Autonomous Vehicle Navigation Using Stereoscopic Imaging

Project Report

By:

Adam Beach
Nick Wlaznik

Advisors:

Dr. Huggins
Dr. Stewart

May 15, 2007

Abstract

This capstone project is an integrated hardware-software system that autonomously navigates a mobile platform through a terrain using stereoscopic imaging, which is the same technique used by the Mars rovers, Spirit and Opportunity. Stereoscopic imaging allows 3D information to be extracted from images obtained by two cameras. The system utilizes color correlation to determine the shift index between corresponding pixels of an image set. The software then uses this data to calculate the distance from the platform to the obstacles using the pinhole model for the cameras. Finally, control signals are generated, based on this distance information, to move the platform through the terrain without collisions.

Table of Contents

I. Introduction

The objective of the Autonomous Vehicle Navigation Using Stereoscopic Imaging senior capstone project, NavBot, is to develop a robot that can independently navigate through a terrain that contains colored obstacles. The system utilizes stereoscopic imaging and color correlation to detect objects in its path. It then calculates the distance from the robot to the obstacles. Distance calculations are made using the pinhole model for the cameras. There will be two modes of operation for the robot. The first mode, calibration, will be used to setup and ensure that the subsystems are functioning within specifications. Navigation mode will be the main mode of operation. The goal is to navigate through the terrain as quickly as possible.

II. System Description

The system consists of two cameras, a Gateway laptop computer, and an ActivMedia Pioneer 2 Mobile Robotic Platform as depicted in Figure 1. The robotic platform is the same one that was used in the GuideBot and MapBot projects of 2005 and 2006 respectively. The cameras are Logitech Buddy Cams. The two cameras and the laptop computer are mounted on top of the robotic platform.

The NavBot uses stereoscopic imaging and color correlation to assess the terrain through which it is moving. Stereoscopic imaging is a technique used to create a three-dimensional map from 2 two-dimensional images. The distance from the robot to obstacles in its view can be calculated from the resulting three-dimensional map. The robot will be stationary when the images are to be taken, allowing the images to be captured one after the other rather than simultaneously. The three-dimensional map is generated using edge detection and color space correlation. Appendix A contains a short description of color space correlation. The distance to the obstacles is calculated using the pinhole model for the cameras. Information regarding the pinhole model can be found in Appendix B.

There will be two modes of operation: Calibration and navigation. The calibration mode will be used to set up the various subsystems and confirm that they are operating within the specifications. The system will switch to navigation mode once calibration is complete.

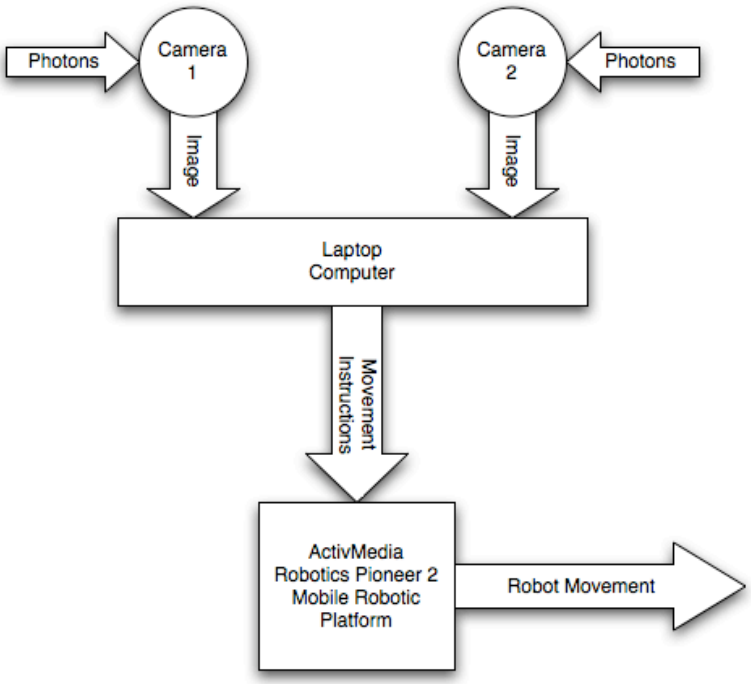


Figure 1: System Block Diagram

A. Subsystems

As depicted in Figure 1, the system has three subsystems. They are the camera subsystem, the laptop and software subsystem, and finally the robotic platform subsystem.

1. Camera Subsystem

The camera subsystem consists of two Logitech Buddy Cams. These are color webcams that interface to the laptop via USB. As depicted in Figure 2, the cameras are mounted on a horizontal platform rigidly attached to the robot and are a fixed distance apart. The camera subsystem converts photons into digital data upon a trigger sent to them from the laptop computer. This digital data is compatible with Matlab. The image is stored in the Matlab workspace as a three-dimensional array containing the red, green, and blue values (0-255) for each pixel. The inputs and outputs to the camera subsystem are shown in Figure 2. The cameras function the same in the calibration and navigation modes.

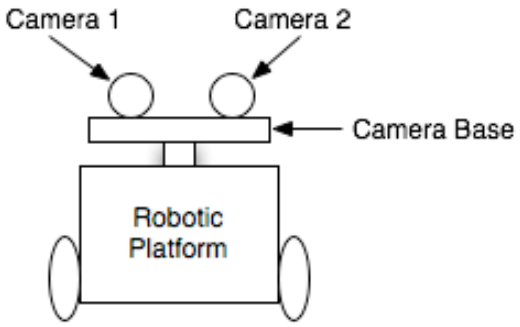


Figure 2: Illustration of Camera Mount

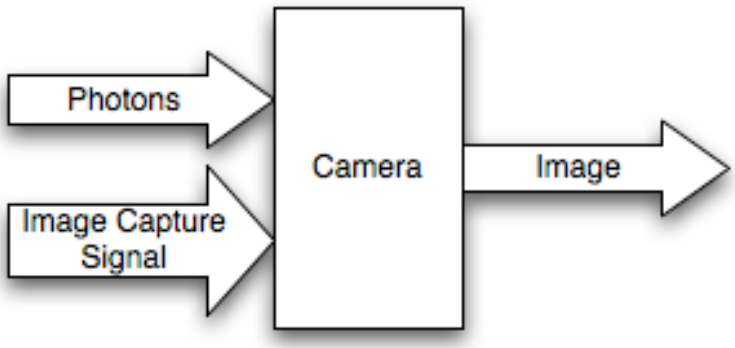


Figure 3: Camera Subsystem

2. Laptop and Software Subsystems

The laptop runs the software necessary for system operation. The laptop sends a signal out via USB to instruct each of the cameras to take a picture. The resulting images are then sent to the laptop via USB. The software on the laptop does the necessary calculations and makes a decision. Once the software has made a decision, signals are sent via a serial interface instructing the robot to move in the appropriate direction. The signals received from the robot contain information regarding robot motion and will be used during calibration mode. The user inputs and outputs are also used during calibration mode.

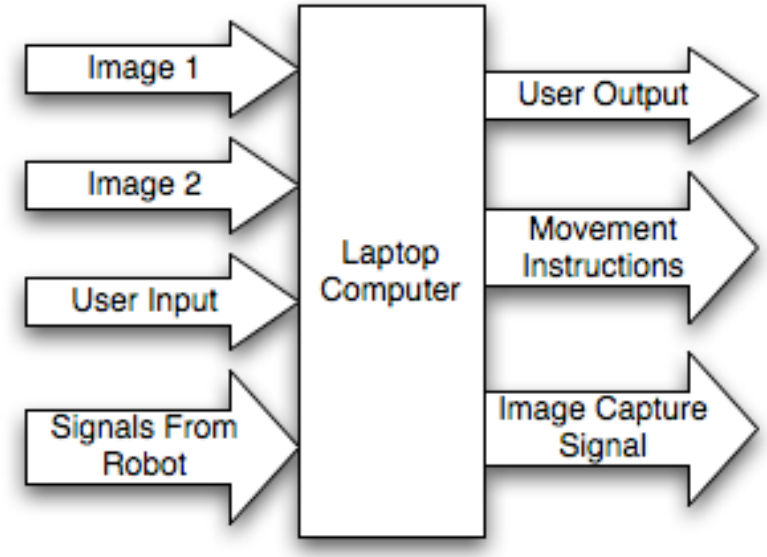


Figure 4: Laptop Computer Subsystem

3. Robotic Platform Subsystem

The ActivMedia robotic platform uses a software package called Aria. With this software, the robot can be controlled using single letters as instructions. For example, to move the robot directly forward, the letter d is outputted from the laptop via the serial interface. The microprocessor on the robotic platform analyzes the movement instructions signal and then sends signals to the motors to turn the wheels in the appropriate directions. The robot sends out other signals via the serial interface, however they are not relevant to this project. The robot functions the same in both modes.

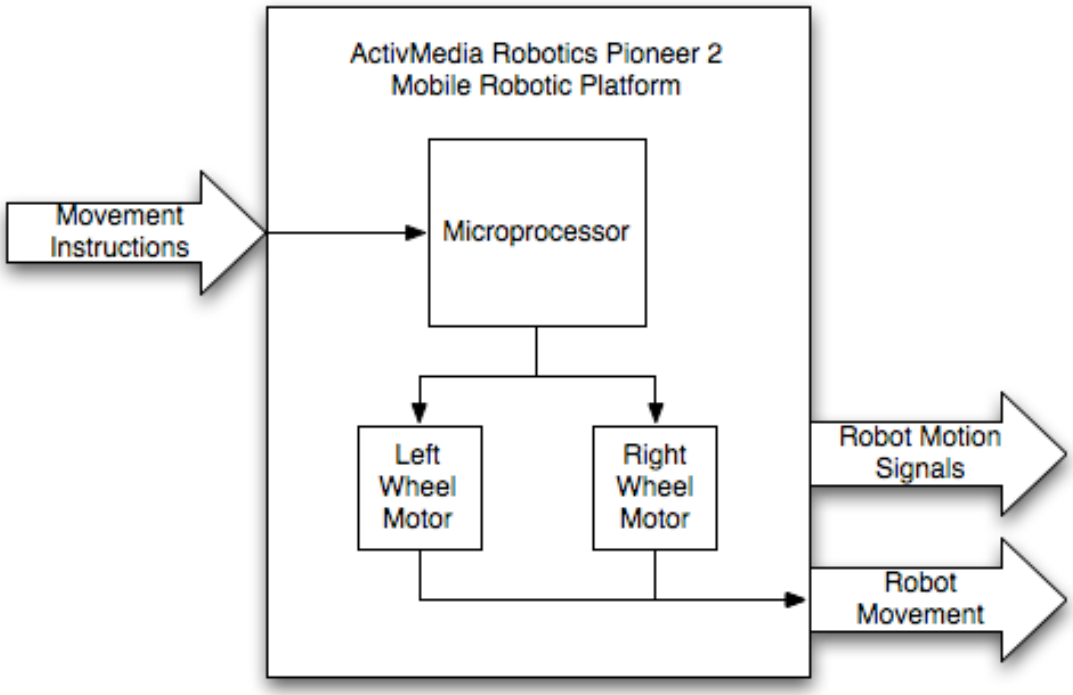


Figure 5: Robot Subsystem

B. Modes of Operation

There are two modes of operation, calibration mode and navigation mode. Each of these modes will be described in the following sections.

1. Calibration Mode

This mode will be used to ensure that the subsystems are setup and working properly. The user will initiate all of the steps performed in calibration mode manually. To ensure that the lighting is sufficient, a preview window of each camera will be opened in Matlab. The user will observe the preview and determine if the lighting is appropriate. While these preview windows are open, the user will also manually focus each camera if necessary. Lastly, the user will send various motion commands to the robot to ensure that it moves within specifications. If movements are not within specifications, appropriate changes will be made to the software to compensate for any discrepancies. When calibration is complete, the mode will be changed to navigation.

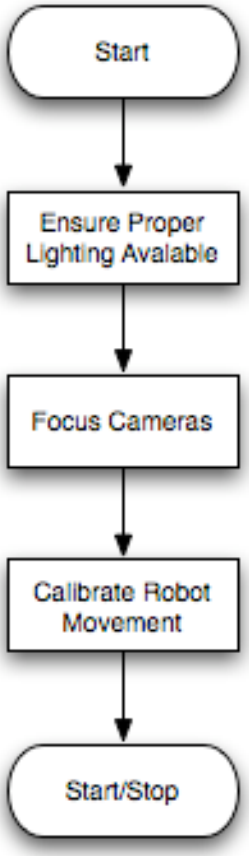


Figure 6: Calibration Mode Flow Chart

2. Navigation Mode

Navigation mode is the primary mode of operation. The software flow for this mode is shown in Figure 7. The first step is to retrieve the images from the cameras. The software sends a trigger to initiate this task. A signal is sent to each camera to capture an image. The image is then stored in the Matlab workspace as an image array. Since the robot is stationary, the two images can be captured in succession. Unless supported by the cameras, Matlab is unable to process two image captures simultaneously. The Logitech Buddy Cams do not support this feature. After the images are captured, they are straightened and cropped so that the objects lie on the same horizontal lines. Once the images are aligned, the software uses them to generate a three dimensional map. The edges of the objects will be detected first. The software will use color space correlation to determine which edges belong to each object. The software will then use the pinhole camera model to calculate the robot's distance from each of the obstacles.

Once the system computes the necessary distances, it will need to determine which way to move. The software will have the robot's physical dimensions hard-coded. The gaps between obstacles will be calculated and compared to Navbot's actual size. The system will determine where a large enough gap is and then move so that it is inline with the gap and facing it. From here, the robot will move forward. If the robot is more than one meter away from the closest obstacle, it will move the appropriate distance to bring it within one meter. From there the robot will move 30 centimeters at a time, reanalyzing its situation after each movement.

The system will continue through this process until it has determined that the navigation is complete. This decision will be based on a timer. A predetermined navigation time will be set and the robot will cease navigation once the time has expired.

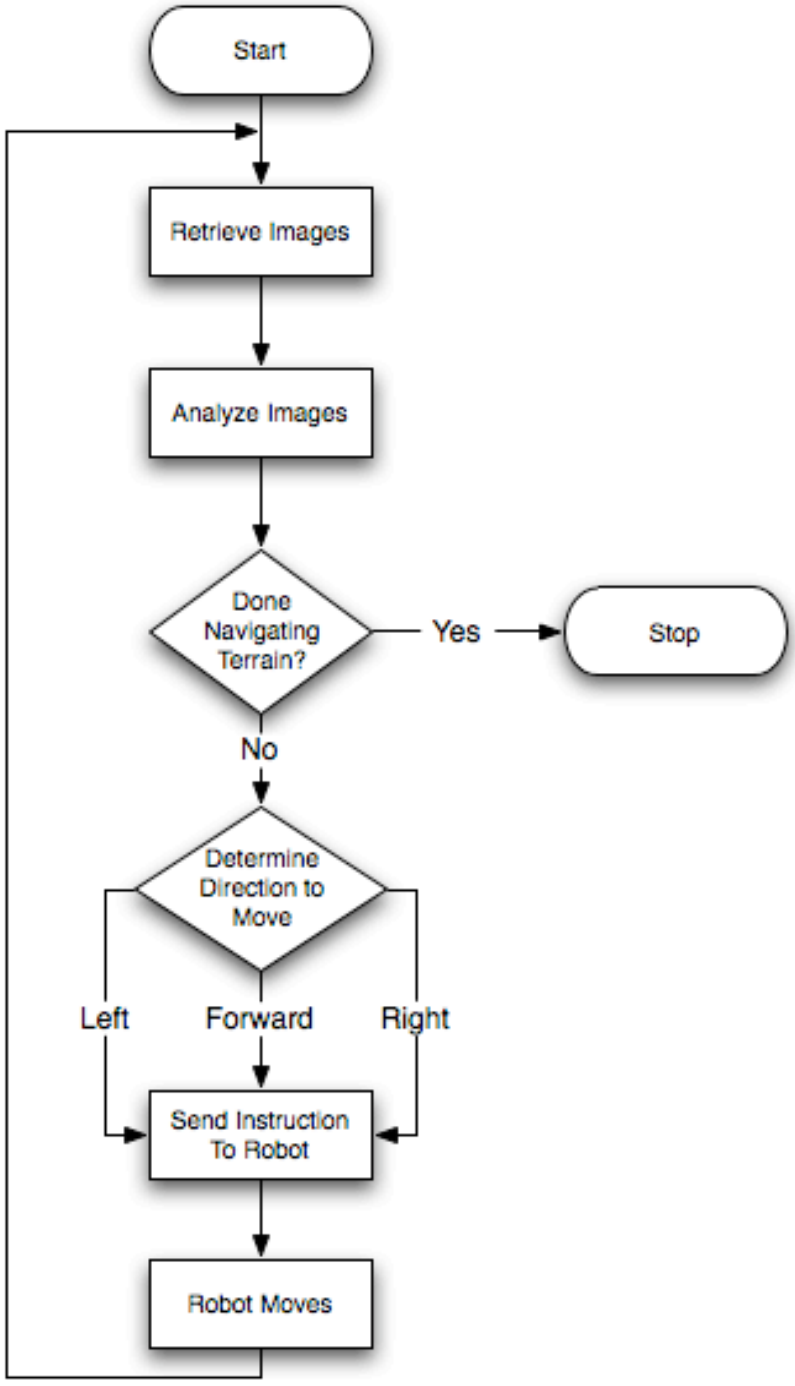


Figure 7: Navigation Mode Flow Chart

III. Subsystem Requirements

Each of the three subsystems described above are expected to perform within certain specifications. The specifications for each subsystem will be described in the following sections. Summaries of the specifications are also given in Tables C1 through C4 in Appendix C.

1. Camera Requirements

Both camera systems are the same therefore the requirements for each camera are identical. As depicted in Figure 3, the cameras detect photons and output a digital image. The stereoscopic imaging calculations are based on color correlations between the two cameras. In order to avoid false matches while correlating the images, the distance in color space between the colors seen by each camera shall not exceed 50 out of 255. In order to achieve the speed requirements, the cameras shall output images at a resolution of 320x240. A resolution greater than this will result in more calculations, therefore slowing down the system. The cameras shall be focused manually to avoid complications with any auto focus software or circuitry. The system begins making navigation decisions once it is within one meter of the closest object. To guarantee the system does not collide with any obstacles, the depth of field from the camera shall be at least three meters. To ensure proper light exposure, the camera shall have a minimum aperture of *f*/32. The focal length is the measure of how strongly the lens focuses light. There is no requirement for focal length, however it will need to be known for the distance calculations. The cameras shall not have any on board software or circuitry that is used for face tracking. Due to the available ports on the laptop computers, the cameras shall interface with the laptop computer via USB.

2. Laptop and Software Requirements

The laptop computer will be mounted on top of the robotic platform, as shown in Figure 2. It will perform all of the calculations and generate control signals. It grabs images from the cameras for analysis. As stated in the camera requirements, USB interfaces shall be used. Since there are two cameras, the laptop shall have at least 2 USB ports. The laptop will also be interfaced with the robot platform. The robot requires a serial interface. Therefore, the laptop shall have at least one serial port. The laptop shall also have the Mathworks Matlab package installed with the Image Processing and Image Acquisition toolboxes in order to perform the software tasks.

The goal of the NavBot project is to navigate through a terrain as quickly as possible. To achieve this goal, the software must be quick and efficient. There are five blocks to the software subsystem as shown in Figure 8. The first step shall only take about one half of one second. Blocks two, three, and four shall require less than 15 seconds to complete. The final block shall also require less than half of one second. The distance calculations need to be very accurate to avoid colliding with any obstacles. The software shall be capable of detecting edges of objects to within 1cm of accuracy.

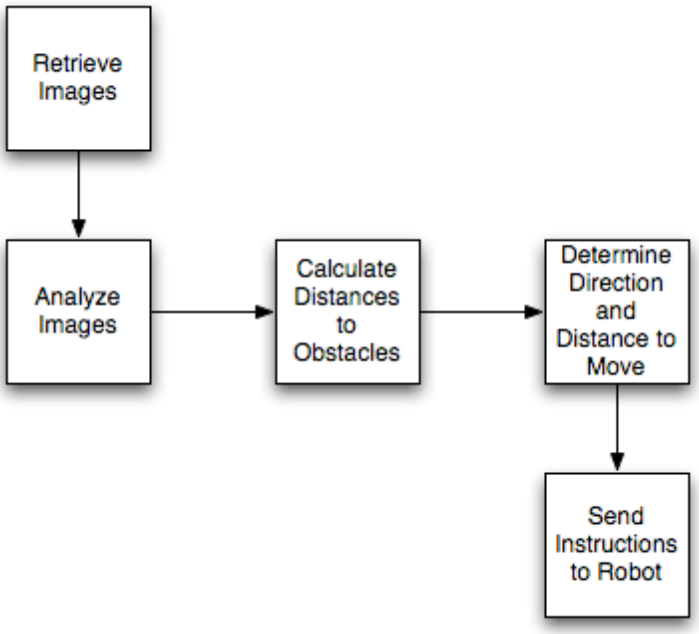


Figure 8: Software Block Diagram

3. Robotic Platform Requirements

The robot needs to be able to maneuver with speed and precision. The body of the robot is 44 cm x 38 cm x 22 cm. It is capable of traveling 1.6 meters per second on flat surface with no payload. For each meter of travel, the robot shall be within 4 cm. Should the robot need to turn around, its turning radius is 32 cm swinging from a stationary wheel. The robot's wheels have a diameter of 32cm. The motor gear ratios are 38.1:1 and each motor has a 500 tick rotary encoder. These specifications were taken from the robot manufacture's website, www.activmedia.com.

IV. Results

This year, major progress was made on four areas of the project. The pinhole model was implemented and the conversion factor was found experimentally. Using the conversion factor information, image correlation was used to generate a distance calculation function. A manually adjustable image capture and correction function was also created. Finally, progress was made towards implementing edge detection to automate the distance calculations.

A. Pinhole Model and Conversion Factor

Stereoscopic imaging gives the ability to calculate the three dimensional position of an object in Cartesian coordinates using the position of the object in images captured by two cameras. The cameras can be oriented in any fashion as long as equations are derived for that specific orientation. For this project, the cameras are side by side and 10 cm apart, which is denoted as d in Figures 9 and10. A set of equations can be derived using the pinhole camera model from which an object's X, Y, and Z coordinates can be determined. However, only X and Z need to be calculated for this project since the terrain is two-dimensional. Figure 9, shows the setup of the pinhole model for side-by-side cameras and Figure 10 shows a view from the +y axis. Using similar triangles, the relation between X, Z, X_R, X_L, and d can be found and is shown in Equations 1(a) and 1(b). Currently, progress has only been made toward determining Z. X will need to be found by future teams in order to complete the project.

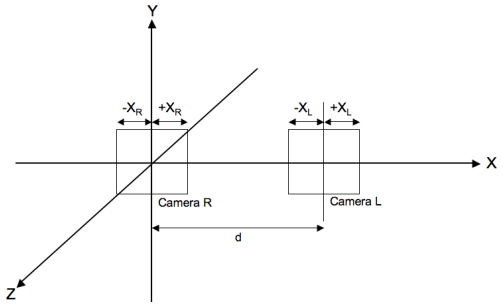


Figure 9: Pinhole model system setup for side-by-side cameras

$$Z = \frac{d}{(X_L - X_R)} * \text{Conversion Factor}$$

Equation 1(a): Equation for Z

$$X = \frac{-X_R * d}{(X_L - X_R)}$$

Equation 1(b): Equation for X

Equation 1(a) shows the relationship between Z and the value of $X_L - X_R$ and its validity is apparent from Figure 10.

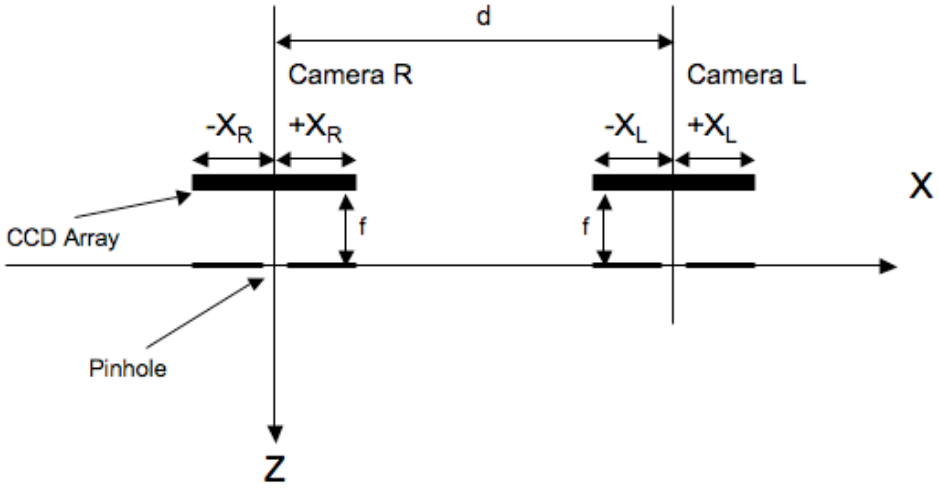


Figure 10: View of pinhole model from +y axis

This figure shows that the pinhole plane is in front of the CCD array in the XY plane. The pinhole is f meters in front of the array. If, for example, an object is on the Z-axis, then X_L is positive and X_R is zero. This yields a positive value for Z as expected. .

As shown in Equation 1(a), there is also a conversion factor involved in the calculation. The distances in the equations are measurements in meters. However, when the images are captured and imported into Matlab, they only contain pixel information. The conversion factor is used to convert the pixel information to meters. The value of f is typically available as a specification for the cameras. However, this value was not available for the Logitech Buddy Cams. Since f is a constant, it was moved into the conversion factor. Both the conversion factor and f were found experimentally.

To find the conversion factor experimentally, sets of images were taken at distances of 2 meters down to 0.5 meters in 10 cm decrements. The images were manually analyzed in Matlab to determine $X_L - X_R$ for each distance, from which the conversion factor can be calculated. Table 1 shows the data and calculations from this experiment.

Table 1: Conversion factor experiment data

Distance From Robot (m)	X_{left} (pixels)	X_{right} (pixels)	X_L (pixels)	X_R (pixels)	$X_L - X_R$ (pixels)	d (m)	Conversion Factor (pixels)
2.0	186	122	21	-43	64	0.10	**1280**
1.9	171	105	6	-60	66	0.10	**1254**
1.8	190	123	25	-42	67	0.10	**1206**
1.7	188	119	23	-46	69	0.10	**1173**
1.6	181	109	16	-56	72	0.10	**1152**
1.5	169	94	4	-71	75	0.10	**1125**
1.4	179	102	14	-63	77	0.10	**1078**
1.3	182	102	17	-63	80	0.10	**1040**
1.2	178	94	13	-71	84	0.10	**1008**
1.1	182	93	17	-72	89	0.10	**979**
1.0	179	83	14	-82	96	0.10	**960**
0.9	184	81	19	-84	103	0.10	**927**
0.8	183	70	18	-95	113	0.10	**904**
0.7	188	60	23	-105	128	0.10	**896**
0.6	188	42	23	-123	146	0.10	**876**
0.5	184	10	19	-155	174	0.10	**870**

The results show that the conversion factor is not constant. This is probably due to deviations of the actual camera optics from the pinhole model. At first inspection, the conversion factor seems that it may be linear across the range. To determine how linear the conversion factor is, it was plotted against the actual distance. The following two figures are conversion factor versus distance. Figure 11 has a linear trend line while Figure 12 has a polynomial trend line.

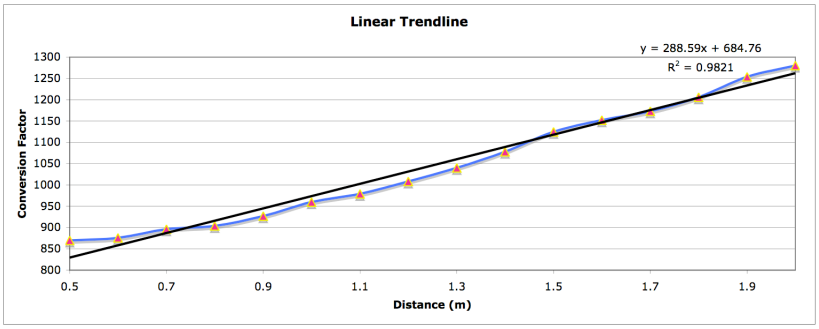


Figure 11: Conversion factor vs. distance with linear trend line

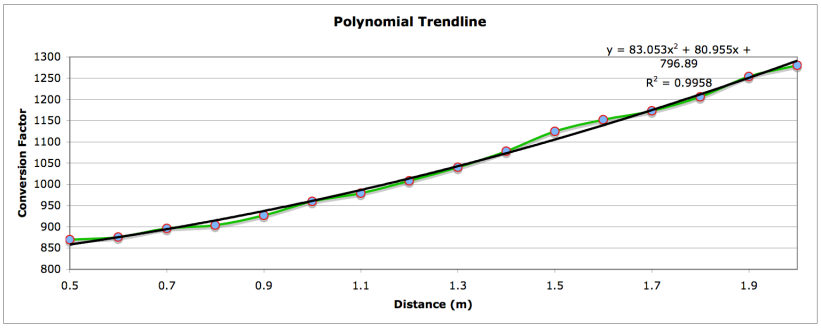


Figure 12: Conversion factor vs. distance with polynomial trend line

It is clear that the data is not perfectly linear. In fact, it almost matches perfectly with the polynomial trend line. However, R^2 value for the linear trend line is still 0.9821, which is very close. For the requirements of this project, it is acceptable to assume the conversion factor is linear. This will greatly reduce the complexity of the distance calculation algorithm.

It was decided for Z value calculations that the best solution to the non-constant conversion factor was to develop an iterative algorithm. The function firsts calculates an intermediate value for Z using a constant conversion factor. It then chooses the correct value from a lookup table. If the value is not in the table, the function interpolates. The next step was to decide what to use as the constant conversion factor to calculate the intermediate Z value. Table 2 shows the intermediate values of Z if calculated using four different conversion factors.

Table 2: Conversion factor intermediate Z values

Z (m) Actual	Z (m) CF = 1	Z (m) CF = 870	Z (m) CF = 1280	Z (m) CF = 1000
2.0	0.00156	1.359	2.000	1.563
1.9	0.00152	1.318	1.939	1.515
1.8	0.00149	1.299	1.910	1.493
1.7	0.00145	1.261	1.855	1.449
1.6	0.00139	1.201	1.778	1.389
1.5	0.00133	1.160	1.707	1.333
1.4	0.00130	1.130	1.662	1.299
1.3	0.00125	1.088	1.600	1.250
1.2	0.00119	1.036	1.524	1.190
1.1	0.00112	0.978	1.438	1.124
1.0	0.00104	0.906	1.333	1.042
0.9	0.00097	0.845	1.243	0.971
0.8	0.00088	0.770	1.133	0.885
0.7	0.00078	0.680	1.000	0.781
0.6	0.00068	0.596	0.877	0.685
0.5	0.00057	0.500	0.736	0.575
Range	**0.00099**	**0.85938**	**1.26437**	**0.988**

The range row in Table 2 is the distance calculated at 0.5 meters subtracted from the distance at 2.0 meters. The range is the greatest when the conversion factor is equal to 1280. Using a value that allows for a greater range will all for more accuracy when interpolating. The values are more spread out, so it is less likely that an intermediate value will be interpolated between the inappropriate values in the look up table. Therefore, the constant conversion factor will initially set as 1280. Appendix C contains the interpolation code.

B. Color Correlation and Distance Calculations

The situation depicted in Figure 10 is a single point in space illuminating single pixels in each camera. However, in an actual operating environment, many pixels will be illuminated and it is necessary to correlate the pixels in each camera to the edges of the obstacles. This is a difficult process in general. In the case of color obstacles, color correlation can be used and was chosen for this project.

Color correlation was developed and used in Nick Patrick's masters degree project. A description of color correlation from Patrick's paper, *Stereoscopic Imaging*, can be found in Appendix A. Patrick included Matlab code to perform color correlation for two images. This code was slightly modified for use with the NavBot project. The most notable change was in regards to the amount the right image is shifted over the left. Patrick's test images did not have any correlations greater than 60 pixels apart. Depending on the distance the images are taken at, the shift index can be anywhere between 60 and 175. The code was modified to accommodate the larger shift indices.

Color correlation is based on a calculation of distance in colorspace for two pixels. The equation Patrick used to calculate colorspace distance is shown in Equation 2. The variables relate to the R, G, and B values of the pixel in each of the images.

$$dist = \frac{\sqrt{(RL - RR)^2} + \sqrt{(GL - GR)^2} + \sqrt{(BL - BR)^2}}{3}$$

Equation 2: Distance in colorspace

Patrick's code was used as a foundation for the distance calculation software. The color correlation software provides a matrix containing the shift index, X_L - X_R, for each pixel in an image. Code was written to calculate a distance using based on the pinhole model equations. This code utilizes the shift indices from Patrick's code.

The software starts by reading the left and right images. It then slides the right image across the left and records the minimum slide for each point. The index of matching points is determined and saved in an array called min1Seq. This array is then plotted as a disparity map. A threshold is applied to the minimum value for each pixel. If the disparity is greater than the threshold, a value of -1 is recorded. The appropriate shift index is chosen based on the x and y values determined by the edge detection software. This shift index is used to calculate an intermediate value for Z using a constant conversion factor of 1280. The intermediate Z is used to interpolate the actual Z from a lookup table. A flowchart of the complete function is shown in Figure 13. The code is provided in Appendix C.

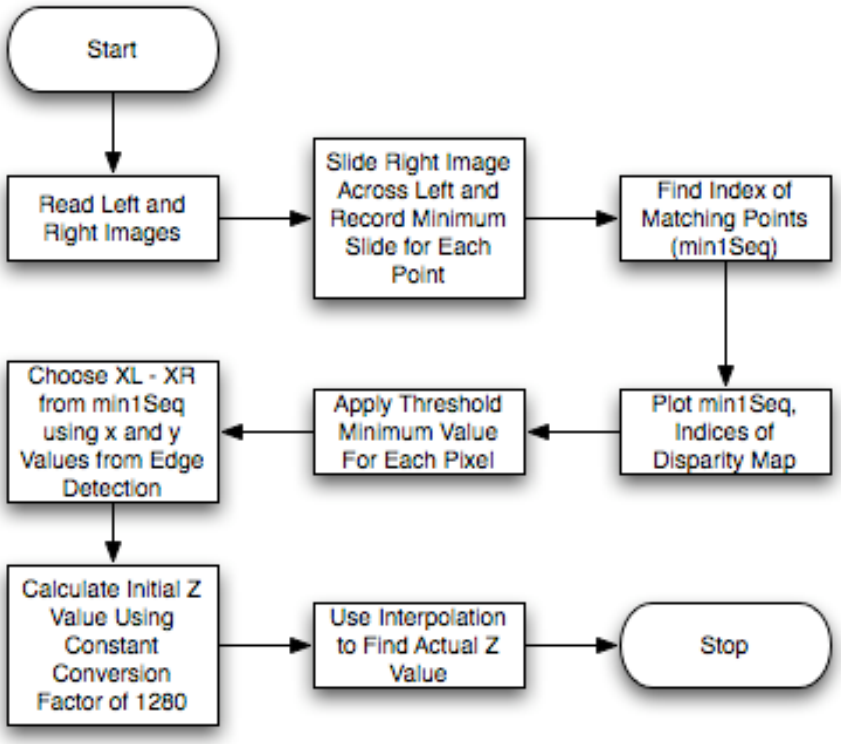


Figure 13: Color correlation and distance calculation software flowchart

This software was used on a couple of sets of images with partial success. The main problem was that the images used were of a sheet of black paper taped to a wall. This resulted in essentially flat images. The correlation software had difficulty correlating the images due to a lack of three-dimensional disparity between them. Future senior teams should design test obstacles with actual three-dimensional objects, such as colored boxes.

C. Image Capture and Correction

A critical portion of this project was to capture good images. Without a good set of images, it is difficult to correlate pixels. A good set of images should have little to no distortion. There were primarily three types of distortion observed in the laboratory.

The first type of distortion observed was color distortion. Originally this was attributed to poor lighting conditions. Later it was thought to be due to the poor quality web cameras. Finally, it was found that the color distortion was a product of the way Matlab was capturing images. Matlab initialized the cameras every time it took a picture. This resulted in inconsistent gamma adjustments and white balance levels. If higher end web cameras were implemented (with face tracking) the face tracking would also automatically be enabled upon initialization (every time a picture was taken). To remedy the problems that resulted when using Matlab to capture images, Dorgem was used. Dorgem is an open source web camera capture program. Dorgem can save pictures to a compressed JPG or run in the background of the computer as a web server. (Appendix D contains a tutorial on how to set up multiple cameras for Dorgem). After Dorgem was implemented, constant gamma levels and white balance levels were selected based on the lighting conditions of the room. If these levels were selected poorly, pictures would appear yellow or washed out. This was acceptable, as the primary goal was to improve consistency between pictures, not accuracy.

The second type of distortion observed was camera alignment. Each camera had a slightly different angle with respect to the other one. The pitch, yaw, and skew of each camera were slightly different, resulting in different placing of the horizon. This distortion made pixel correlation virtually impossible. The pictures were corrected in Matlab with the imrotate command, and some image cropping code. This code is given in Appendix E. It should be noted that this code was using static variables and was not working perfectly, however it was functional**.**

The third type of distortion was lens distortion. It was noted that the pictures had a slight fisheye distortion. Several tests were conducted, and it was decided that this distortion was minor and would consume too much time to correct. It was suspected that some kind of lens distortion was causing variation in the conversion factor, which was corrected with iteration.

D. Edge Detection

Edge detection was used to locate objects. Once a full distance map was created, the computer needed to locate the objects in the distance map, in order to avoid them. Edge detection was used as a way of intelligently guessing what, in the images, were objects. Once the computer had a guess as to what was an object, it could extrapolate the location of the object, and find the distance to the object from the cameras, using the distance calculation software. Edge detection was selected for object detection primarily because all the objects used had straight vertical edges (such as a box). In addition, all of the objects were solid colors that differed from the background. The code shown in Appendix F would need to be modified slightly to implement the max command in order to extrapolate the location of the object.

The results for edge detection were mixed. Insufficient time was allotted toward this portion of the project to complete it. The results primarily varied with the picture taken. If a set of images had a very distinct set of vertical edges, the program could easily find the edges. Figure F4 shows clearly detected lines. The program would need to be tweaked to detect less distinct lines. Two ways to change the matlab script would be: 1) implement a more sensitive vertical edge filter or 2) change the threshold of the enhanced image (Figure F3). If the filter and enhanced image are too sensitive, noise will be added.

5. Conclusion and Recommendations for Further Work

The four portions of the project discussed above are all critical to achieving the final goals of this project. The image capture and correction software is required for the color correlation to function properly. If the images are not perfectly aligned, the objects will not correlate correctly and false results will be generated. The color correlation provides crucial data necessary to perform the distance calculations. Finally, edge detection is a process that aims to determine which pixels of an image to use for distance calculations.

Recommendations to Future Senior Teams:

- Develop software to determine distances on X axis
- Use three-dimensional objects for testing
- Obtain and implement high quality cameras or a single camera on a rail
- Develop a graphical user interface for calibration
- Integrate software with robot movement

V. Equipment List

1. Hardware
    - 2 Logitech Buddy Cams
    - Gateway Laptop
    - ActivMedia P2-DX Robotic Platform
2. Software
    - Dorgem
    - Mathworks Matlab
    - Image Processing Toolbox
    - Image Acquisition Toolbox

VI. Related Patents

A patent search resulted in a number of patents related to the NavBot project. The patent number and a short description of each related patent is shown in Table 3.

Patent Number	Brief Description
6396397	Vehicle imaging system with stereo imaging
5675377	True three-dimensional imaging and display system
4709263	Stereoscopic imaging apparatus and methods
6775614	Vehicle navigation system using live images
6151539	Autonomous vehicle arrangement and method for controlling an autonomous vehicle
6751535	Travel controlling apparatus of unmanned vehicle
5812269	Triangulation-based 3-D imaging and processing method and system
6661449	Object recognizing apparatus for vehicle and the method thereof

Table 3: Related Patents

VII. Bibliography

Patrick, Nicholas. Stereoscopic Imaging.

Crombie, Brian and Zivney, Matt. Project Proposal. Bradley University.

"Aperture." Wikipedia. 11 Dec. 2006. 13 Dec. 2006 <http://en.wikipedia.org/wiki/Aperture>.

"Focal Length." Wikipedia. 8 Dec. 2006. 13 Dec. 2006 <http://en.wikipedia.org/wiki/Focal_length>.

Appendix A

Subsystem Specification Summary Tables

Table C1: Camera Specifications

Color	16 bit
Focus	Manual
Interface	USB
Aperture	f/32
Resolution	320x240
Depth of Field	3 m
Face Tracking	No

Table C2: Laptop Specifications

USB Ports	At least 2
Serial Ports	At least 1
Processor Speed	1.6 Ghz Pentium M
Memory	512 Mb
Software	Matlab Image Acquisition Toolbox Image Processing Toolbox

Table C3: Software Specifications

Retrieve Images	0.5 seconds
Analyze Images	5 seconds
Calculate Distances	5 seconds
Determine Direction to Move	5 seconds
Send Instructions To Robot	0.5 seconds
Distance Calculation Accuracy	1 cm

Table C4: Robotic Platform Specifications

Height	38 cm
Length	44 cm
Width	22 cm
Max. Speed	1.6 m/s
Turning Radius	32 cm
Wheel Size	16.5 cm dia
Motor Gear Ratios	38.2:1
Motor Rotary Encoders	500 tick
Movement Accuracy	4 cm per 1 m of travel

Appendix B
Colorspace Correlation

Color space correlation is a technique used in stereoscopic imaging to correlate images taken from two cameras. The following information was taken directly from Nick Patrick's paper *Stereoscopic Imaging*. The NavBot system will utilize this technique.

> "In order to determine the image disparity for each point, the correlation between one pixel and another needs to be defined. Since the images are supplied as RGB matrices, I can determine the distance in the color space for each color and sum each difference. The norm function is used for this purpose.
>
> $$dist = \frac{\sqrt{(RL - RR)^2} + \sqrt{(GL - GR)^2} + \sqrt{(BL - BR)^2}}{3}$$
>
> The square root term in the distance function could be eliminated to make the calculation more efficient; however it is useful in this case for keeping the range of distance between 0 and 255. The terms in the distance function correspond to the intensity of each pixel in each color, i.e. RX, GX or BX. The pixel from each diagram is referred to by XL or XR for the left or right figure.
>
> In order to visualize the color space distance function, it is useful to imagine each color to be an axis. The distance between two points is then defined by the equation above. The image in Figure B1 shows two points in the color space dimensions. The coordinates of the points are R(230), G(100), B(40) for the red point and R(100), G(230), B(200) for the blue point. The distance between these points is 203.3.

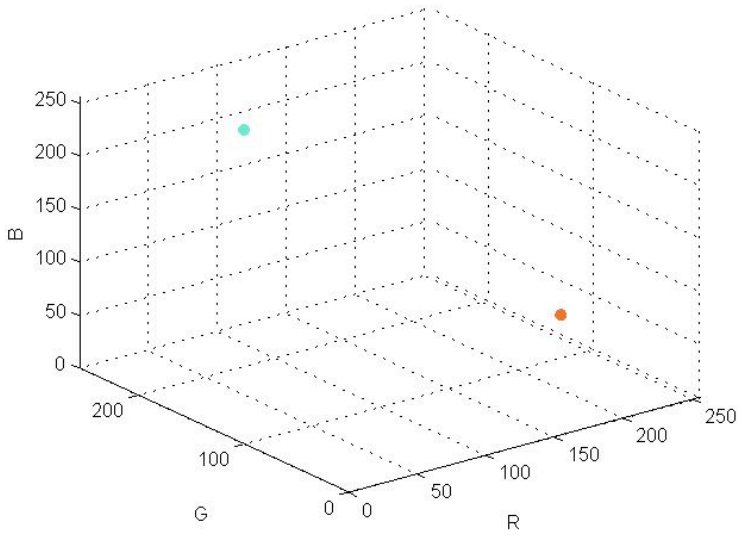


Figure B1: Two points in color space

Now that distance between two colors is defined, this function has to be applied to find corresponding pixels between two figures. Since each figure is aligned vertically, each pixel on the left figure only needs to be checked against the pixels on the same scan line on the right figure. The method that was used to determine the minimum color distance is to slide the right image across the left image from left to right. In order to ensure that noise does not cause too many false positives, each subtracted image is filtered by a small Gaussian filter from the MATLAB image processing toolbox. The filter causes each color distance to be affected by the color distance of its neighbors.“

Appendix C

Color Correlation and Distance Calculation Matlab Code

calc_dist.m

```
%Color Correlation by Nick Patrick
%Distance Calculation by Nick Wlaznik

% Read Images

%fname1 = 'l6crop.jpg'; % Left Image
%fname2 = 'r6crop.jpg'; % Right Image

%im1 = double(imread(fname1,'jpg'));
%im2 = double(imread(fname2,'jpg'));

im1 = double(saveL);
im2 = double(saveR);

[m,n,o]=size(im1);
j=1;

h=fspecial('gaussian',3,1.5);

minMap = repmat(255,m,n);

% slide 2nd image across 1st image and record minimum slide
for each point
for i=1:100 % for i, slide right image right 1 pixel,
subtract images
    % slide right image right across left image
    newRight = im2;
    newRight(:,1+i:n,:)=im2(:,1:n-i,:);
    subImg = uint8(abs(round(sqrt(im1.^2-newRight.^2))));
    subImg(:,1:i,:)=255;
    subImg = imfilter(subImg,h,'conv');
    figure(1);
    image(subImg);
    drawnow;
%     pause;

    % record each difference image
    minMap(:,:,i) = sum(subImg,3)/3;
end

min1Seq=repmat(255,m,n);min1Val=minMap(:,:,1);
npix = m*n;
```

```
% Find index of matching points
for j=1:100
    ind=find(minMap(:,:,j)<min1Val);
    min1Seq(ind)=j;
    min1Val(ind)=minMap(ind+(j-1)*npix);
end

% Threshold the minimum value for each pixel
min1Seq(min1Val>90)=-1;
min1Seq(min1Seq>100)=-1;

% Plot min1Seq, indices of disparity image
figure(2);
imagesc(min1Seq);

% Optional
colormap('gray');

%extract shift index from color correlation results
%x and y are provided by edge detection
xlxr = min1Seq(y,x,:)

%calculate intermediate Z value using constant conversion
factor
z_init = (.10/(xlxr))*1280

%Actual Z values
t = 0.5:.1:2;

%interpolation values
p = [0.7360 0.8770 1.0000 1.1330 1.2430 1.3330 1.4380
1.5240 1.6000 1.6620 1.7070 1.7780 1.8550 1.9100 1.9390
2.0000];

%interpolate to find final Z value
z = interp1(p,t,z_init)
```

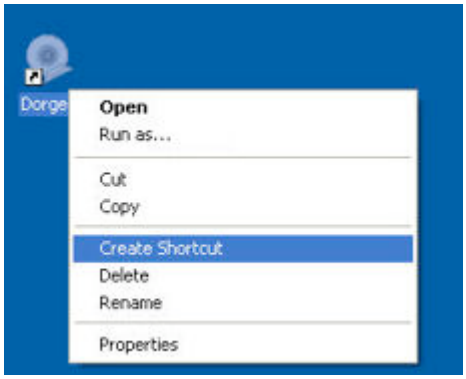
Appendix D

How to set up multiple web cams in Dorgem

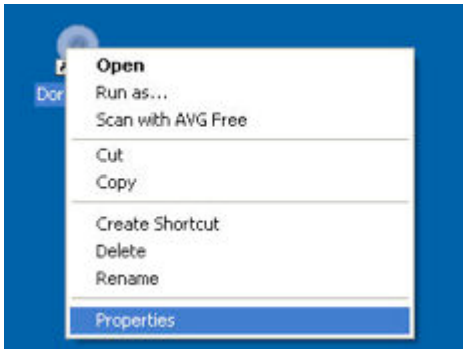
This tutorial was taken directly from the simplehelp.net website. The URL of the website is http://www.simplehelp.net/2006/09/27/how-to-use-your-pc-and-webcam-as-a-motion-detecting-and-recording-security-camera/

If you can get more than one Webcam to work in Windows (not always an easy task, esp. if they're Labtec), Dorgem can support them all. You don't need to install another copy of Dorgem, but you do need to start each instance differently. To do so, follow the steps below..

1. Right-click on your current Dorgem desktop icon and select **Create Shortcut**



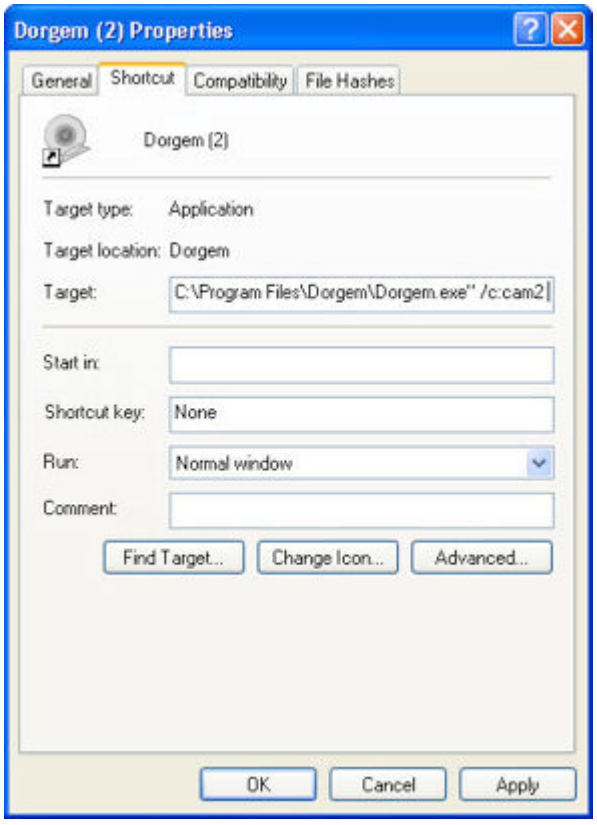
2. You should see a new **Dorgem (2)** icon on your desktop - right-click on it this time, and choose **Properties**



3. Make sure the **Shortcut** tab is selected, and change the **Target:** to:

   "C:\Program Files\Dorgem\Dorgem.exe" /c:cam2

   If you installed Dorgem to somewhere other than the default location, make the appropriate changes.



4. Click **Apply** and then return to your desktop. Double-click the Dorgem (2) icon. When it launches, notice that it's titled **Dorgem (cam2)**. Repeat the same steps you did for the first camera (though choose a different file name to save the image as).

Appendix E
Image Capture and Correction Matlab Code

pic.m

```
%function pic creates the following
%frameL frameR: the right and left frame, unedited captured
from dorgem
%saveL saveR: the cropped and rotated images

%web server ~dorgem
frameL=imread('http://localhost:8081/');
frameR=imread('http://localhost:8082/');

%%%%%
%rotation
%our static variables go here
%we could possibly find how far the images need to be
rotated by using
%a function similar to the one I found on the web called
'croockedness'
%but for the time alloted for this project I was unable to
get auto
%rotation completed
%%%%%%

rotateLeft=.6;
rotateRight=-1.5;

saveL=frameL;
saveR=frameR;

saveL=imrotate(saveL,rotateLeft,'bicubic');
saveR=imrotate(saveR,rotateRight,'bicubic');


saveL=saveL(10:280, 10:340, 1:3);
saveR=saveR(15:285, 15:345, 1:3);

figure,imshow(saveL);figure,imshow(saveR);
```

Appendix F
Edge Detection Matlab Code

edge.m

```
close all;
%closes all the open figure windows
```

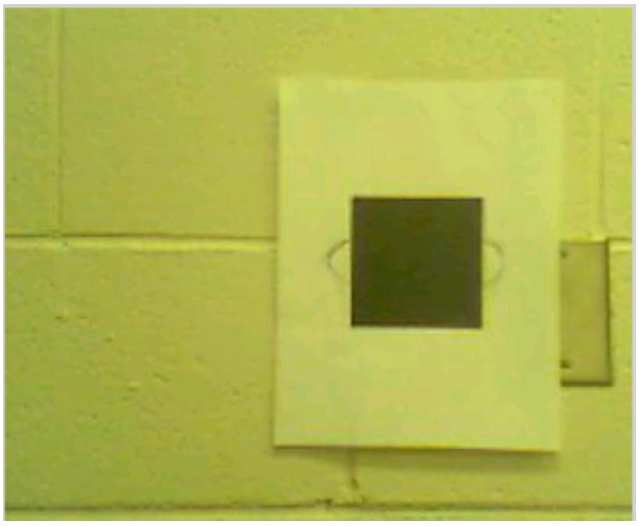


Figure F1: Original picture

```
vertFilt = [-1,2,-1;-1,2,-1;-1,2,-1];
%sets up the vertical filter
%if we change 2 to a larger number (3) the image will
%have sharper edges
%to do a horizontal filter we would use=
%[-1 -1 -1;
% 2  2  2;
%-1 -1 -1]
%this is what it looks like after the vertical filter
%is applied
```

Figure F2: Picture after vertical filter

```
fsaveLV = imfilter(saveL,vertFilt);
%our image after we apply the vertical edge detection

for k=1:3 %we have 3 components RGB
     for j=1:271 %image height
          for i = 1:331 %image width

               %this value sets up the threshold
               if fsaveLV(j,i,k) > 50%thrhld7g

     fsaveLV2(j,i) = 255;
               else
                    fsaveLV2(j,i) = 0;
               end;
          end;
     end;
end;
```

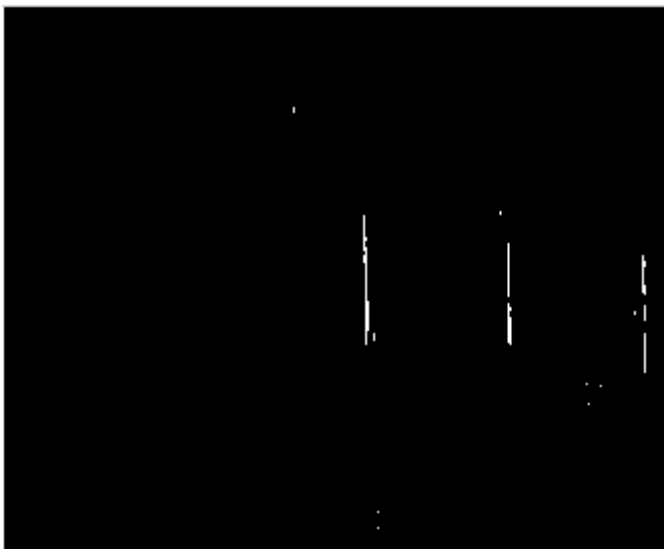


Figure F3: Picture after enhancing the filtered image

```
fsaveRV = imfilter(saveR,vertFilt);

for k=1:3
     for j=1:271
          for i = 1:331
               if fsaveRV(j,i,k) > 50%%thrhld7g
                    fsaveRV2(j,i) = 255;
               else
                    fsaveRV2(j,i) = 0;
               end;
          end;
     end;
end;

figure,imshow(fsaveLV)
figure,imshow(fsaveRV)

figure,imshow(fsaveLV2)
figure,imshow(fsaveRV2)

%now we transform it to a single x-y graph, by adding up
the
%the components along the x axis
fgraphL(1:331)=0;
fgraphR(1:331)=0;

for j=1:271
    for i = 1:331
        if fsaveLV2(j,i) == 255;
            fgraphL(i) = fgraphL(i) + 1;
```

```
        end;
    end;
end;
```

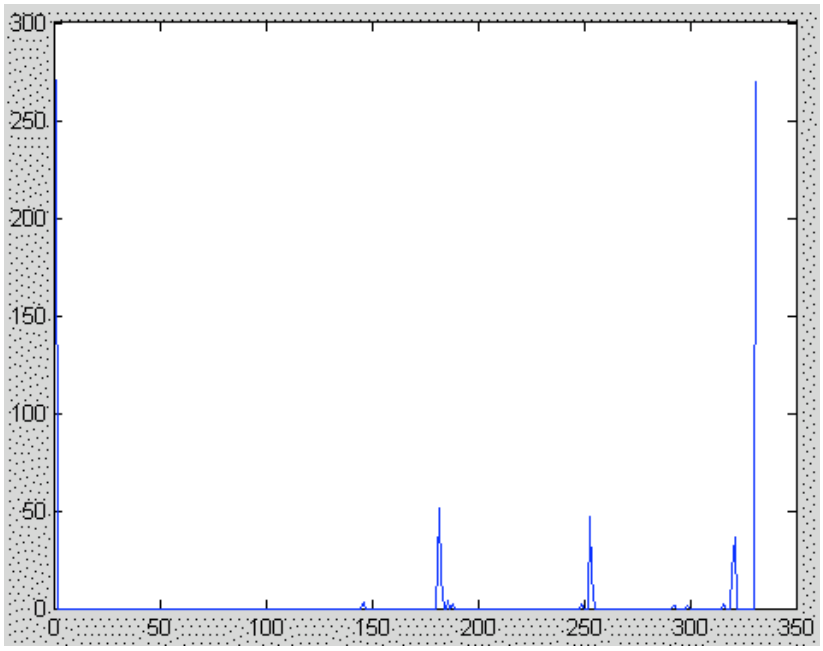


Figure F4: Result after

```
for j=1:271
    for i = 1:331
        if fsaveRV2(j,i) == 255;
            fgraphR(i) = fgraphR(i) + 1;
        end;
    end;
end;

%now that we have these we could use a max detection
%and directly output our max numbers to edges which
%can be used in distance calculations

figure,plot(fgraphL);
figure,plot(fgraphR);
```