworks.com

[5] http://en.wikipedia.org/wiki/Pinhole_camera