Local Positioning System with Multiple Cameras

An Engineering Paper

Presented to

Junior Science, Engineering and Humanities Symposium

Maryville University

by

Connor Monahan

Sophomore

559 E Highway N of
Wentzville, Missouri
63385

April 8, 2014 – November 14, 2014

Dr Barr
Physics Teacher
Timberland High School

Acknowledgements

Hunter Park

Dr. John Barr

Vince Redman

David White

Bob Gaines

Cindy Gaines

Wentzville School District

Abstract

This paper presents the process of obtaining, filtering, and tracking targets to obtain the camera's relative position. The program used computer specific architecture features to optimize the program and provide consistent results immediately. First, images are obtained of the environment from 3 different infrared cameras in gray-scale with a field of view of 120 degrees. Next, the images underwent a series of image processing algorithms using the OpenCV programming libraries, such as morphological transformations and binary topological analysis, to identify the targets in the image. This converts the 1080p 1-channel matrix into a vector of points called contours. A series of tests were then used to remove potential noise and error from the data. The program then uses vectors and matrices as storage for calculating the positioning and distance. At this point the information from each camera was combined and utilized in a GPS style calculation was used to find the local position of the robot relative to the field. Finally, this information was used to solve for the relative pose of the robot by solving a system of non-linear functions. The results provide positioning and motion information of the robot relative to the static environment.

Table of Contents

1. Background

Computer vision is a rapidly evolving field with many different available applications of

the technology. Robotics is just one field among others, including biomedical imaging and surveillance [10]. An image can be represented as a matrix of pixels [5]. Many different types of features can be extracted from just one image based on the known environment of the image [9]. Many applications of robotic localization in the past have used radio-frequency or light beacons surrounding a field or area of operation [1,7]. These approaches have yielded highly accurate data in their respective implementations but they are not applicable to every circumstance [3,4]. A scenario permitting limited radio use and lacking the opportunity to modify the environment prohibit the use of a beacon setup. This algorithm allows use in such conditions based on the known descriptions and appearance of specific items in the environment, such as a house or tree [6].

Other previous uses involve the comparison of field objects with databases of images known to correspond to particular locations [2]. These can provide a generalized location finding, however they cannot account for the placement of the same object in multiple locations. They also do not provide a specific level of detail that can be used at short range.

This paper presents an approach that can use predefined features in camera images to calculate the positioning of a mobile robot without using techniques like radio frequency or ultrasound. Position is found from the relative position of each target using a circle combination algorithm.

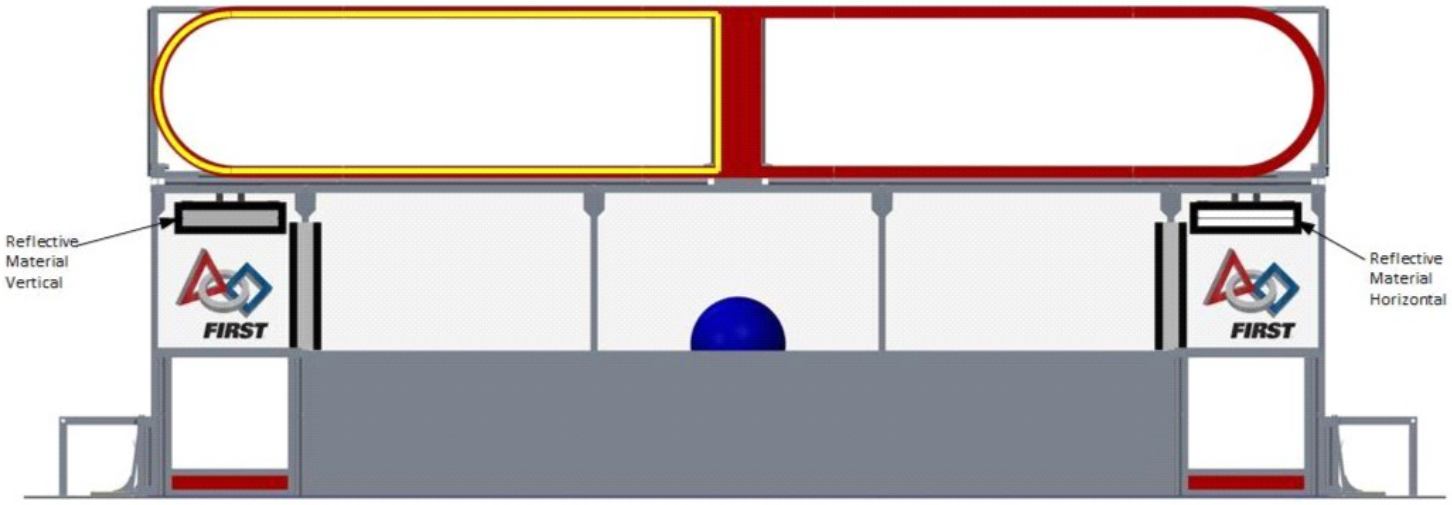
2. Introduction

Computer vision was applicable for the FIRST (For Inspiration and Recognition in Science and Technology) robotics competition this year. The solution needed to be able to find relative position of the robot on the field. Various options for our approach included tracking objects mounted on a wall and following moving objects on a field. The program was successfully able to expand upon these ideas by using these basic elements to find precise details, primarily the

robot's positioning relative to different environmental aspects.

The game's objective was to launch large plastic balls into targets situated at both ends of a field accessible by the robots (Fig 1). Under each target was a pair of horizontal and vertical pieces of retro-reflective tape that were visible from a robot using infrared lights. The size and positioning of each target were known. Each of the four corners of the field featured a pair of targets, as shown in Fig 1. The rectangular shape of the targets allowed a program to use various techniques to find positioning [8]. For the most accurate data, the location of as many targets as possible was necessary. It would not be necessary to accommodate for rotation as the height of the targets and the robot cameras would be held constant.

Fig 1. FIRST Aerial Assist field with Targets



3. Apparatus

3.1 Genius 120° Camera

This algorithm demanded a high quality camera with a large field of view and utilized three cameras. The cameras needed to have as close to 120 degrees of view in the horizontal direction in the goal of obtaining as much information as the environment as possible. A depth map was

not necessary due to the use of geometry in calculating the distance to each visible target on the field. The field of view was the primary reason for the use of these cameras. Theoretically, this would allow the robot to see all four corners of the field, and consequently all of the four target pairs. This increased the accuracy as the positioning could be confirmed with more sources of input data.

The cameras originally were standard color cameras intended for the purpose of videoconferencing. Like most other color cameras, they contained a rudimentary infrared light filter in between the lenses. In order for the algorithm to work correctly, the filter had to be removed, which was accomplished by removing the lens and cracking the portion of glass that contained the special filter. The reason for the IR filter removal is explained in section 3.4 which goes over the use of infrared light to achieve a clear image.

3.2 ODROID-XU

The robot needed an on-board computer that could be used for inputting images and processing calculations. This computer was selected for a variety of reasons. A major benefit was the low power consumption of this device. The ODROID uses a processor that is based on the ARM architecture. This is different from regular PCs and is mainly used on limited resource devices like mobile phones. These were designed with the goal of reducing battery usage. They are also advantageous on the robot, as most of the available power from the 12 volt lead acid batteries needed to be used by the motors to move around the field.

Another advantage of the computer was the size and form factor. The robot was designed to be small and lightweight. A large computer would have required lots of design and planning for it to be integrated smoothly on-board. The small frame allowed for later work and to attach it after the completion of the robot itself. The processor and memory on the computer provided more speed

and storage in comparison to similar small computers, as well as additional processing cores that allowed the introduction of multi-threading into the program. The light weight allowed for the addition of more technology to the robot and makes it faster.

Technical specifications of the computer we used include an ARM Cortex-A15 processor with a speed of 1.6 GHz and four physical cores for multiprocessing. These provided more power and faster speed in comparison to several other small computers, such as the Raspberry Pi or the cRIO. It also provided 4 gigabytes of memory, which made development of the software quicker and increased speed.

3.3 Ubuntu Linux

All vision development was written on computers running Ubuntu GNU/Linux 13.10. This operating system is very appealing because it is free to acquire. A major benefit is its availability for many architectures including Intel x86 and Samsung ARM. It provides a collection of helpful software, including everything required for writing and editing code, managing revisions, compiling code, linking to helpful libraries like OpenCV, and accessing cameras. This allowed customizations that enabled debugging and the ability to run the vision program at the system level.

3.4 LEDs

The program used infrared light from rows of LEDs to find the targets. These were advantageous because it removes the need to focus on a particular color in the environment, and instead allows one to use gray-scaled images and preform a single channel threshold, which increases program speed. The targets were layered in retro-reflective tape, which redirects light directly back at the light source. With the cameras being in close proximity with rows of LEDs, the targets would appear very bright in the produced image. This tape is common and it is used on cars, stop signs,

bicycles and safety vests that are commonly used to increase visibility.

4. Formation of Problem

Through image analysis, the height of a contour can be obtained through applying a bounding rectangle to the entire contour. Then to find contour height, the center of the top 2 layers of pixels in the contour is found, as well as the center of the bottom two layers, and stored as Point2fs, which allows for floating point precision. If one were to simply call the height of the bounding rectangle, it would return an integer value, thus losing 8 significant figures in calculations.

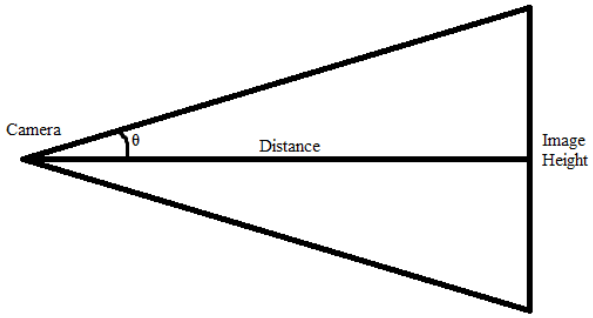
With knowledge of the height of the contour in pixels, as well as height of the target the contour represents in the image, one can calculate distance based off proportions of the image resolution the size of the target in the image

Eq 1.

$$\frac{Image\ Height\ (pixels)}{Contour\ Height\ (pixels)} = \frac{Viewable\ Target\ Plane\ Height\ (inches)}{Target\ Height\ (inches)}$$

The target plane height is defined as the part of the plane the target resides on that the camera is able to see with its field of view and resolution.

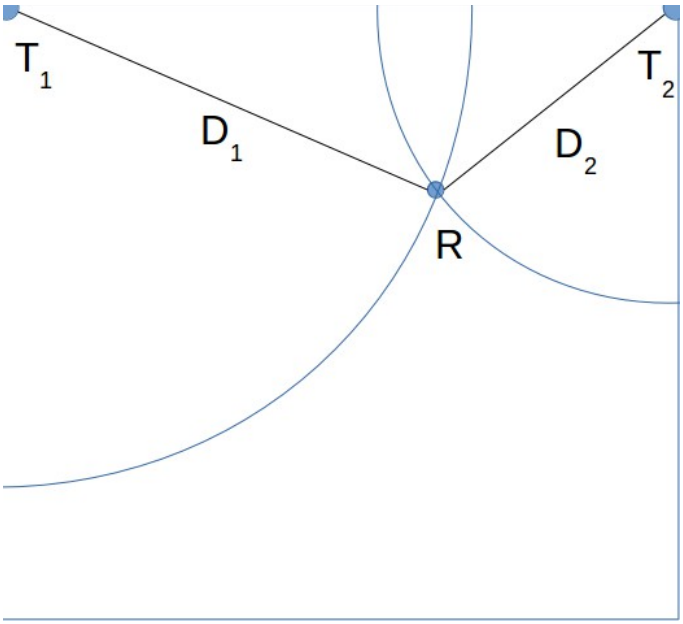
Now an isosceles triangle can be formed between the target plane and the camera's view of it.



Theta is defined as half of the field of view in the Y dimension visible by the camera.

Distance may easily be calculated through solving either the upper or lower triangle. This calculation is done for every target that is seen, but is adjust for target plane width when looking at verticle targets. If the program has identified at least 2 targets, it is able to triangulate where the center of the robot is which respect to all the targets by a GPS style calculation using intersections of circles.

Eq 2.



R is the robot, each D is the distance to each target T.

There can be a max system of 8 nonlinear equations. To ensure an anomaly in the inputted data doesn't influence the positioning calculation very much, each target is triangulated with every other target, and then the results are averaged. If a set of values are statistically significantly different than the average value, they are not included in the average calculation. With this, the robot is able to know with floating point accuracy, where it is on the field.

5. Procedure

The entire process of the program starts when the robot is powered up. In a cycle repeating 30 times per second, an image is retrieved from each of the cameras in three parallel processes, or threads. Each is responsible for retrieving and processing an image from each camera. This allows for the speed of the program to be more than tripled, as it does not have to wait for the retrieval and processing of the previous camera's image before beginning the transmission of the next. Each processor core of the computer is utilized for faster vision processing. There are multiple methods of querying a camera to retrieve image data. This program directly uses the

Linux V4L2 driver (Video for Linux 2). The program was therefore allowed to customize the internal parameters of the camera, allowing for a predefined focus and specific resolution output. The images were taken at 1280x720 and scaled down to 852x480 to make image processing significantly faster, while not losing too much data.

Fig 2. View of one Target Pair

Each image was then processed using multiple algorithms to focus directly on the targets. Initially, the top portion of the image was selected specifically to search for targets. The bottom portion would only include a view of the ground, game pieces, and other robots. A ROI (Region of Interest) was set on the top half because the images would only appear on the top half of the image because the cameras were lower than the targets, thus further increasing computational speed. In the perceived image by a camera that does not filter out infrared light, it appears in shades of gray, from pure black to white, where 0 is black and 255 is white. The brightest white indicated the reflection from the targets. The program then proceeded to perform a binary threshold on the images based on the intensity of the pixel. Only the highest levels, pixels that

had a value of 250 or greater, were accepted as potential targets. This step is known as thresholding the input image, and it results in a binary image, an image that only includes pure black, zero, and white, 255, to indicate the presence of a potential target or lack thereof.

Fig 3. Image after thresholding of IR light

Morphologic transformations are used next to remove anomalies in the image, called noise. Noise is the result of incoming infrared light from the sun in windows or from lights in the room's ceiling. The morphological process of eroding an image is taking the minimum of the pixel neighborhood (Eq. 3), then altering said neighborhood with respect to a structure element, while the process dilating the image takes the maximum of said pixel neighborhood (Eq 4) and altering the pixel neighborhood with respect to a structure element.

Eq. 3

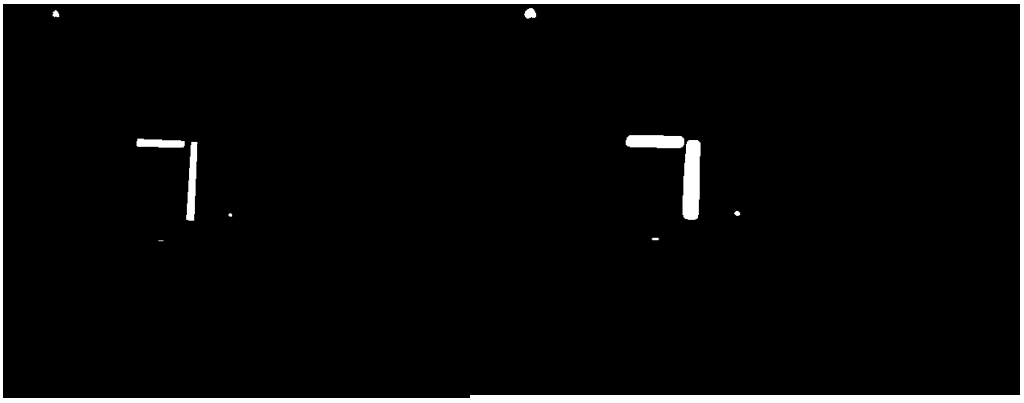
$$\texttt{dst}(x,y) = \min_{(x',y'):\,\texttt{element}(x',y')\neq 0} \texttt{src}(x+x',y+y')$$

Eq. 4

$$\texttt{dst}(x,y) = \max_{(x',y'):\,\texttt{element}(x',y')\neq 0} \texttt{src}(x+x',y+y')$$

Each performs the opposite of each other and is known together as the Open operation [3]. The erode process helps remove small regions of noise in the image, mostly appearing as small dots. However, this has the side effect of reducing the area of the targets in the image. The complementary operation dilate expands upon the remaining areas of the image and restores the visible area of the target back to the original size, which is important for accurately calculating distance. The idea behind this is to eliminate noise with the erode function, then restore your pixels of interest with the dilate function [3 p129].

Fig 4. Image after the process of erode/dilate



In order to further process the image data, the remaining visible regions are converted to contours (Fig 6). A contour (Eq. 5), mathematically, is defined as a directed curve which is made up of a finite sequence of directed smooth curves whose endpoints are matched to give a single direction.

Eq 6

$$\Gamma = \gamma_1 + \gamma_2 + \cdots + \gamma_n.$$

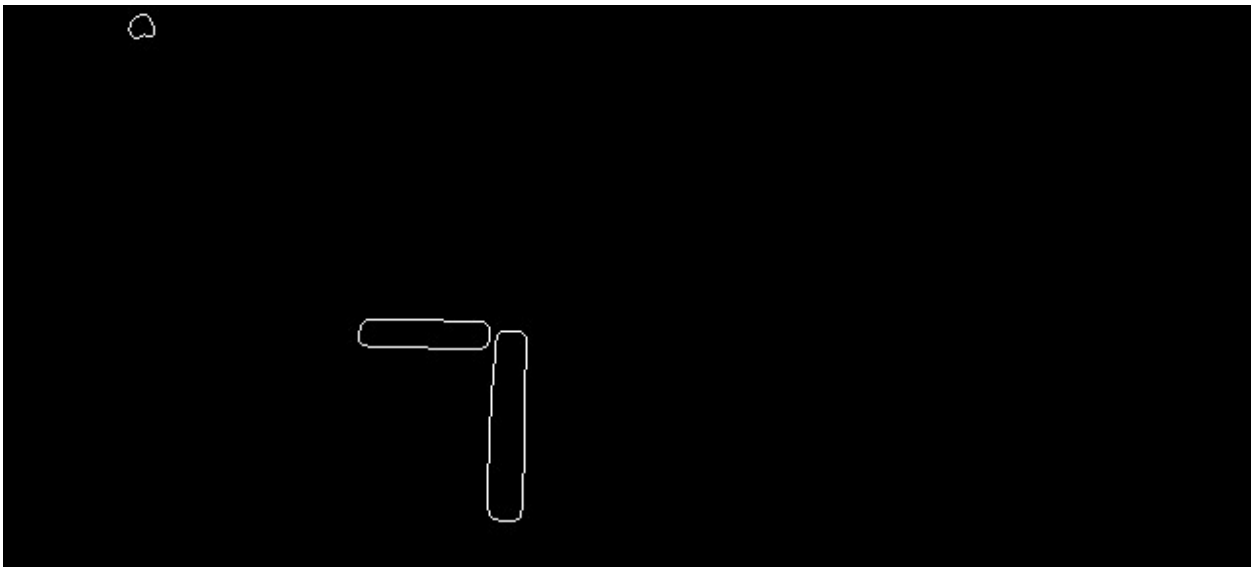
The contour integral of a function *f*: **R**→**R** can be given in terms of partitions of the contour in analogy with the partition of an interval. The integral over a contour is defined as the sum of the integrals over the directed smooth curves that make up the contour, which is given by Eq. 7

Eq. 7

$$\int_a^b u(t)dt + \int_a^b v(t)\,dt$$

Now that the potential targets are stored in a shape format, many functions can be used to further test for actual targets. The first is an approximation of polygons from the detected regions. This simplifies a piecewise linear curve with the Douglas–Peucker algorithm [6] by recursively dividing the line, automatically marking the two furthest points, then finds the point that is furthest from the line segment with the first and last points as end points. If the point is closer to the line segment than the maximum distance allowed to exist between the line and a point (ε), then any points not currently marked to be kept can be discarded without the simplified curve being worse than ε. A simple test can determine the resulting shapes (rectangle, triangle, circle, ..) from the amount of points, or sides left in the contour. In this program, this was used to convert the detected image contours into a list of straight lines. The ability to process the detected regions as rigid shapes instead of a series of curves allowed for more filtering techniques and tests, enabling the use of many basic geometric algorithms and features.

Fig 5. Detected contours in image



The program followed up the final contour detection with more tests based on the resulting data. A function was used to check if the returned polygons were convex, and if they weren't, they would be eliminated. To remove small noise, a couple of tests were also used. Green's theorem (Eq 8) is used to determine the area of a contour by integrating over the perimeter [11].

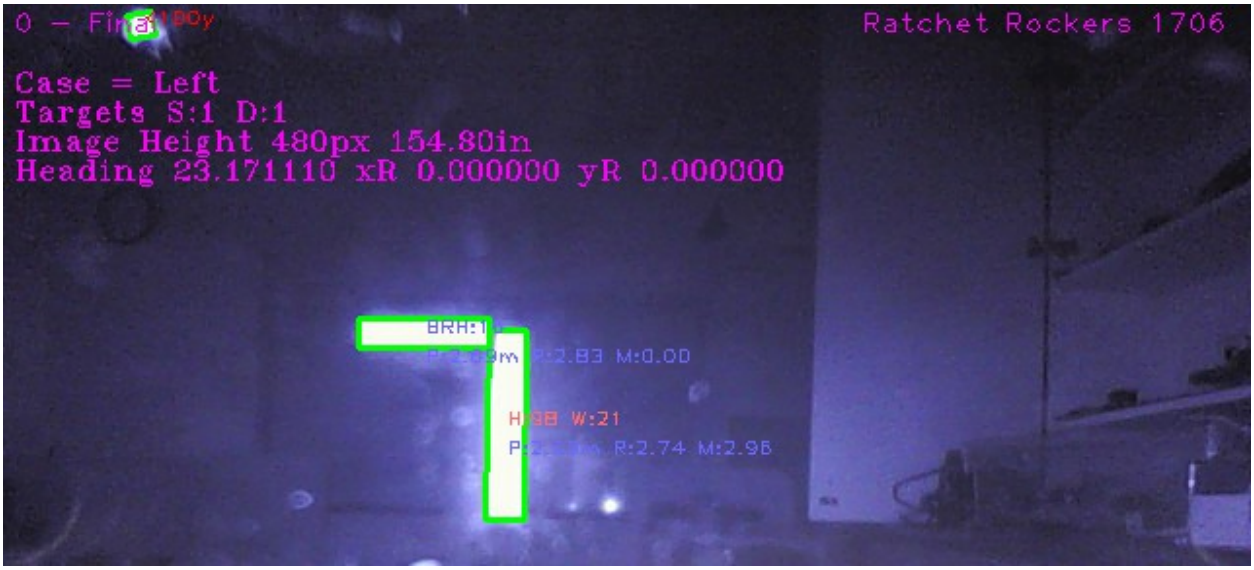
Eq 8

If it was less than what was ever expected to process, then that contour would be skipped, to save processing time and ensure the results are correct. A ratio test of the height of the contour in pixels and the width of the contours was also used. Each of the targets was either a long or tall contour. One dimension would be many times longer than the other. The function used to check if the detected shapes were a valid contour was to ensure these matched a certain range. If the dimensions were similar, the object was assumed to be square noise (Fig 6). If the dimensions were very different, the object was assumed to be unhelpful noise, such as visible light leaking in through cracks in the filter or a tall object from another robot.

The next step after ensuring the program is processing correct data is distance calculation and determine where on the field the robot was with respect to the target using the math outlined in

section 4.

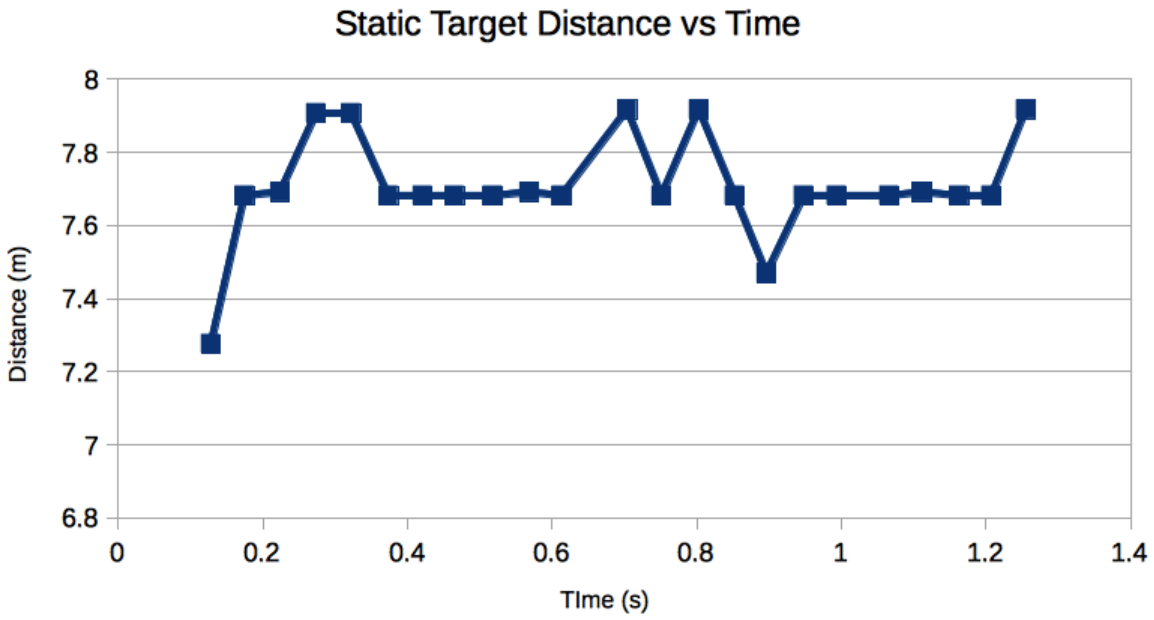
Fig 6. Final image displaying output of algorithms



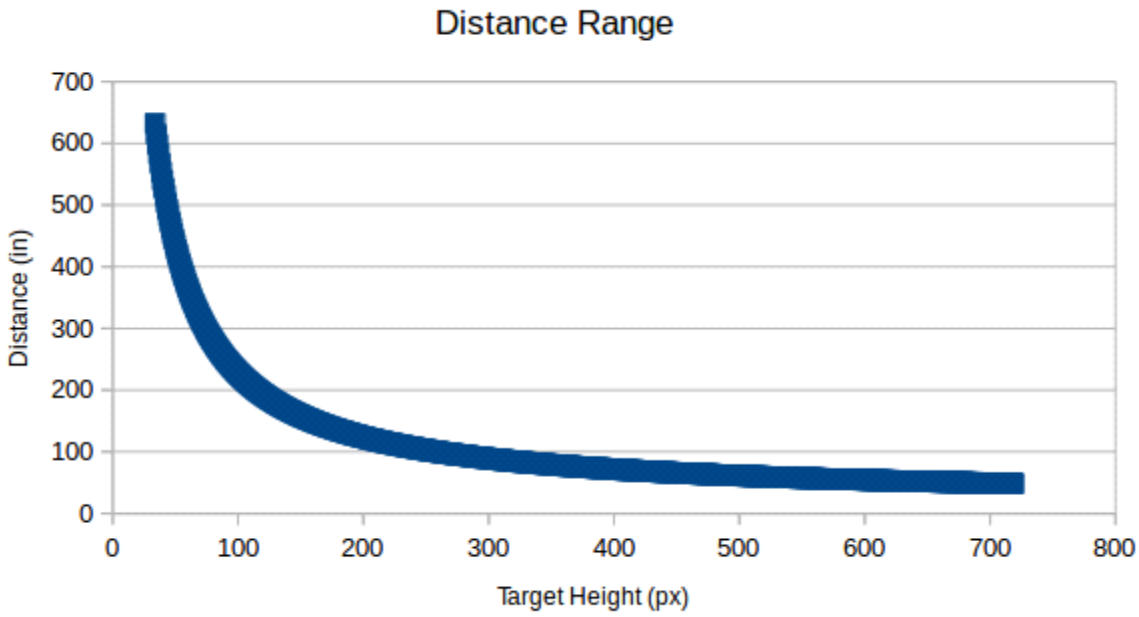
The previous figure (Fig 6) illustrates the final processed view of a single target visible from one of the robot's cameras. The distance is calculated as 2.74m. This is achieved through the mathematics overviewed in section 4. After the vision data is processed by each camera, the data is then filtered based on the possible situational cases. Theoretically, each camera could see a minimum of zero target pairs and a maximum of two pairs based on its position on the field and the direction. This procedure iterates through the cameras and assigns the pairs in the images to predefined indexes for each target pair on the field.

After all the distances and the specific targets are known, the position on the field can be found by solving a system of nonlinear functions, outlined in section 4. This knows the actual positions of the targets on the field and, after processing, returns a position relative to the top left corner of the field.

6. Results



This is a plot of time vs distance in a static environment. The mean of this data is 7.71 m with a standard deviation of 0.145 m. The inconsistency in this data set is largely irrelevant due to the very small deviation.



The distance returned by the algorithm was calculated to error by a maximum of 17.5 in, but the

maximum average error was found to be 0.43 in.

The results were shown to be precise as well as accurate. Any possible error could be reduced by using higher quality images from the robot cameras.

7. Discussion

Creating a working solution to a computer vision problem typically requires overcoming many various problems or shortcomings in a typical software development situation. These are commonly introduced as a result of some sort of change. New features or a different way of doing things may trigger unforeseen errors. Developing a viable local positioning system with multiple cameras for real-time use on a robot was no exception. With limitations on computational power, hardware and construction of the algorithm had to be carefully considered.

A large problem the vision solution had to work around this year was the issue of the target boxes combining in the image due to their close proximity to one another (Fig 7), the scale of our image, and the dilation algorithms. When the two targets combined, they would no longer show up as a pair of horizontal and vertical targets, as is expected. They appeared on-screen rather in a form of upside-down 'L' shape. This effect occurs when we are so far back from the target pair that the image quality makes it impossible to separate the two. It can also occur if we are too close and the IR lights' power is set too high, washing out the image.

Fig 7. Combined targets figure

To combat this problem, we used an innovative solution in order to deal with the combined targets. In addition to tracking the pairs of targets, the program is also able to detect a set of targets using the ratio of area in the contour to the area of the bounding rectangle. In order to find corners and other information about the targets, bounding rectangles are applied to the returned contours. This algorithm encloses all the pixels of a contour in a rectangle with the smallest possible area. A regular target is rectangular and reflects enough light back so the contour area is close to the area of the rectangle. If there is not a significant amount of light pixels in the rectangle, then the contour may be a combined pair of targets. This occurs because the combined target pair is not going to form a perfect rectangle. The ratio between the area of the contour and the area of the bounding rectangle in a combined pair stayed the same most times while testing. This allows us to filter our results for contours that more-or-less meet this ratio.

8. Real-Life Applications

There are many possible applications for the vision programming concepts used in this project. The simple target tracking could be applied to track many different objects as well, such as objects in the ocean. It would be able to track floating objects that appeared to be spheres or

rectangles with a distinct color. These objects could be picked up from various sources including satellite and aircraft. Since our software is made for real time tracking, the program would be able to immediately report back its information to a pilot or ship captain. It has support for processing previously captured images as well, meaning it can be used on satellite imagery.

As mentioned earlier, the tracking systems are able to accurately compute distance and position of an object based on its represented size in an image. This could be coupled with GPS technology to report back the size and absolute location of any objects it detects from only one view of it. The location could also be relative when reported back in real time.

Potential problems with this approach include detectability issues in certain situations. As these processes depend on the object of tracking being differentiated from its surroundings, they could be thrown off. If the object happened to be blue or white, it could accidentally report a false positive due to the condition of the water. The algorithms could be falsely triggered by a ship or a small island that happen to resemble the object. There would need to be more solutions put in play to discard unexpected readings from the program. This vision solution could use a series of tests, such as the ones used earlier this year to throw out abnormalities, or objected recognition software could be used to extract objects out of images.

We experimented with calculating the velocity of other objects on the field. This produced inaccurate results even when standing still because any slight change in the image produced a change in contour size, which gave the appearance of motion. A possible solution to this problem would be to average previous solutions within a given time frame together to find the average velocity and reduce noise in the image as much as possible.

9. References

[1] Smith, Aaron D., H. Jacky Chang, and Edward J. Blanchard. "An outdoor high-accuracy local positioning system for an autonomous robotic golf greens mower." *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012.

[2] Wolf, Jürgen, Wolfram Burgard, and Hans Burkhardt. "Robust vision-based localization by combining an image-retrieval system with Monte Carlo localization." *Robotics, IEEE Transactions on* 21.2 (2005): 208-216.

[3] Bradski, Gary, and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[4] Szeliski, Richard. *Computer vision: algorithms and applications*. Springer, 2010.

[5] Hansen, Per Christian, James G. Nagy, and Dianne P. O'leary. *Deblurring images: matrices, spectra, and filtering*. Vol. 3. Siam, 2006.

[6] Veregin, Howard. "Line Simplification, Geometric Distortion, And Positional Error." Cartographica 36.1 (1999): 25. Academic Search Elite. Web. 2 Feb. 2015.

[7] "Is LPS (3Mm), Not GPS (30+Mm), Driving New 3-D Grade Control Systems?." Roads & Bridges 36.11 (1998): 48. MasterFILE Premier. Web. 4 Feb. 2015.

[8] Zhi, Lihong, and Jianliang Tang. "A complete linear 4-point algorithm for camera pose determination." *AMSS, Academia Sinica* 21.239-249 (2002): 18.

[9] Hartley, Richard, and Andrew Zisserman. "Multiple view geometry in computer vision." *Robotica* 23.2 (2005): 271-271.

[10] McColl, Roddy I., and Andrew Johnson. "The Comparative Effectiveness Of Conventional And Digital Image Libraries."*Journal Of Audiovisual Media In Medicine* 24.1 (2001): 8-15.*Academic Search Elite*. Web. 9 Feb. 2015.

[11] Yang, Luren, and Fritz Albregtsen. "Fast and exact computation of Cartesian geometric moments using discrete Green's theorem." *Pattern Recognition* 29.7 (1996): 1061-1073.