Stereo Vision-Based Navigation for the ERTS Golf Cart

S. Sinha and B. Zaitlen

December 9, 2011

Abstract

The ERTS electric golf cart serves as a platform for experimental research in autonomous robotics. The system has both GPS capability as well as a laser scanning system to aid the cart in navigation. In this article we detail our efforts in adding a stereo vision system to the cart to provide visual object detection to guide navigation. We make extensive use of the Open Source Computer Vision (OpenCV) library of programming functions for real time computer based vision.

The article is divided into three sections. The first section defines the scope of our efforts and provides an overview of the subcomponents in our software architecture. The second section describes in detail a) the software we developed using the OpenCV software, and b) the method we chose to interface it with the CartFS system. The third section describes our simulated stereo vision based target distance estimation results. In addition, we document our successes and failures in field trials with the cart. The included appendices provide greater details on our algorithms used for: inserting additional waypoints, obstacle avoidance, vision based object detection, and camera calibration.

We feel the modularity of our design lends itself to easy expansion. We hope this facilitates future endeavors to expand the Cart's functionality with stereo vision based navigation.

Contents

1 Introduction **2**
1.1 Objectives 2
1.2 OpenCV Components for Stereo Vision 2

2 Implementation **3**
2.1 Stereo Camera Prototyping Hardware 3
2.2 Camera Calibration 4
2.3 Stereo Calibration and Rectification 5
2.4 Color Based Object Detection 7
2.5 Disparity and Distance Estimation of Pre-Identified Objects . . . 8
2.6 Integration with CartFS 10
2.7 Conclusions 11

A Vision Code Documentation **12**
A.1 Object Detection Documentation 12
A.2 Disparity Map Documentation 12

B Algebraic Calculations for Adding Additional Course Waypoints and Obstacle Avoidance **12**

C Visualization of Telemetry Data using *matplotlib* and *Matlab* **12**

List of Figures

1 Photograph of a single USB camera in our stereo vision testbed system. 3
2 Photograph of the machined stereo camera rig. 4
3 Screenshot of corner detection using a calibration pattern. 5
4 Example of a Calibrated and Rectified Stereo Image Pair with a Disparity Map. 6
5 Screenshot of object detection based on pixel HSV. 8
6 Creating a bounding box from stereo frames and cropping it out. 9
7 The technique for gauging depth information given two offset images is called triangulation. 10
8 Schematic of the method used to add additional waypoints. . . . 13

1 Introduction

1.1 Objectives

Our objective in this project was to implement a stereo vision-based navigation system for the ERTS cart. The cart itself has GPS and laser-range finding capabilities for localizing itself in space and detecting foreign objects in its environment. Complimenting these systems with vision based object detection and localization capabilities seemed both interesting and educational to us. With these motivations, and the potential future benefit to the course, we made steps toward a stereo vision-based pre-defined object detection and distance mapping program.

Vision based guidance is a *substantial* undertaking that lead us to a number of specific design constraints. Constraints were imposed along two broad fronts and were necessary at a minimum to contend with the limited project development time frame. First, we wanted the vision system to operate independently from the other cart processes. This allowed us greater flexibility in developing the intended software. Interface with the cart was accomplished by calling the vision program within the current architecture, and then having the vision program write appropriate dictionary entries for CartFS.

Second, object detection and distance estimation are both complex computer vision problems. As a result we radically reduced the problem by a) predefining an object to identify in each video frame, and then only estimating that object's distance in video streams.

1.2 OpenCV Components for Stereo Vision

To tackle the vision processing goals and attempt to meet the hard time limits imposed by autonomous navigation, we leveraged the well developed OpenCV library of functions. OpenCV is an open source computer vision library written for computational efficiency. We used this library of functions for every part of our stereo-vision based distance estimation program. OpenCV is structured in to five main components. We made extensive use of functions from the HighGUI component and the CV component. The former provides I/O routines and functions for storing and loading videos and images. The latter provides image processing and higher level computer vision algorithms. Structural analysis and shape descriptors, and camera calibration and 3D reconstruction routines were of key importance.

The details of our design are outlined in the following section on Implementation. In essence we divided our overall vision task into the following subtasks. The first required us to acquire cameras and machine a stable camera mounting rig. The subsequent subtasks made use of specific components from OpenCV.

1. Stereo Camera Calibration
2. Hue-based Object Detection

3. Disparity and Distance Estimation of Pre-Defined Objects

4. Navigation Based on Object Distance

2 Implementation

2.1 Stereo Camera Prototyping Hardware

In our prototype we made use of two identical USB cameras. An ideal stereo camera system would use synchronous frame grabbing cameras. We however made use of two cameras running independently. The cameras had 24 bit RGB capability and collected 15 frames per second. The maximum resolution possible was 640 x 480 pixels and they automatically performed contrast balancing. The lens of each camera was manually adjustable, and we generally set the focal length for prototyping to approximately 2m. For actual use on the cart we expected to focus on objects at least 4-5 meters away.

To mount the cameras we machined an aluminum platform. It consisted of two 14 x 2 x 1/8 thick aluminum plates with grooves cut out to accept the camera bases. The grooves were machined so that the center of the camera lenses were 12 inches apart. The cameras were held rigidly in place by modestly separating the two plates with threaded rods (see Figure 1 and 2). The two metal plates were secured to a 1/2 inch thick aluminum base plate. This base plate could be mounted to a tripod using standard 1/4 x 20 threaded hole (see Figure 2).

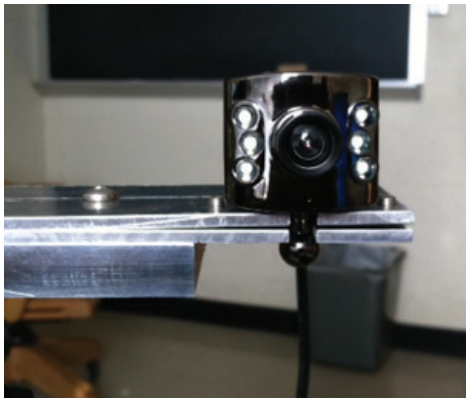


Figure 1: Photograph of one of the USB cameras in our stereo vision testbed system. The camera has a variable focal length (3cm - ∞), 24-bit RGB resolution, and a maximum spatial resolution of 640 x 480 pixels. We focused the lenses to 10m. The infrared LEDs on either side of the lens were not used.

Figure 2: Our two USB cameras mounted on our custom built platform. Our camera platform can be screwed into a tripod stand using a threaded 1/4-20 hole. The camera lens centers are spaced 12" apart and secured rigidly in place.

2.2 Camera Calibration

Camera calibration is important for relating camera measurements with measurements in the real world. The details for calibration are rather involved and we do not describe them in exhaustive detail. Instead we only briefly describe the overall steps in calibration, and routines from OpenCV that we use for this project. For details on the mathematics and more on practical limitations refer to Learning OpenCV by O'Reilly.

OpenCV takes as a starting point for calibration a simple pin hole camera model. A lens is always used in cameras for practical reasons. In theory, it is possible to define a lens that will introduce no distortions, however, in practice no lens is perfect. This is mainly for reasons of manufacturing; it is much easier to make a spherical lens than to make a more ideal parabolic lens. It is also difficult to mechanically align the lens and a camera's imaging plane exactly. As a result two main distortions generally arise. Radial distortions arise as a result of the shape of lens, whereas tangential distortions arise from the assembly process of the camera as a whole. Radial distortions are generally modeled using three coefficients and tangential distortions with two coefficients.

The relation that maps the point P_i in the physical world with coordinates (X_i, Y_i, Z_i) to the points on the imaging plane with coordinates (x_i, y_i) is called a projective transform. This transform in the case of a camera is described with the intrinsic matrix which has components that characterize the focal length of a lens along its x-, and y- axes (f_x and f_y) and correction factors that account for the imaging chip being offset from the optical axis of the lens (c_x and c_y).

OpenCV has routines that allow these intrinsic and distortion properties of a camera to be computed. It provides several routines to help us compute these intrinsic parameters. The actual calibration is done via cvCalibrateCamera2. In this routine, the method of calibration is to target the camera on a known

structure that has many individual and identifiable points. By viewing this structure from a variety of angles, it is possible to then compute the (relative) location and orientation of the camera at the time of each image as well as the intrinsic parameters of the camera.

In principle, any appropriately characterized object could be used as a calibration object, yet the practical choice, and utilized in OpenCV, is a regular pattern such as a chessboard (Figure 3). While a three-dimensional object can be used, flat chessboard patterns are much easier to deal with. OpenCV thus requires using multiple views of a planar object (a chessboard) rather than one view of a specially constructed 3D object. To find the corners of a chessboard pattern and plot on an image the identified corners we used cvFindChessboardCorners and cvDrawChessboardCorners.

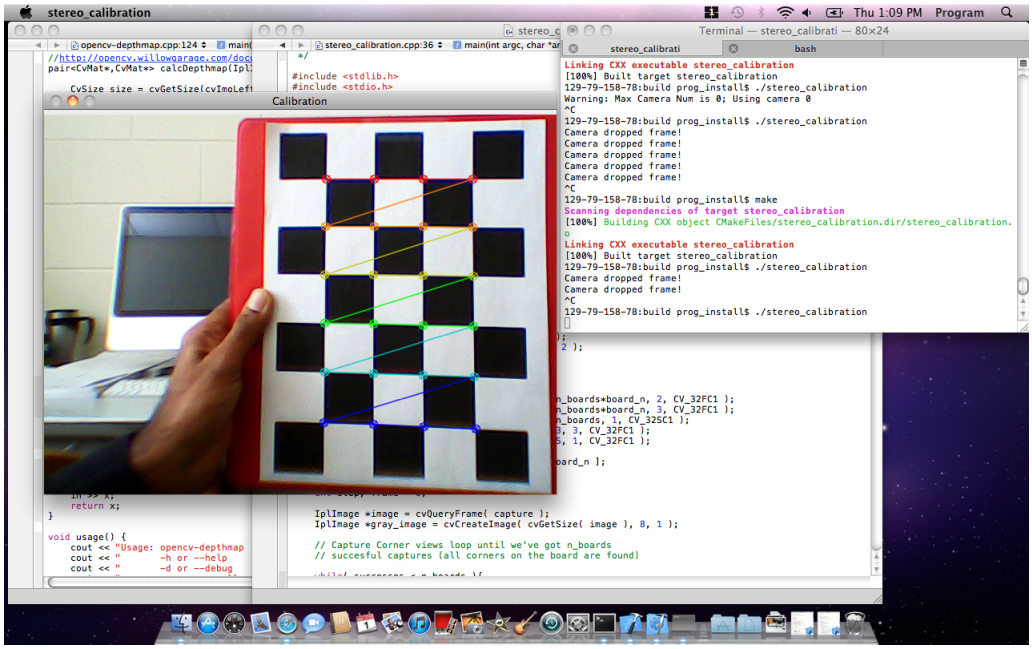


Figure 3: Single camera calibration using a two dimensional chess board pattern. The corners of the pattern were identified and the a set of colored points and lines were added to confirm for the user the points the routine has identified.

2.3 Stereo Calibration and Rectification

Now we describe steps involved in preparing stereo frames for subsequent processing. Computer vision systems accomplish stereo imaging by finding correspondences between points that are seen by one imager and the same points as seen by a second imager. With such correspondences and a known baseline separation between cameras, we can compute the 3D location of the points. Al-

though finding corresponding points can be computationally expensive, we can use our knowledge of the geometry of the system to narrow down the search space as much as possible. We followed a four step calibration procedure for our stereo imaging.

1. Perform calibration of two cameras estimate intrinsic parameters, distortion effects, and the relative position between the 2 cameras using cvStereoCalibrate.

2. We removed radial and tangential lens distortion, a process called *undistortion* using cvUndistortPoints. The outputs of this step are undistorted images. These procedures have been described above.

3. Adjust for the angles and distances between camera images, a process referred to as *rectification* where we used cvStereoRectifyUncalibrated. The outputs of this step are images that are row-aligned and rectified.

4. Compute a map of the joint undistortion+rectification transformation to use on all subsequently acquired image frames using cvInitUndistortRectifyMap.

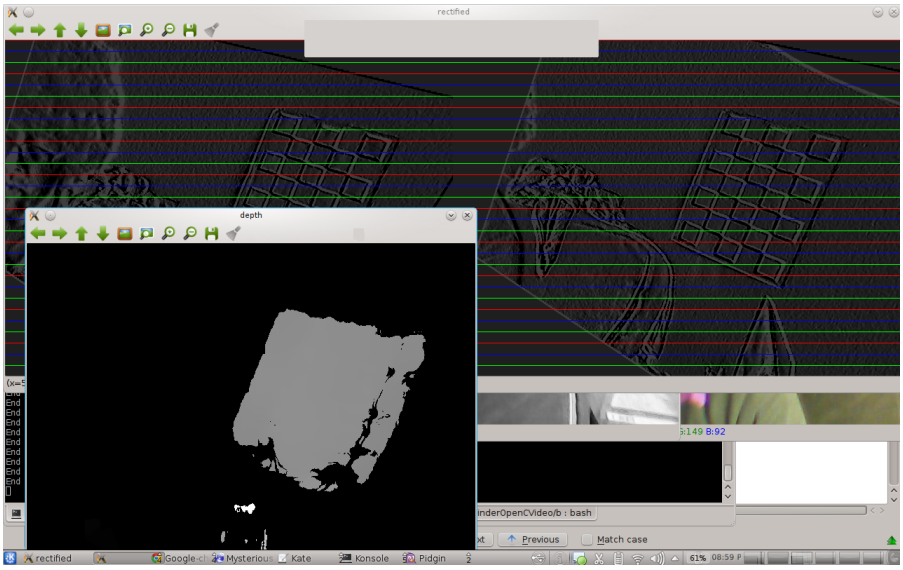


Figure 4: The top two panels show the left and right calibrated and rectified stereo images. The chessboard calibration pattern can be seen in the right halves of the top windows. Clearly adjustments need to be made as the images are rotated, whereas the actual calibration pattern was parallel to the floor. The bottom window shows the corresponding disparity map. More work is needed in order to have other objects in the original scene represented in the disparity map.

2.4 Color Based Object Detection

We decided from the start to search solely for pre-defined objects. Early on we chose to use traffic cones as they have a characteristic shape and characteristically non-natural color, making their identification in a natural environment possible. Based on this type of object we developed routines to parse video frames by a) identifying pixels with an appropriate hue-saturation-value (HSV), b) then thresholding all pixel values to find pixels with a strong signal, c) then connecting regions with strong HSV signatures, and finally d) cropping out these regions. We realized that HSV values can vary substantially with natural lighting conditions and we made tentative efforts to characterize the variation of the pre-defined HSV under a range of conditions. The routine performed the following sequence of steps:

1. Threshold image using cvInRangeS specifying the image and HSV Minimum and Maximum values
2. Use cvBlobsLibs to run a connected-components algorithm on the image and exclude blobs with less than N components
3. Region of Interest (ROI) Determination based on bound bounding box of blob
4. Crop ROI

Figure 5 shows a screenshot that demonstrates part of this routines performance. The reddish three ring binder on the desk is successfully identified with this routine. The top frames are from left and right cameras. The yellowish dots are from consecutive previous frames and identify the centroid of a region with HSV values meeting the preselected hue criterion [1]. The bottom windows show the corresponding frames but thresholded to more clearly show the selected pixels. It should be noted that the routine's ability to select pixels corresponding to the binder color was compromised if the lighting conditions in the room changed significantly.

Figure 6 goes further and finds an object in a cluttered field and crops it out. The bottom left figure has a pink dot on a green object. In addition the scene is complicated with other objects, variations in lighting, and colors. In the bottom right screen the object is selected and a black bounding box is placed around the pixels with the relevant HSV. The thresholded version of this image is shown in the top right window. Finally the region of interest is cropped and shown in the top left window. This method of specifying a HSV range is sensitive to lighting conditions.

[1] $CvScalar hsv_min = cvScalar(0, 50, 170, 0);$
$CvScalar hsv_max = cvScalar(10, 180, 255, 0);$
$CvScalar hsv_min2 = cvScalar(170, 50, 170, 0);$
$CvScalar hsv_max2 = cvScalar(255, 180, 255, 0);$
$cvInRangeS(imgHSV, hsv_min, hsv_max, imgThreshed);$
$cvInRangeS(imgHSV, hsv_min2, hsv_max2, imgThreshed2);$
$cvOr(imgThreshed, imgThreshed2, imgThreshed);$

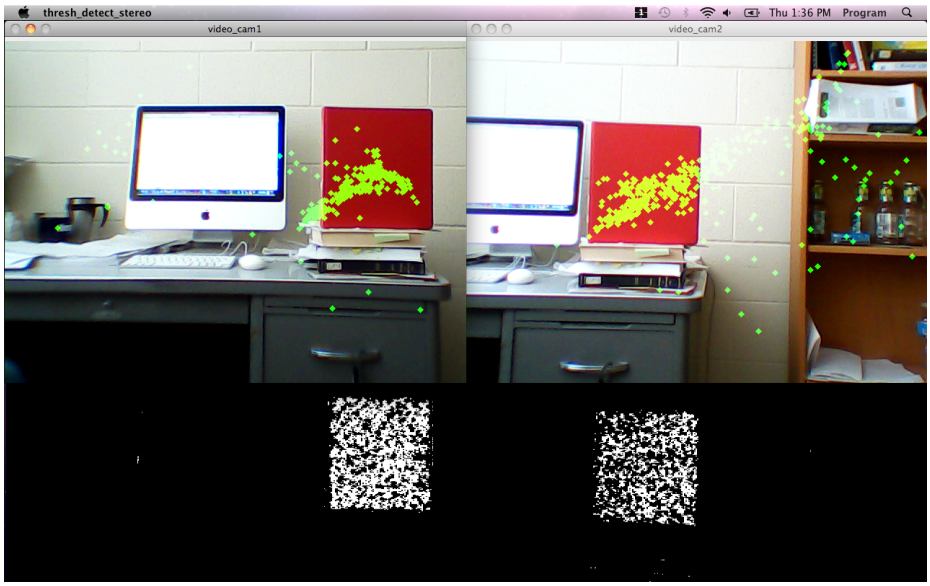


Figure 5: Object detection using a color based scheme for uncalibrated frames. Top: A single frame from each camera. Yellow dots represent regions where the program detects pre-selected HSV values. Bottom: HSV-based thresholding of camera frame images. The white dots correspond to points cumulatively acquired that have the appropriately HSV specification. Even in cluttered scenes objects can be identified if their characteristic statistics deviate from the surroundings.

2.5 Disparity and Distance Estimation of Pre-Identified Objects

Once an object has been identified in each camera frame our plan was to use a straightforward method from OpenCV that simply attempts to match windows around each pixel from one frame to the next known as block matching. The term "block matching" is a catchall for a whole class of similar algorithms in which the image is divided into small regions called blocks. Blocks are typically square and contain some number of pixels (e.g. 10 x 10). The selection of blocks can overlap and, in practice, algorithms generally do this. Block-matching algorithms attempt to divide both the previous and current images into such blocks and then compute the shift of these blocks. This shift in blocks from one frame to the next, or in our case, across two essentially simultaneously sampled frames is referred to as disparity. This method generally holds because frames can be captured more quickly than the real world images themselves change. While this will aid in generating a disparity map this class of algorithms play an important role in optical flow for computer vision.

To better explain this we include Figure 7. For any point P of some object

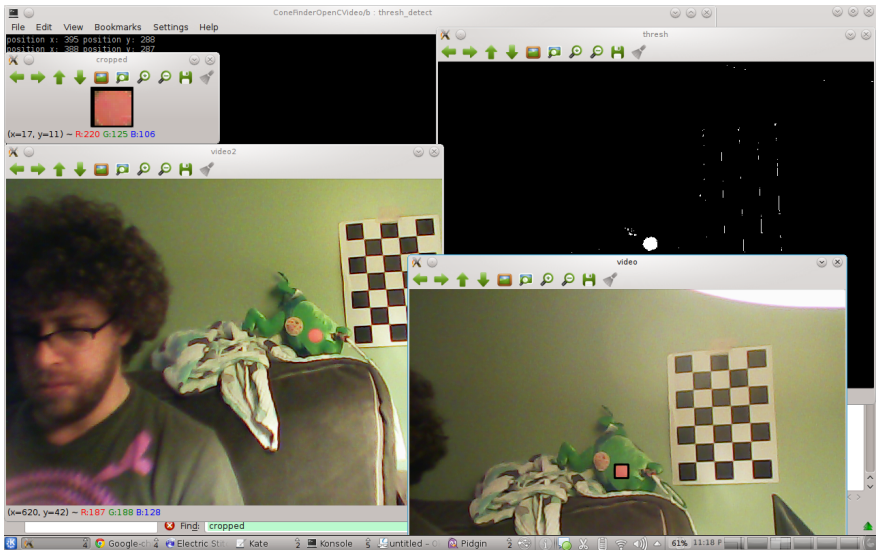


Figure 6: A bounding box can be put around a predefined target and it can be cropped out of an a video frame. Top Left: A pink object has been bounded and cropped out of a larger visually complex field. Top Right: Objects with a predefined hue are detected and the image is thresholded. The pink button and the edges of a calibration pattern are marked by the program. Bottom: Left and right image frames from a pair of cameras. The pink button in the center of a green toy is detected based on its hue.

in the real world, P1 and P2 are pixel point representations of P in the images IP1 and IP2 as taken by cameras c1 and c2. f is the focal length of both camera, and is the distance between lens and imager. b is the offset distance between cameras c1 and c2. v1 and v2 are the horizontal placement of the pixel points with respect to the center of the camera. The disparity of the points P1 and P2 from image to image can be calculated by taking the difference of v1 and v2. This is the equivalent of the horizontal shift of point P1 to P2 in the image planes. Using this disparity one can calculate the actual distance of the point in the real world from the images Eq. 1.

$$D = \frac{(b \times f)}{d} \tag{1}$$

where D is the distance between the real object and the imaging planes, b is the distance between the centers of imaging plane, f is the common focal length, and d is the calculated disparity.

With regards to OpenCV we have attempted the following two step procedure.

1. Find the same features in the left and right camera views, a process known

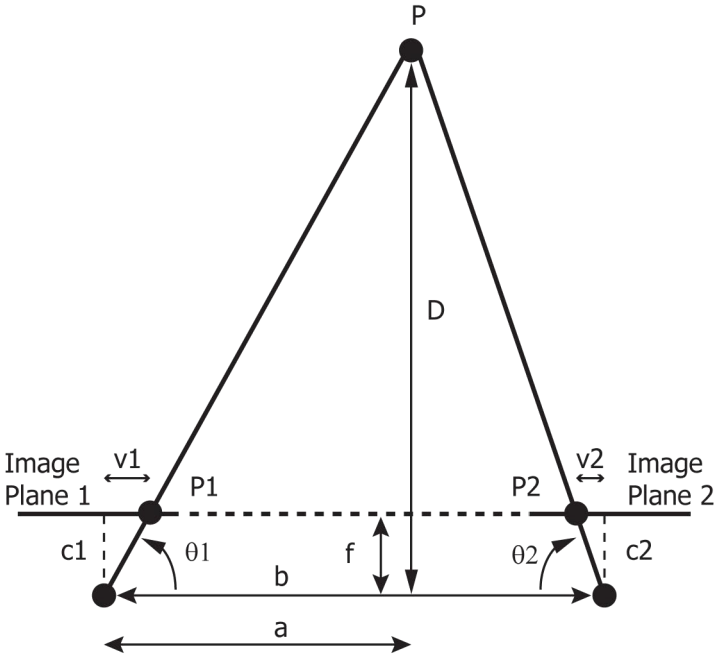


Figure 7: Triangulation makes use of a number of variables; the center point of the cameras (c1, c2), the cameras focal lengths (f), the angles ($\theta 1$, $\theta 2$), the image planes (IP1, IP2), and the image points (P1, P2).

as *correspondence*. The output of this step is a disparity map. We used OpenCV's cvFindStereoCorrespondenceBM.

2. Given the geometric arrangement of the cameras, we can turn the disparity map into distances by triangulation. This step is called *reprojection*, and the output is a depth map for which we can use 1.

2.6 Integration with CartFS

We developed a small class which uses the JsonCPP library to handle interactions with the JSON dictionaries on CartFS. While, the class needs to be fleshed out with proper read and write functions, we did develop a sketch for the dictionary which will hold three key/value pairs.

1. Enable : {true, false}
2. ObjectWarning : {low, medium, high}
3. Location : {left, right, both}

The stereo vision depth finder program would pass out information about the approximate distance of objects (ObjectWarning key) and approximate location information (Location key).

2.7 Conclusions

Our objective for this project was to develop a prototype stereo vision depth finder. If successful our intention was to then plan in greater detail a system for integration with the ERTS cart.

Currently we have built a simple stereo camera platform, and successfully implemented single camera calibration and object detection based on HSV using OpenCV routines. We have a partially working stereo calibration program which needs reworking to correct obvious errors in rotation and distortion of camera images. To date we have been unsuccessful in creating accurate disparity maps. Our maps currently have very little spatial resolution, and the represented gray scale range varies little across any given. As a result we did not proceed with a reprojection step, and are currently debugging our disparity map routines in the hope we can be more successful in the short term.

In the future we would recommend the development of more accurate disparity maps and corresponding depth maps. In addition development of routines for object detection based on shape and based on color statistics would be of interest, and which hope to pursue for our own research.

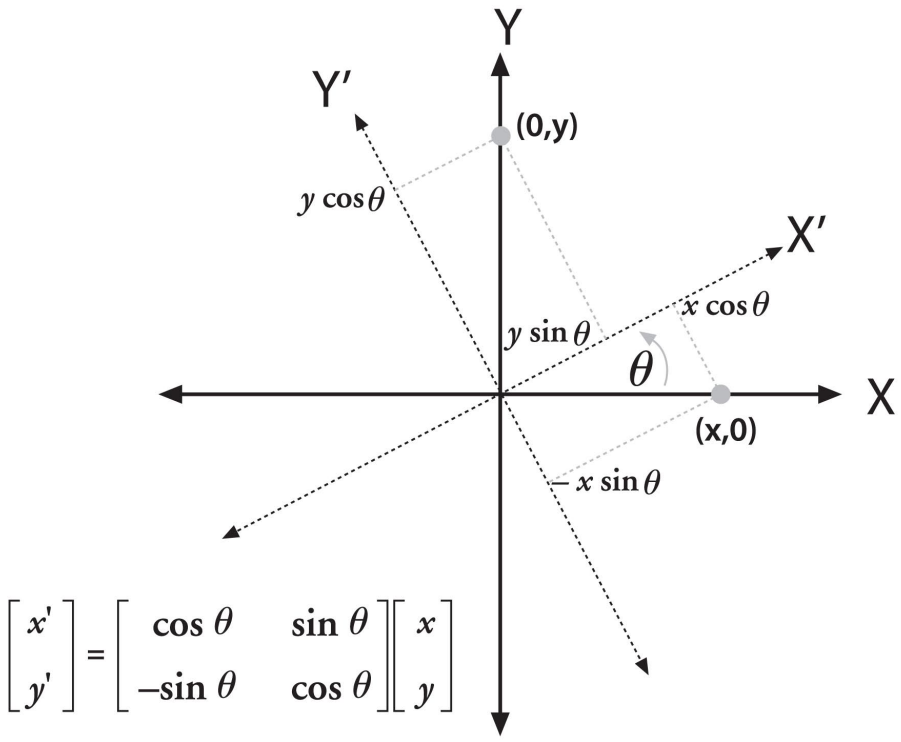
A Vision Code Documentation

A.1 Object Detection Documentation

A.2 Disparity Map Documentation

B Algebraic Calculations for Adding Additional Course Waypoints and Obstacle Avoidance

C Visualization of Telemetry Data using *matplotlib* and *Matlab*



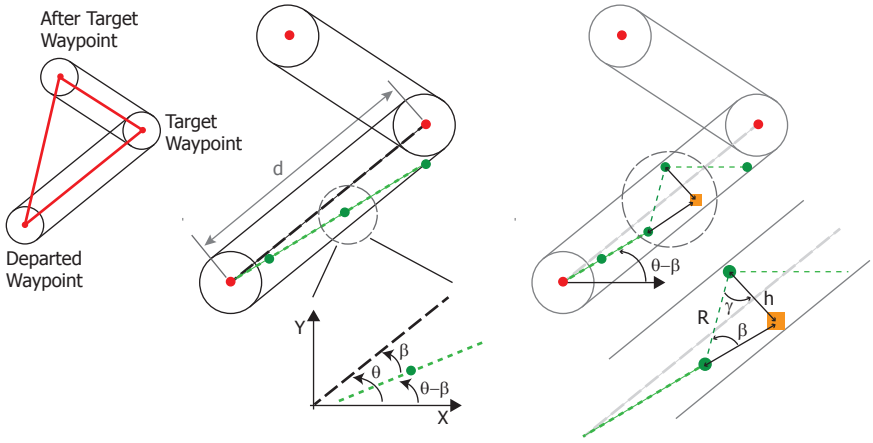


Figure 8: Object detection using a color based scheme. Top: Frame from each camera. Yellow dots represent regions where program detects appropriate color. Bottom: Thresholded camera frame images. The white dots correspond to points cumulatively acquired that have the appropriately color based on the detection algorithms color specification.