Detecting, Tracking, and Warning of Traffic Threats to Police Stopped Along the Roadside

by

James Karraker

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

© Massachusetts Institute of Technology 2013. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2013

Certified by .....
Seth Teller
Professor
Thesis Supervisor

Accepted by .....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Detecting, Tracking, and Warning of Traffic Threats to Police Stopped Along the Roadside

by

James Karraker

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2013, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Despite years of research into improving the safety of police roadside stops, reckless drivers continue to injure or kill police personnel stopped on the roadside at an alarming rate. We have proposed to reduce this problem through a "divert and alert" approach, projecting lasers onto the road surface as virtual flares to divert incoming vehicles, and alerting officers of dangerous incoming vehicles early enough to take life-saving evasive action. This thesis describes the initial development of the Officer Alerting Mechanism (OAM), which uses cameras to detect and track incoming vehicles, and calculates their real-world positions and trajectories. It presents a procedure for calibrating the camera software system with the laser, as well as a system that allows an officer to draw an arbitrary laser pattern on the screen that is then projected onto the road. Trajectories violating the "no-go" zone of the projected laser pattern are detected and the officer is accordingly alerted of a potentially dangerous vehicle.

Thesis Supervisor: Seth Teller
Title: Professor

Acknowledgments

This work is partially funded by the National Institute of Justice. Any opinions, findings, and conclusions or recommendations expressed in this thesis are those of the author and do not necessarily reflect the views of the National Institute of Justice.

Professor Seth Teller, for giving me this wonderful opportunity to work on a great project with life-saving potential.

Professor Berthold K. P. Horn, for his invaluable help with the machine vision algorithms and all the math behind them.

Brian Wu and Pallavi Powale for being great teammates on the project, and putting up with me in the lab.

Contents

List of Figures

List of Tables

Chapter 1

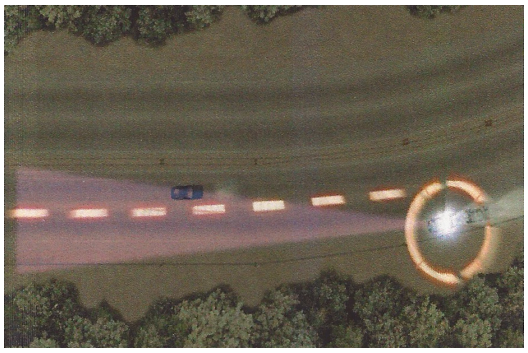
Introduction

Police officers and their vehicles are frequently struck by incoming drivers while stopped along the roadside. Despite regular use of vehicle mounted emergency lighting (VMEL) systems to alert incoming drivers, there are a regrettably high number of injuries and deaths caused by impaired drivers. Other drivers are apparently not impaired, yet still do not notice or react to the police officer on the side of the road. This leads us to believe that existing VMEL systems fail to generate sufficiently effective cues to divert incoming vehicles. Some drivers, however, are indeed impaired, in which case no type or amount of emergency lighting may divert that driver from colliding with the officer. In this case an easy-to-deploy mechanism is needed for alerting the officer of an incoming potentially dangerous vehicle, early enough so that the officer has enough time to get out of harm's way.

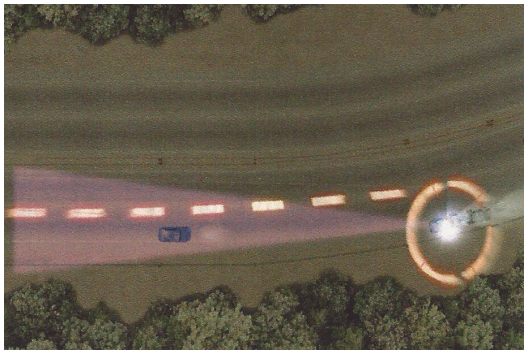
To solve these two main problems and reduce the number of these preventable accidents, we use a two component approach: divert and alert. Our Motorist Diverting Mechanism (MDM) projects virtual flares onto the road using lasers, giving incoming drivers visible cues to change course and avoid the officer and parked police vehicle. Our Officer Alerting Mechanism (OAM) detects incoming vehicles and determines if they are potentially dangerous, in which case it alerts the officer of the danger so that he or she can take evasive action. Figure 1-1 shows an example of the environment in which the system would be used.

The Divert and Alert project is a joint project between the Robotics, Vision, and

Sensor Networks group (RVSN), BAE, and the Massachusetts State Police. This thesis will focus mainly on the OAM, and the detection, tracking, and machine vision algorithms that have gone into it. Section 2 reviews relevant background material. Section 3 states the problem. Section 4 provides an overview of the technical achievements I have made and detail the steps I have taken in my work on the OAM. Section 5 evaluates some results of my work, analyze my contributions to the project, and present opportunities for future work.



(a) A police cruiser parked on the roadside looks back at the road and projects a line of laser virtual flares.



(b) The system detects when an incoming vehicle breaks the flare line and presents a danger to the officer, then alerts the officer early enough to take life-saving evasive action.

Figure 1-1: An example of the Divert and Alert system usage case. The dotted lines are laser virtual flares, and the pink cone represents the camera's field of view.

Chapter 2

Background

2.1 Review of Relevant Literature

In this section I will perform a brief review of literature relating to roadside accidents, lane estimation, and vehicle tracking.

2.1.1 Incidence of Roadside Accidents

Roadside accidents involving police officers are relatively well documented. I will highlight some interesting statistics that have come out of the documentation on these types of accidents. The most recent findings show that over the decade from 2000-2009, there were 47 law enforcement officer deaths from being struck by a vehicle while conducting a "Traffic stop, road block, etc." in the United States. Another 73 deaths occurred while the officer was "Directing traffic, assisting motorist, etc"[21].

Solomon and Ellis (1999) found that most roadside incidents with police vehicles occur on straight, dry roads in clear weather[17]. Charles et al. (1990) found that the normative condition for roadside accidence involved unimpaired drivers, and that the majority of accidents occurred on unmarked roads[3]. They also found that the number of roadside accidents with police vehicles was proportionally similar to the number of accidents with passenger vehicles. In addition they found that a common cause of the accidents is sleep deprivation. Agent and Pigman (1990) also found that

sleepiness, as well as alcohol impairment, were major causes of roadside accidents[1]. These results seem to suggest that there are many accidents that no diverting system can prevent because the driver is so insensate due to impairment or sleepiness. In these cases a warning system would be the only effective way to avoid injury to the officer on the side of the road.

2.1.2 Lane Estimation

Our own group, the Robotics, Vision, and Sensor Networks Group, has developed sensor-based methods for identifying and locating nearby traffic lanes on real roads under challenging lighting and weather conditions[13, 14]. Lane estimation was formatted as a curve-fitting problem, where local sensors provided partial, noisy observations of lane geometry. The system can handle roads with complex geometries without making assumptions about the position or orientation of the vehicle.

Mobileye is a commercially available system that uses monocular vision algorithms to perform lane detection, among other applications. The Mobileye system sits on the dashboard of a moving car, and uses its algorithms to help the driver drive more safely, by giving him or her lane departure warnings, forward collision warnings, and similar features[16]. To our knowledge no lane-finding method specific to roadside vehicles has been developed.

2.1.3 Vehicle Tracking and Classification

An enormous body of work exists on video-based object detection, tracking, and classification. Stauffer and Grimson (1999) created a widely used mixture-of-Gaussians background modeling algorithm that detects moving objects, which works very well for stationary cameras, but cannot handle slight perturbations caused by wind or small vehicle movements[19].

There are also many existing object tracking systems, which either associate individual independent detections over time[5], or search for an object at later times based on knowledge of that object at earlier times[4, 22].

Work in the field of visual surveillance has led to a number of techniques for building normalcy models based on continuous observation[18, 23, 2]. Classification algorithms can use these normalcy models to predict future behavior, such as speed and direction, as well as identify anomalous objects that do not fit the normal behavior, such as a swerving car.

2.2 LCM

We use the Lightweight Communication and Marshaling (LCM) protocol to communicate between separate parts of the system. LCM is a system of libraries for transferring data over a local-area network, which was designed by the MIT DARPA Urban Challenge Team[15]. LCM is made for real-time systems such as ours that require high bandwidth and low-latency data transfer. We have 142 GB of LCM logs (about an hour) of roadside video that we use to test our system. These logs simulate the communication between the cameras and the software system using LCM. We also use LCM to communicate between the laser and the software system.

Chapter 3

Problem Statement

Impaired, inattentive, or sleeping drivers strike police officers and other emergency workers, as well as their vehicles, at an alarmingly high rate, causing damage, injuries, and often fatalities. The traditional way to combat these types of accidents is to alert drivers to the presence of the vehicle on the side of the road in some way. Vehicle-mounted emergency lighting (VMEL) is routinely used to divert drivers from the officers and their vehicles on the roadside. Yet there is still an unnecessarily high rate of accidents, many of them involving apparently unimpaired drivers (Charles et al., 1990)[3]. We therefore conclude that existing VMEL systems fail to generate sufficiently effective cues to divert drivers from collisions with the vehicles on the side of the road.

Some drivers cannot be diverted by any emergency lighting system. In this case, something else must be done to reduce the danger of roadside collisions. We have developed the early stages of an easy-to-deploy mechanism to warn the officer of a potentially dangerous oncoming vehicle, early enough to allow the officer to get out of the way in time to avoid injury.

Our system addresses the problem of roadside collisions from two separate angles, based on the needs we perceive:

1. Create a more effective vehicle-mounted emergency lighting system to divert oncoming drivers.

2. Create a warning system that robustly perceives potentially hazardous oncoming vehicles, and effectively alerts officers in time to avoid disaster.

By combining these two approaches, our system contains two layers of risk mitigation. The first layer will divert many possibly dangerous oncoming drivers, and the second layer will warn the officer of any vehicles that break through the first safety layer. By integrating these two approaches into a single system we will give police officers and other emergency workers an effective safety net from roadside collisions. I worked on the second approach, the alerting approach, and this thesis will cover that portion of the project.

3.1 Officer Alerting Mechanism (OAM)

The development of the OAM has been focused on writing and evaluating software designed to monitor the roadway behind a stopped police vehicle, detecting oncoming vehicles using machine vision algorithms, evaluating the potential danger of a detected oncoming vehicle, and alerting the police officer of any vehicles deemed potentially dangerous. Research for the alerting portion of the project will consist of evaluating the software system on LCM logs of camera data taken from the roadside during police escorted ride alongs.

3.1.1 Sensors and Geometry

Vision at night is very challenging. Road markings and vehicle bodies are extremely dimly lit, requiring long exposures to show sufficient detail. The long exposure time of the camera causes blur on moving vehicles, which also makes recovering details difficult. Additionally long exposure causes bright objects, such as headlights and streetlights, to saturate the camera and produce a blooming effect. With this bloom, it is very hard to precisely track bright lights. We would like to be able to minimize bloom and track headlights very precisely, but at the same time be able to see details of the road and vehicle bodies.

To accomplish these two seemingly contradictory tasks, we mount two identical cameras, one set to short exposure, and the other to a longer exposure. The short exposure camera is set around 1/2000 second exposure time, and is used to detect moving headlights and other bright lights, while minimizing motion blur. The long exposure camera is set around 1/60 second exposure time, and is used to capture details of the road, such as lane lines and other markings, which are stationary and so not subject to motion blur. Figure 3-1 shows the difference between the images from these two exposure settings.

Although we are using two cameras, we are still using a monocular vision approach. The two cameras do not function as a stereo pair, they only differ in their exposures, and are relatively close in position. We do not use a stereo system for three main reasons:

1. We would like the entire system to fit in a small box on top of the vehicle, and a stereo pair of cameras would require a large distance between the two cameras.

2. Stereo works better the closer the object is to the cameras, but our system is focused on detecting objects at a far distance, and so in this case stereo would not provide much of an advantage.

3. Stereo would require adding at least one more camera, which would bring an additional cost to the system.

A monocular vision approach implies that we must determine distance in other ways, such as using assumptions about the shape and size of the road, or assumptions about the height of headlights off the ground.

3.1.2 Vehicle Detection

One of the most critical aspects of the project is the ability to detect and track oncoming vehicles. The task of detection involves recognizing objects as vehicles and determining the precise position of those vehicles. Tracking involves keeping a

(a) Short-exposure

(b) Long-exposure

Figure 3-1: Short and long exposure images of the road. Notice the bloom in the long-exposure image, making it very difficult to differentiate headlights in the distance.

consistent representation of each vehicle's position and velocity as it moves through the cameras field of view.

The easiest way to detect vehicles is by looking for headlights. Using the short exposure camera, headlights do not bloom and so can be relatively compact, and so are seen to be distinct from one another. Because the short exposure camera is highly sensitive to bright lights, and dulled to most everything else, it is reasonably simple to detect and track vehicles with their headlights on.

3.1.3 Vehicle Tracking

Once we determine the positions of oncoming vehicles, we form a coherent track for each vehicle by correlating detections over camera frames based on sensor geometry, motion consistency, and proximity.

3.2 Sensor Calibration

We employ three levels of offline sensor calibration. First we estimate camera parameters such as focal length and radial lens distortion, which remain fixed, using a standard procedure[6]. Second we compute the alignment between the two cameras, which remain fixed. Third, we determine a transformation between the cameras and

the laser from the MDM using a sequence of calibration steps built into the software system. This laser calibration will also remain fixed in the final system, which will contain all parts of the system in one enclosure.

Chapter 4

Technical Achievements/ Steps

The majority of the work I did was focused on three main problems — vehicle detection and tracking, recovering world coordinates from road lines, and calibration with the laser system. The Divert and Alert system contains two cameras, one at long exposure and one at short exposure, aimed at the road. The short exposure camera is used for vehicle detection and tracking, as the short exposure reduces headlight bloom and darkens other objects in the image, making it easier to separate headlights. The long exposure camera is used in the officer's interface, so he can easily see road lines and reference objects along with the headlights of incoming vehicles.

4.1 Assumptions

In the current version of the system, we assume that the road is flat and straight. This assumption will not always hold, but for most cases, especially on the highway, the road is relatively flat and straight. The straight road assumption allows us to use the road lines to find the vanishing point of the image and to infer the world coordinate system from these straight road lines. We also assume a fixed headlight height of 0.6 meters above the road. This allows us to determine the real-world position of headlights once we have calculated the transformation between the camera image and the road. The process of determining real-world coordinates is described in Section 4.3. The flat road assumption allows us to perform a homography between

the laser coordinate system and the camera coordinate system, so we can calibrate the two systems and project lasers on the road using the camera interface. This procedure is outlined in Section 4.4.

4.2 Vehicle Detection and Tracking

Vehicle detection and tracking are critical to the process of alerting an officer of a dangerous incoming vehicle. Detection consists of finding blobs of pixels that correspond to headlights, while tracking consists of keeping track of headlights and more importantly headlight pairs which represent vehicles. This process is performed monocularly on the short exposure camera, so that the effects from headlight bloom and other, less bright objects are minimized. Figure 4-1 shows a screenshot of the tracker code running on a log from the short exposure camera.

4.2.1 Detection

The vehicle detection process consists of separating headlights in the image from the short exposure camera. To do this, the system first finds all pixels in the greyscale image over a threshold light value, usually 240 for an 8-bit pixel value. We put all of these pixels into a simple "Union Find" data structure. Our Union Find data structure is a simple disjoint set data structure that packs each pixel entry into a single integer array for performance.

Once we have all relevant pixels, we can separate them out into clusters. This is when our Union Find data structure becomes useful. The disjoint-set data structure allows us to cluster all connected pixels that are above the threshold light value together efficiently. Once we have clustered together all bright pixels, which we will call blobs, we assign a random ID to each blob. We then discard all blobs smaller than a minimum size value, in our system usually 40 pixels. This leaves us with only large blobs of light, which generally represent headlights. For each blob that we detect in the image, we calculate the centroid and store it as a headlight for the current frame.

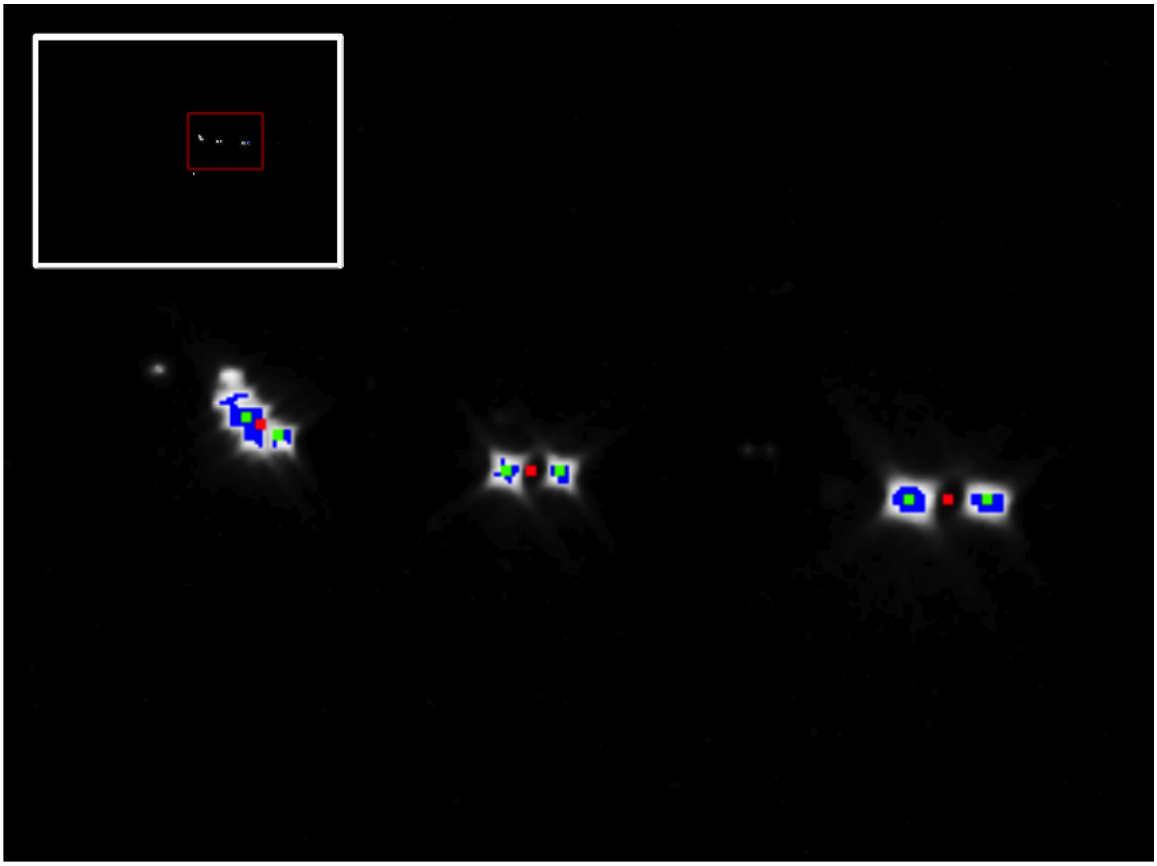


Figure 4-1: A zoomed-in image of the tracker code running on a log from the short exposure camera. The tracker first finds blobs of light pixels (blue), then finds the centroids of these blobs to track as headlights (green), then matches headlight pairs (red). The inset shows the original image, and the portion which is zoomed in on.

4.2.2 Tracking Headlights

We would like to have some sort of memory of the headlights that we detect, so we can keep track of a vehicle's trajectory and velocity. At the beginning of each frame, we load all of the headlights from the previous frame. When we look at the new blobs found in the current frame, we examine the blob IDs to determine if some of the blobs correspond to headlights carried over from the previous frame. Our Union Find data structure carries over blob IDs if they are connected to a blob in the previous frame, so as long as the blobs don't jump too far between frames, we can keep track of headlights from frame to frame, based on their blob ID.

At each frame, we check to see if each headlight is present, and when a headlight

is missing for a threshold number of frames, usually just 1 or 2, then we remove that headlight from our headlights array. In this way we can keep only the active headlights at all times, and so therefore consistently track the headlights of vehicles.

4.2.3 Headlight Pairs

Tracking headlights is not sufficient for tracking vehicles. Most, but not all, vehicles have two headlights, so we must pair our tracked headlights to track vehicles. This is slightly more complicated than just tracking blobs of light. We could have a handful of headlights in the image, but if we don't know which headlights go together, then these headlights don't do us much good. We would like to track vehicles, because if we just have unpaired headlights then we cannot know the exact position of the vehicle, and so our tracking will not be very accurate. Therefore we needed to come up with some sort of pairing algorithm.

Currently we pair by two main factors, a y-distance threshold and a pure distance threshold. In other words, if two headlights are within a small threshold number of pixels in the y direction, then we consider them a possible pair. This works because headlight pairs are horizontal, and so their positions in the image should be roughly equal in their y position. We also consider a pure distance factor, as headlights should be relatively close to each other. For instance there could be two separate headlights that have equal y values, but that are across the image from each other. These are clearly not a pair; the distance factor takes care of this case.

Other factors that we have considered are velocity factors and spacing factors. We could track the velocity of each headlight, and if a possible pair has similar velocities, then this would make them more likely to be an actual pair. Similarly, if we tracked the pixel spacing between two pairs, this should strictly increase as the two headlights approach the camera. If this does not happen, this decreases the likelihood of being an actual pair.

Note that we mention tracking possible pairs. We actually pair up all possible pairs that meet our factor thresholds, and so sometimes headlights end up in multiple possible pairs. At some point we trim down the pairs until all headlights are contained

in only one pair, based on the factors above.

4.3 Recovering World Coordinates from Road Lines

We now know where the vehicles are in the image, and can track their position, but only in the camera image. We want to know their position in the world, so we can track their distance from the vehicle, their velocity, and other relevant parameters. To find the relationship between the camera image coordinate system and the world coordinate system, we need to have some sort of basis in the camera image that we can relate to the world. For this, we decided to use lines on the road, because they are constant and are relatively easy to see on the camera image.

Once we have the road lines in the camera coordinate system and also have some idea about their position in the real world, we can calculate a transformation from the camera coordinate system to the world coordinate system, and so be able to track the real world position and velocity of each vehicle.

4.3.1 Vision Algorithms to Recover Road lines

We first define the coordinate system that we will use throughout. Let $(X_w, Y_w, Z_w)^T$ be "world coordinates" aligned with the road. The Z_w-axis is parallel to the road, the Y_w-axis is parallel to the gravity vector, and the X_w-axis is orthogonal to these two (and hence horizontal). Let Z_w increase in the direction the camera is aimed, let X_w increase to the right when facing in the positive Z_w direction and let Y_w increase downward. This gives us a right-hand coordinate system.

For convenience we pick the origin of this coordinate system to coincide with the center of projection (COP) of the camera — that way, the road and camera coordinate systems are related just by a rotation, not a rotation and a translation.

Let $(X_c, Y_c, Z_c)^T$ be "camera coordinates" with Z_c aligned with the optical axis, X_c horizontal to the right, and Y_c vertical down. The origin of this coordinate system is also at the center of projection. These coordinate systems are shown in Figure 4-2.

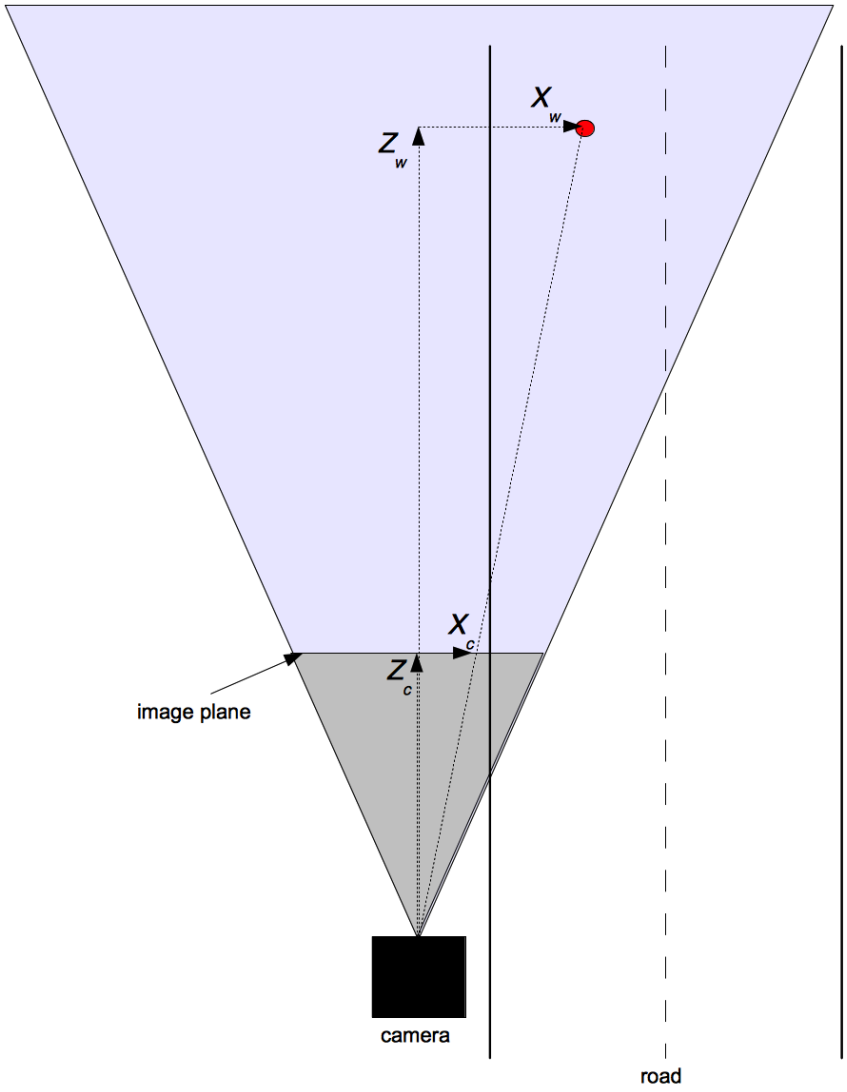


Figure 4-2: A diagram of the camera and the road showing the camera and world coordinate systems. The world coordinates X_w and Z_w of the red dot, as well as the camera coordinates X_c and Z_c, are shown. The y coordinates are not shown because the y-axis is perpendicular to the page, but both Y_w and Y_c are increasing downward.

The perspective image projection equations are

$$\frac{x}{f} = \frac{X_c}{Z_c} \quad \frac{y}{f} = \frac{Y_c}{Z_c}$$

where f is the focal length and (x, y) is the position in the image — measured relative

to the image center, or principal point.

Next, the camera is mounted on a tripod that permits two rotations, yaw θ about the Y axis, and pitch ϕ about the X axis. We then have the relationship between the world coordinate system and the camera coordinate system

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = R \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix}$$

where

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

or

$$R = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ -\sin\theta\sin\phi & \cos\phi & \cos\theta\sin\phi \\ -\sin\theta\cos\phi & -\sin\phi & \cos\theta\cos\phi \end{pmatrix}$$

From this we can get the vanishing point (x_v, y_v) of the road by projecting the rotated $(0, 0, Z_w)^T$ to obtain

$$\frac{x_v}{f} = \tan\theta\sec\phi \quad \frac{y_v}{f} = \tan\phi$$

These equations allow us to estimate yaw θ and pitch ϕ from the location of the vanishing point. We can easily find the vanishing point by finding the intersection of two parallel road lines in the camera image.

Now we can estimate how far a line parallel to Z_w, such as a road edge or lane marking, is from the camera. Suppose that the line passes a distance w to the right of the COP and a distance h below it. This road line projects into a line in the image. We will estimate the ratio w/h from the angle ξ of the road line in the image relative to the y-axis of the image. Then, knowing h we can estimate the horizontal offset w.

Points on the road line have world coordinates $(w, h, Z)^T$ and as Z tends to infinity

the projection of this point tends to the vanishing point. We need only one other point on the projection of this line into the image in order to determine it fully. We find an arbitrary point (x_c, y_c) in the image. We are only interested in the direction of the line, so we compute

$$\frac{x_c - x_v}{f} = \frac{X_c}{Z_c} - \tan\theta \sec\phi = \frac{X_c \cos\theta \cos\phi - Z_c \sin\theta}{Z_c \cos theta \cos\phi}$$

and

$$\frac{y_c - y_v}{f} = \frac{Y_c}{Z_c} - \tan\phi = \frac{Y_c \cos\phi - Z_c \sin\phi}{Z_c \cos\phi}$$

where

$$\begin{aligned} X_c &= w\cos\theta + Z\sin\theta \\ Y_c &= -w\sin\theta\sin\phi + h\cos\phi + Z\cos\theta\sin\phi \\ Z_c &= -w\sin\theta\cos\phi - h\sin\phi + Z\cos\theta\cos\phi \end{aligned}$$

We find after some simplification that

$$\begin{aligned} Y_c\cos\phi - Z_c\sin\phi &= h \\ X_c\cos\theta\cos\phi - Z_c\sin\theta &= w\cos\phi + h\sin\theta\sin\phi \end{aligned}$$

consequently

$$\frac{x_c - x_v}{y_c - y_v} = \frac{w\cos\phi + h\sin\theta\sin\phi}{h\cos\theta}$$

This is the tangent of the angle between the projection of the line and the vertical in the image, so

$$\tan\xi = \frac{w}{h}\frac{\cos\phi}{\cos\theta} + \tan\theta\sin\phi$$

This allows us to calculate w/h from the angle ξ of the projected line and the known camera orientation parameters θ and ϕ. If we know the height h of the COP above the road surface we can then estimate the horizontal offset w of the line from the camera. In our case, we will have a fixed height h as the police vehicle will have a

constant height. If we perform this calculation for two lines, such as lane markings, then we can estimate the distance between them, which would be the lane width.

4.3.2 Software System To Recover Road Lines

Using the machine vision algorithms from the previous section, I designed a software system in C++ to recover road line information given the camera image of the road. The system displays two road lines and a horizon line overlaid on the camera image of the road, which is read using LCM. The officer can adjust these three lines to meet the two edges of the left-most lane and the horizon, respectively. The system uses the intersection of the two road lines to find the vanishing point, and from there calculates the horizontal offset of the road from the camera, as well as the width of the lane. Based on the vanishing point and assuming a flat road, the system can determine the yaw θ and pitch ϕ of the camera relative to the road, and from there determine the entire transformation R from the world coordinate system to the camera coordinate system. The progression of the system is shown in Figures 4-3 and 4-4.

4.3.3 Position and Velocity Calculation

Once we have calculated the transformation between the camera and road frames, we can track the real world position and velocity of the vehicles using the tracker portion of the software system. First we need to transfer the headlight tracking information from the tracker code to the road lines code just described. This is a little trickier than simply passing the headlight pixels to the road lines code, as the tracker is working on the short exposure camera, which is a separate camera from the long exposure camera that the road lines code works on. The two cameras are spaced slightly apart, and the timings of the two

cameras are not completely in sync. The timing of the cameras does not really matter for figuring out position information, as the road lines are constant, but the difference in timing can cause a slight offset when displaying tracking information on the road lines image. This problem is hardly noticeable most of the time, but in

Figure 4-3: The road lines program running on a log of the footage from a ride along with an officer. The user adjusts the road lines shown overlaid on the picture to fit the lines of the road in the picture. The system then automatically calculates the distance to the road, the road width, and a transformation from the camera's coordinate frame to the real world coordinate frame. Based on these calculations, the system then displays meter markings at increments of 10 meters along the road.

future iterations of the project should be dealt with for increased accuracy.

To take care of the position offset between the two cameras, we calculate a camera-to-camera transformation using a simple homography, which remains constant, because the camera rig is constant. We only have to do this once, and can store the homograph in a config file. Using this simple transformation, we can integrate the tracker and the road lines code, and then determine position and velocity of tracked vehicles. To calculate position we simple use the camera-to-world transformation that we calculated above using the road lines and the vanishing point. We also pass in the previous frame's headlights from the tracker, which allows us to calculate the velocity of each tracked headlight pair. In Figure 4-5 we calculate the position and velocity of

Figure 4-4: The user can toggle the level of the road markings between road level and headlight level. When the road lines are at headlight level, it allows the user to see whether or not the car corresponding to headlights on the screen is within the boundary of the road lane. Notice that the car in the left lane has both headlights between the orange lane lines.

incoming vehicles, and mark any that are in the rightmost lane, closest to the parked police vehicle.

4.4 Laser Calibration

In addition to the OAM functions performed by the camera, the image from the camera is also used by the officer to give the system directions on where to place the laser virtual flares. The software system deduces where these flares should go based on the camera image of the road, then sends a command to the laser system for where to fire the laser. This can be tricky, however, because the laser and the

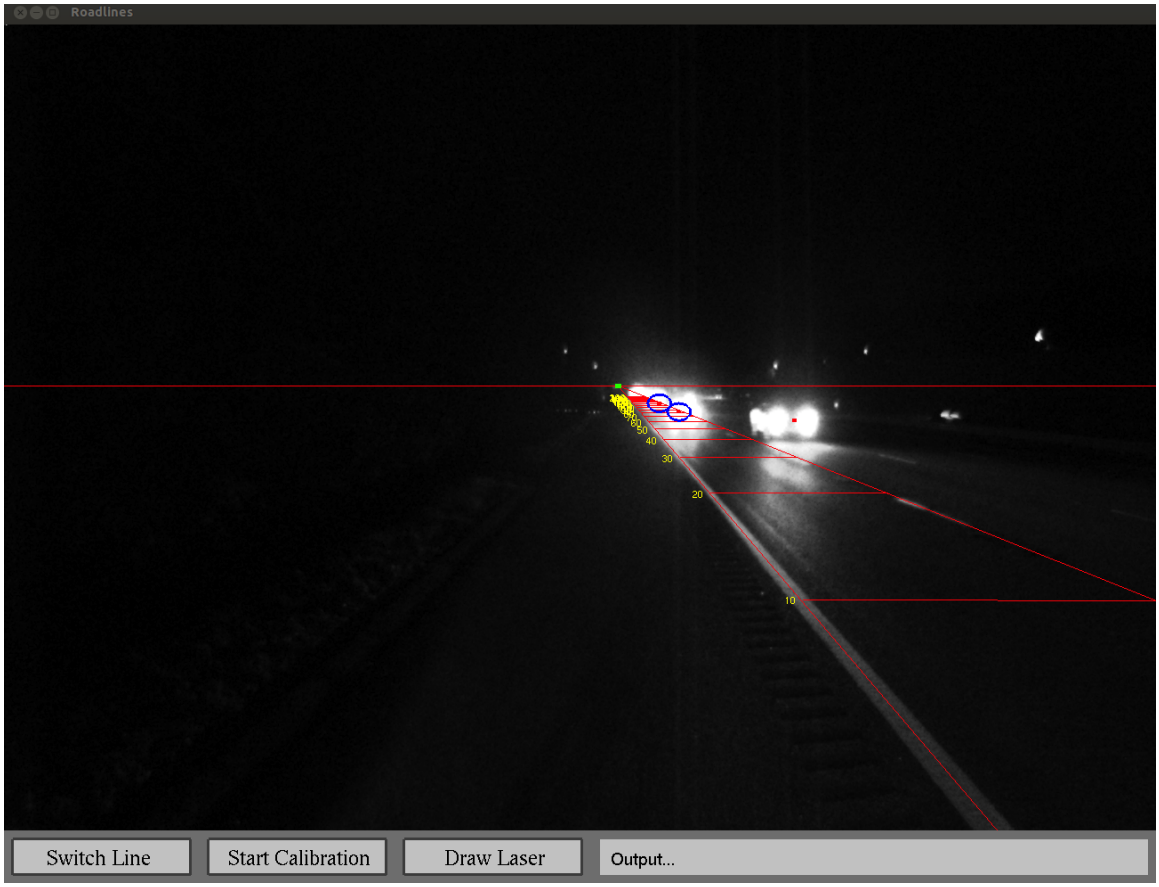


Figure 4-5: The road lines program after integration with the tracker. The red dots denote tracked headlight pairs. Blue circles denote vehicles that are in the rightmost lane, closest to the shoulder.

camera are naturally in different positions, and possibly also different orientations, upon the police vehicle. Therefore the software system must know the relationship between the camera coordinate system and the laser coordinate system. In order to determine this relationship we use the principles of homography.

4.4.1 Homography

We seek to determine the relationship between the camera coordinate system and the laser coordinate system. The most accurate way to do this is to determine the relative orientation (translation and rotation) between the laser coordinate system and the camera coordinate system, using 5 or more pairs of correspondences and a least squares method[9, 10]. However it was quicker to implement a simple homography,

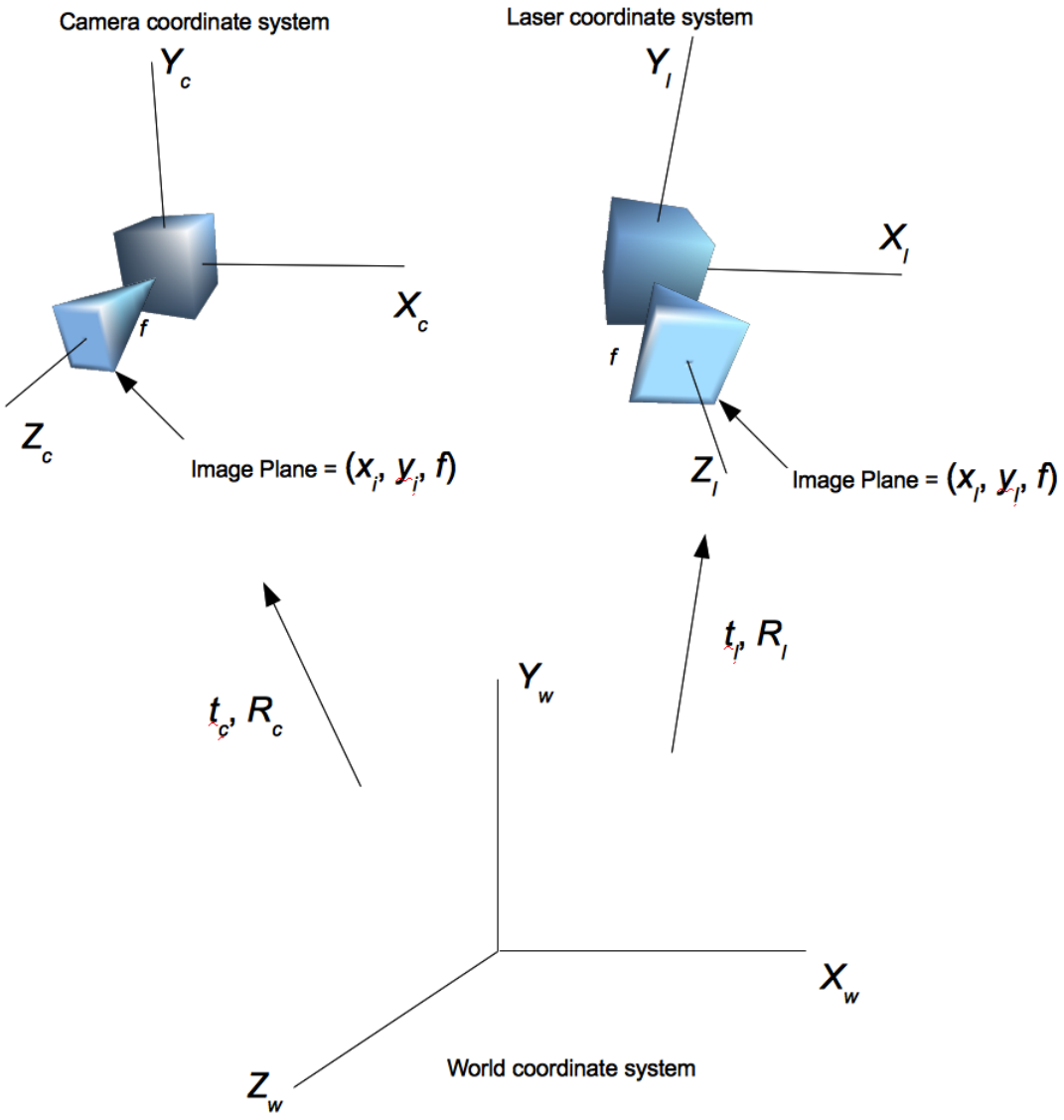


Figure 4-6: The relationships between the camera, laser, and world coordinate systems are shown. f represents focal length, t represents a translation vector, and R represents a rotation matrix.

and this doesn't cause a loss of accuracy unless the data is extremely noisy. This could cause a problem at some point, but for now the homography approach produced accurate enough results[11, 12, 8]. We can use a homograph because we assume a flat road, and so we are finding a transformation between two 2-D planes, namely the

image and the road.

In either case, there are some basic calibration parameters we need to determine. Ideally we would want to know where the principal point is, but we use the image center instead, as this is a close approximation for the scale of the road. Image coordinates are in reference to this point, not one corner or another of the image, and in the same units as the focal length.

Many of the projection equations we use are from the chapter on photogrammetry in RobotVision[7]. The camera's perspective projection equations are

$$\frac{x_i}{f} = \frac{X_c}{Z_c} \quad \frac{y_i}{f} = \frac{Y_c}{Z_c}$$

where (x_i, y_i) are the coordinates of the image of the point $(X_c, Y_c, Z_c)^T$ measured in the camera coordinate system.

It is important to note that the image coordinates x_i and y_i are measured relative to the principal point (approximately at the image center) and in the same units, in our case pixels, as those used for f, the distance from the center of projection (rear nodal point) to the image plane (approximately the focal length). The principal point is only approximately the image center because of the tangential distortion or imperfect centering of the lens components and other manufacturing defects, in other words the imperfect properties of the camera. We can generally treat the principal point as the image center, as the difference is not enough to effect us on the scale of the road. Similarly the focal length is approximate, as the center of projection can correspond to two nodal points in the imperfect lens. Again, this is not enough to effect us in a noticeable way. Further, the camera coordinates X_c , Y_c , Z_c are measured with respect to an origin at the center of projection (front nodal point) with the Z_c axis along the optical axis and the X_c and Y_c axes parallel to the x_i and y_i axes in the image plane.

The accuracy with which we can determine the required transformation will depend to some extent on the relative arrangement of the camera and the laser. If it were possible to place the center of projection of the camera right at the point of

projection of the laser, then only rotation between them would matter. But of course this is not practical. There may be an advantage to keeping them fairly close together anyway, which we will do, as the final product will be entirely contained in a box on top of the police vehicle.

With the laser, we need to determine the actual angles of deflection of the beam, which will be twice those of the mirror rotations. Next, because rotations do not commute, we need to know in which order the beam from the laser hits the two mirrors. Then the direction of the laser beam can be given as something like

$$\begin{pmatrix} \sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix}$$

where θ and ϕ are the deflection angles of the two mirrors. This expression will look slightly different, depending on the arrangement of mirrors and choice of coordinate axes, which we will examine in the following section.

There is an additional issue with the laser in that the mirrors rotate about axes that do not intersect. This makes the calculation of the exiting beam much harder, but we can ignore this based on the large ratio between the distance to the road surface (many meters) and the distance between the two mirror axes (< 10 mm).

The transformation between any arbitrary world coordinate system X_w, Y_w, and Z_w (not necessarily corresponding to the world coordinate system from the previous section) and the camera coordinate system is given by a rotation and a translation

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = R \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + \mathbf{t}$$

where R is an orthonormal rotation matrix and $\mathbf{t} = (X_0, Y_0, Z_0)^T$, is the translational offset.

When dealing with a plane, in this case the road surface, we can construct the

world coordinate system in such a way that $Z_w = 0$ for points on that surface. Then

$$X_c = r_{11}X_w + r_{12}Y_w + X_0$$
$$Y_c = r_{21}X_w + r_{22}Y_w + Y_0$$
$$Z_c = r_{31}X_w + r_{32}Y_w + Z_0$$

where r_{ij} are the components of the rotation matrix for row i and column j. We can also write this as

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & X_0 \\ r_{21} & r_{22} & Y_0 \\ r_{31} & r_{22} & Z_0 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

Using the perspective projection equation we see that

$$k \begin{pmatrix} x_i \\ y_i \\ f \end{pmatrix} = \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

where $k = (Z_c/f)$. So

$$k \begin{pmatrix} x_i \\ y_i \\ f \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & X_0 \\ r_{21} & r_{22} & Y_0 \\ r_{31} & r_{22} & Z_0 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

or

$$kf \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} fr_{11} & fr_{12} & fX_0 \\ fr_{21} & fr_{22} & fY_0 \\ r_{31} & r_{22} & Z_0 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

or

$$k_i \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = M_c \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

That is, there is a simple linear relationship between homogeneous coordinates on the world plane, in our case the road surface, and homogeneous coordinates in the image plane.

We can think of the laser as something like a reverse camera, that fires rays of light from its COP rather than receiving rays of light like the camera. We can imagine a plane a distance f in front of the device and specify the directions of rays from it by coordinates x_l and y_l where the ray intersects this plane. Then there is a similar relationship between coordinates in this plane and world coordinates on the road, just

$$k_l \begin{pmatrix} x_l \\ y_l \\ 1 \end{pmatrix} = M_l \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

We can combine these two transformations to find a transformation between laser coordinates and image coordinates:

$$k \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = M \begin{pmatrix} x_l \\ y_l \\ 1 \end{pmatrix}$$

where $M = M_l M_c^{-1}$.

The calibration task is that of finding M, a 3 x 3 matrix, using correspondences between laser coordinates (x_l, y_l) and image coordinates (x_i, y_i). The matrix M has 9 elements, but since it relates homogeneous coordinates, its magnitude does not matter. So we have only 8 unknowns — we can fix one component, say $m_{33} = 1$, and solve for the rest.

We now have

$$\begin{aligned} kx_i &= M_1 \cdot (x_l, y_l, 1)^T \\ ky_i &= M_2 \cdot (x_l, y_l, 1)^T \\ k &= M_3 \cdot (x_l, y_l, 1)^T \end{aligned}$$

where M_1, M_2 and M_3 are the three rows of the matrix M and the dot denotes dot product. Using the last equation to substitute for k in the first two equations we obtain

$$m_{11}x_l + m_{12}y_l + m_{13} - m_{31}x_i x_l - m_{32}x_i y_l - m_{33}x_i = 0$$
$$m_{21}x_l + m_{22}y_l + m_{23} - m_{31}y_i x_l - m_{32}y_i y_l - m_{33}y_i = 0$$

We obtain two such linear (homogeneous) equations in the unknown elements of the matrix M for every correspondence between a laser direction (x_l, y_l) and the coordinates of the image (x_i, y_i) of the spot that the laser makes on the surface. If we collect enough correspondences, and add the non-homogeneous equation $m_{33} = 1$, we can solve the resulting set of linear equations for the components of the transformation matrix M. Because there are 8 unknowns, we need 4 correspondences to produce 8 linear equations, which we can just solve using Gaussian elimination.

Once we have the transformation matrix M, we can then convert any image point (x_i, y_i) to a laser point (x_l, y_l). The laser, however, does not understand this coordinate system, and only takes mirror angles as input. Therefore we must find a transformation between the laser points (x_l, y_l) and the command angles to send to the laser. To figure out this transformation, we will examine the galvanometers of the laser in greater detail.

4.4.2 Galvanometers

The laser contains two galvanometer-driven mirrors, which control the direction of the laser beam. First, let's define the coordinate system for the galvanometers.

Let Z be the direction of the beam out of the mirror system at zero deflection (i.e. perpendicular to the plate with the three screws visible in Figure 4-7). Let X be the direction opposite to that of the laser beam where it comes into the mirror system (i.e. parallel to the line between the bottom two screws on the front plate in the figure). Let Y be upward, perpendicular to the X- and Z-axis (i.e. parallel to the line connecting the two left screws in Figure 4-7). These axes form a right-hand

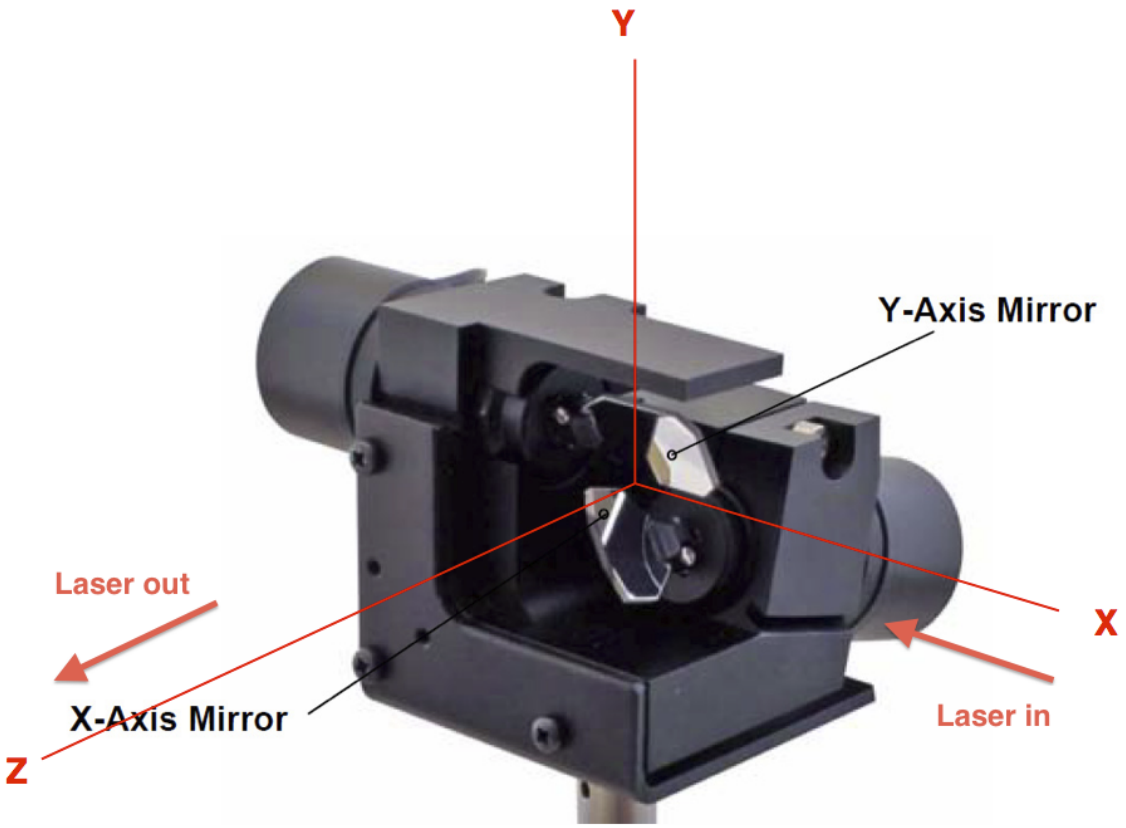


Figure 4-7: The coordinate system for the galvanometers.

coordinate system.

Next, we define the positive angle of rotation of the galvanometers as clockwise when viewed from a point along the axis on the mirror side of the motor. The deflection of a beam by a mirror can be described by

$$\hat{\mathbf{b}}_{\text{out}} = \hat{\mathbf{b}}_{\text{in}} - 2(\hat{\mathbf{b}}_{\text{in}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}$$

where $\hat{\mathbf{b}}_{\text{in}}$ is a unit vector in the direction of the incident beam and $\hat{\mathbf{b}}_{\text{out}}$ is a unit vector in the direction of the reflected beam and $\hat{\mathbf{n}}$ is the unit normal to the mirror. We can verify that the emitted angle equals the incident angle, and that the three

vectors $\hat{\mathbf{b}}_{\text{in}}$, $\hat{\mathbf{b}}_{\text{out}}$, and $\hat{\mathbf{n}}$ are coplanar using

$$\begin{aligned}
\hat{\mathbf{b}}_{\text{in}} - \hat{\mathbf{b}}_{\text{out}} &= 2(\hat{\mathbf{b}}_i \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} \\
(\hat{\mathbf{b}}_{\text{in}} + \hat{\mathbf{b}}_{\text{out}}) \cdot \hat{\mathbf{n}} &= 0 \\
(-\hat{\mathbf{b}}_{\text{in}}) \cdot \hat{\mathbf{n}} &= \hat{\mathbf{b}}_{\text{out}} \cdot \hat{\mathbf{n}} \\
[\hat{\mathbf{b}}_{\text{in}}\ \hat{\mathbf{b}}_{\text{out}}\ \hat{\mathbf{n}}] = (\hat{\mathbf{b}}_{\text{in}} \times \hat{\mathbf{b}}_{\text{out}}) \cdot \hat{\mathbf{n}} &= 0
\end{aligned}$$

Next, we consider the orientation of the two mirrors. The axis of rotation of the second, "Y-axis" mirror is $\hat{\mathbf{x}} = (1, 0, 0)^T$. The normal to this mirror is

$$\hat{\mathbf{n}}_2 = (0, c_2, s_2)^T$$

where $c_2 = \cos(\theta_2 + \theta_{20})$, $s_2 = \sin(\theta_2 + \theta_{20})$, with θ_2 being the rotation of the axis of that galvanometer from its rest position and θ_{20} being the rest position angle ($\approx \pi/4$).

The axis of rotation of the first, "X-axis" mirror is $(0, s_\epsilon, c_\epsilon)^T$ where $c_\epsilon = \cos\epsilon$, $s_\epsilon = \sin\epsilon$ and ϵ is the angle of inclination of the galvanometer axis with respect to the horizontal plane. Measurement of the drawing from the manual of the galvanometers indicates that $\epsilon \approx 11.5°$[20]. The normal to the first mirror is

$$\hat{\mathbf{n}}_1 = (s_1, c_1 c_\epsilon, -c_1 s_\epsilon)^T$$

where $c_1 = \cos(\theta_1 + \theta_{10})$ and $s_1 = \sin(\theta_1 + \theta_{10})$, with θ_1 being the rotation of the axis of that galvanometer from its rest position and θ_{10} being the rest position angle ($\pi/4$).

The beam initially enters the assembly traveling in the negative X-direction

$$\hat{\mathbf{b}}_0 = -\hat{\mathbf{x}} = (-1, 0, 0)^T$$

After reflection by the first mirror the beam direction is $\hat{\mathbf{b}}_1 = \hat{\mathbf{b}}_0 - 2(\hat{\mathbf{b}}_0 \cdot \hat{\mathbf{n}}_1)\hat{\mathbf{n}}_1$ or,

after some simplification,

$$\hat{\mathbf{b}}_1 = \begin{pmatrix} C_1 \\ S_1 c_\epsilon \\ S_1 s_\epsilon \end{pmatrix}$$

where $C_1 = \cos(2(\theta_1 + \theta_{10}))$, and $S_1 = \sin(2(\theta_1 + \theta_{10}))$ (note upper case C_1 and S_1 now for trigonometric functions of double angles).

After reflection by the second mirror the beam direction is $\hat{\mathbf{b}}_2 = \hat{\mathbf{b}}_1 - 2(\hat{\mathbf{b}}_1 \cdot \hat{\mathbf{n}}_2)\hat{\mathbf{n}}_2$ or, after some simplification,

$$\hat{\mathbf{b}}_2 = \begin{pmatrix} C_1 \\ S_1(-C_2 c_\epsilon + S_2 s_\epsilon) \\ S_1(C_2 s_\epsilon - S_2 c_\epsilon) \end{pmatrix}$$

where $C_2 = \cos(2(\theta_2 + \theta_{20}))$, and $S_2 = \sin(2(\theta_2 + \theta_{20}))$ (note upper case C_2 and S_2 now for trigonometric functions of double angles). This can also be written as

$$\hat{\mathbf{b}}_2 = \begin{pmatrix} C_1 \\ -S_1 C_2' \\ S_1 S_2' \end{pmatrix}$$

where $C_2' = \cos(2(\theta_2 + \theta_{20}) - \epsilon)$ and $S_2' = \sin(2(\theta_2 + \theta_{20}) - \epsilon)$. Note how we can compensate for the angular offset of the axis of the first galvanometer by adjusting the at-rest position of the second axis so that the exit beam travels parallel to the Z-axis when the galvanometers are in their rest position. All we need to do is set $\theta_{20} = \epsilon/2$.

Finally, if $\theta_{10} = \pi/4$, then $S_1 = \cos(2\theta_1)$ and $C_1 = -\sin(2\theta_1)$. So we end up with

$$\hat{\mathbf{b}}_2 = \begin{pmatrix} -\sin(2\theta_1) \\ \cos(s\theta_1)\sin(2\theta_2) \\ \cos(s\theta_1)\cos(2\theta_2) \end{pmatrix}$$

This is the actually form of Equation ... from the previous section. Note the negative

sign on the X-component which in practice is taken care of by considering counter-clockwise rotation of galvanometer 1 as positive. Also, note the double angles due to the fact that reflection of light doubles the angles. The rotation commands to the galvanometers already take this into account and correspond to $2\theta_1$ and $2\theta_2$.

Projecting the exit beam onto a planar surface perpendicular to the Z-axis at distance f, which corresponds to a laser point (x_l, y_l), we obtain

$$\begin{pmatrix} x_l \\ y_l \\ f \end{pmatrix} = f \begin{pmatrix} -\tan(2\theta_1)\sec(2\theta_2) \\ \tan(2\theta_2) \\ 1 \end{pmatrix}$$

for the position of the spot. Note that the y_l-position is affected only by θ_2, while the x_l-position depends on both θ_2 and θ_1. The $\sec(2\theta_2)$ multiplier stretches the x_l-component when θ_2 is non-zero.

Conversely, we can get the galvanometer commanded angles from desired positions in the plane using

$$-2\theta_1 = \arctan\left(\frac{x_l}{\sqrt{f^2 + y_l^2}}\right)$$
$$2\theta_2 = \arctan\left(\frac{y_l}{f}\right)$$

This provides the galvanometer command angles given a desired position in the projection plane. We can use these formulas to determine the command angles to send to the laser to fire the beam in a desired direction (x_l, y_l).

4.4.3 Calibration Software System

I designed a calibration process, and added it to the road lines software system. The procedure consists of a series of steps performed by the user of the software system, which will most likely be the officer. The officer follows instructions on the screen, and the software system runs the algorithms described in the previous sections to determine a transformation M between the camera and laser coordinate systems. The

software communicates with the laser using LCM, sending it galvanometer command angles.

Generally the camera and the laser coordinate systems are close together and pointed in generally the same direction, although this procedure works for any arrangement of the laser and camera. The system has no knowledge of the laser's orientation, and only can see through the camera coordinate system. The system initially assumes an identity transformation between the two coordinate systems, although this is never actually the case, as even if the camera were right on top of the laser and pointed in the exact same direction, there is some scale factor that differs in the two coordinate systems. By the end of the calibration process the system knows M and can draw any arbitrary series of laser points in the real world based on the user's clicks on the camera image.

The steps of the calibration process are as follows:

1. First the user clicks on the 'Start Calibration' button in the menu bar. Directions are displayed in the output section of the menu, which tells the user to click in a quadrilateral on the road surface to send to the laser.

(a) User clicks on 'Start Calibration'.

(b) The system prompts the user to click four corners of a quadrilateral.

Figure 4-8: Step 1.

2. The user clicks on the four corners of a quadrilateral. The system calculates the corresponding galvanometer command angles, based on its current idea of the

transformation M, which is initially just the identity. The laser points corresponding to the clicked corners then appear on the world surface. Additionally a plot of the galvanometer command angles appears, as well as a graph of the clicked points in the laser coordinate system.

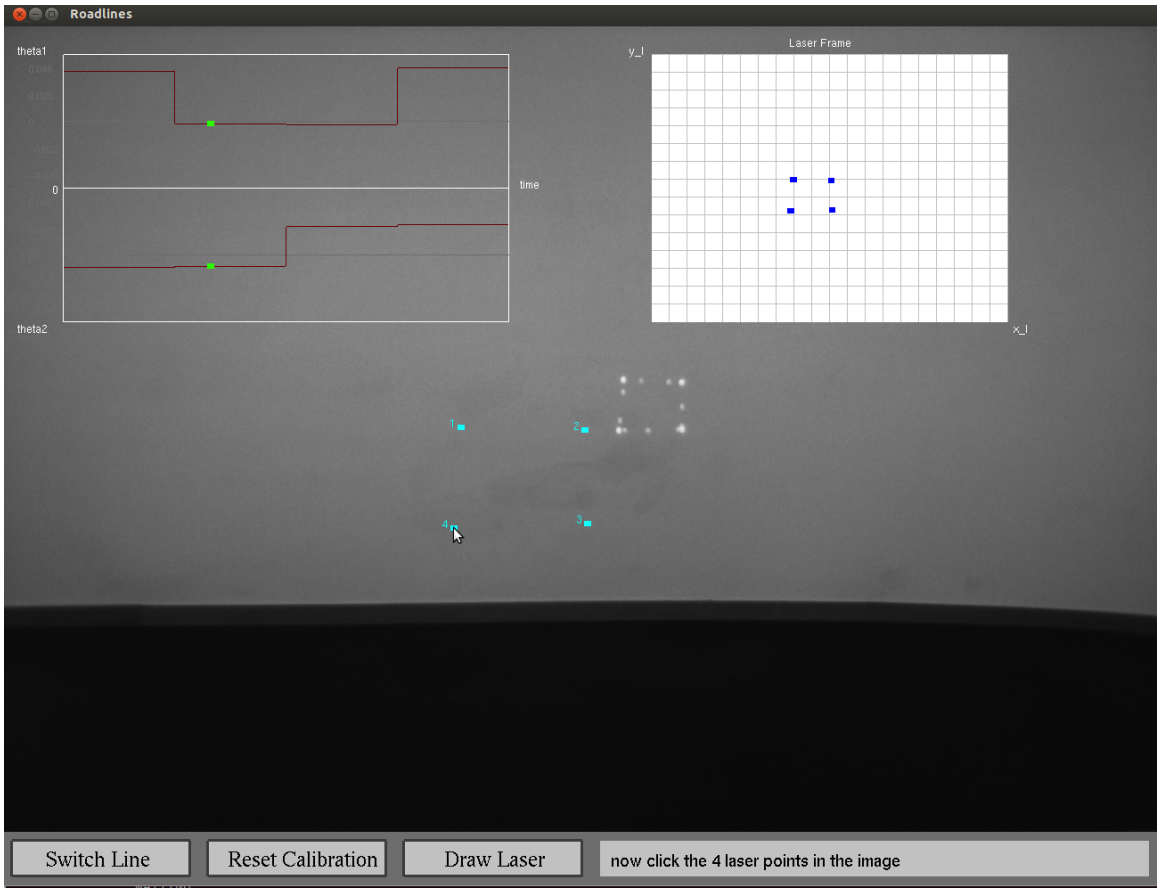


Figure 4-9: Step 2. The user clicks a quadrilateral (light blue), which the system transforms to a series of laser points, as seen to the upper right of the clicks. A plot of the galvanometer command angles appears in the top left of the screen, and shows the angles of the two motors, with a green dot that follows the angles being sent by the software to the laser. In the top right of the screen a plot of the clicks in the laser coordinate frame appears, with points in blue.

3. The user is prompted to click on the four laser points as they appear on the screen, in the order corresponding to the order clicked. With each clicked laser point, the system gets a laser-camera correspondence, as described in the section on homography. Once the system has four such correspondences, it can perform the algorithms previously described to calculate the transformation matrix M.

After these four clicks, the system should be fully calibrated. If the calibration is a little off, then the user can reset the calibration process and start over.

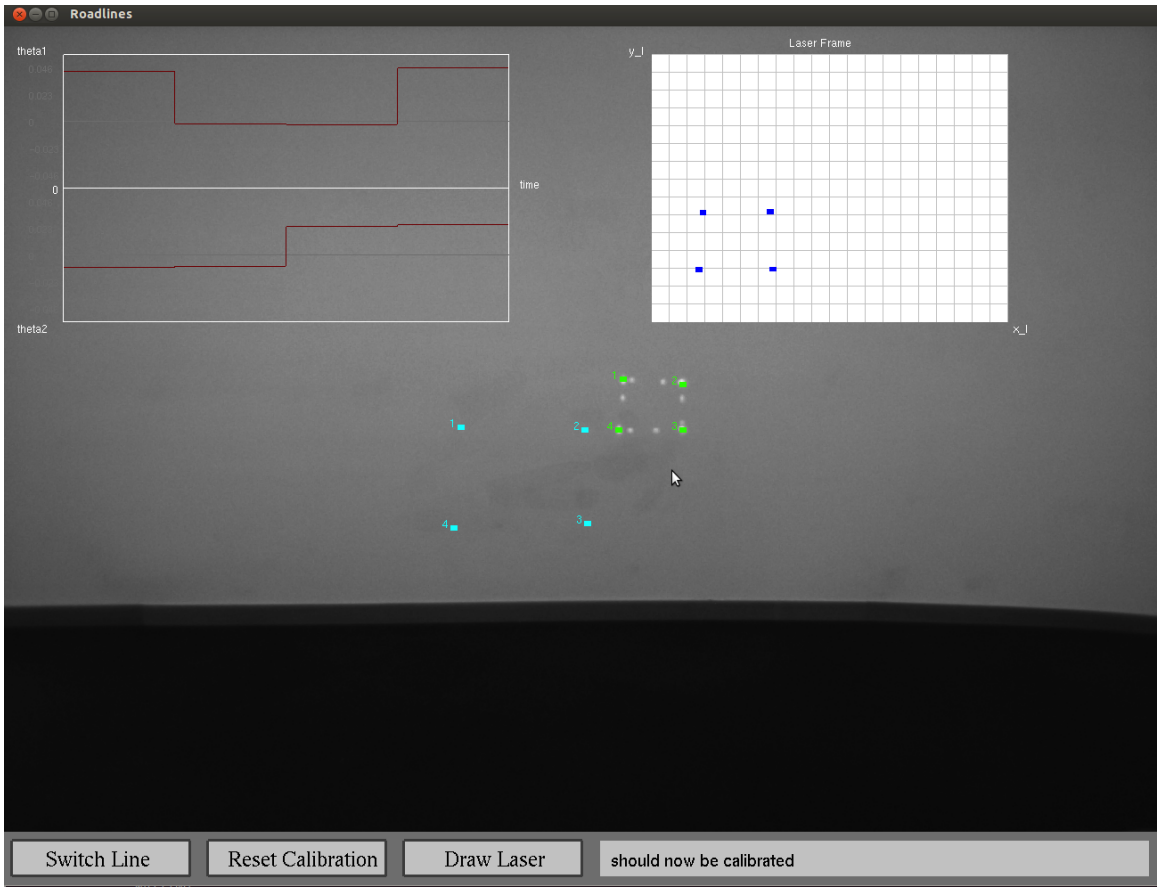


Figure 4-10: Step 3. The user clicks on the laser points corresponding to the corners of the quadrilateral. These clicks appear in green. Notice the blue points in the laser frame have changed position, as the transformation matrix M has been updated, and so the laser frame coordinates of the calibration quadrilateral have shifted.

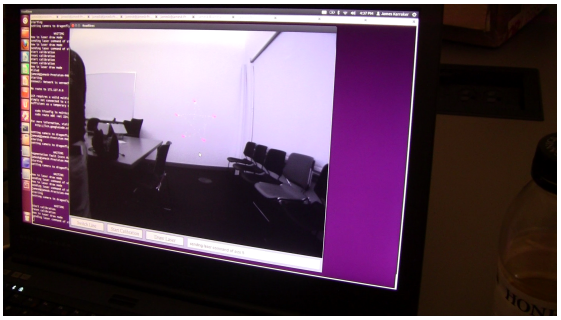
4. The user can now click on the 'Draw Laser' button to command the laser to draw out any arbitrary shape. The user clicks on an arbitrary series of points on the world surface in the image, and the system stores an array of galvanometer command angles corresponding to these image points.

5. When the user clicks on the 'Send Laser' button, the array of galvanometer command angles is sent to the laser, and the laser outputs the desired points on the world surface.

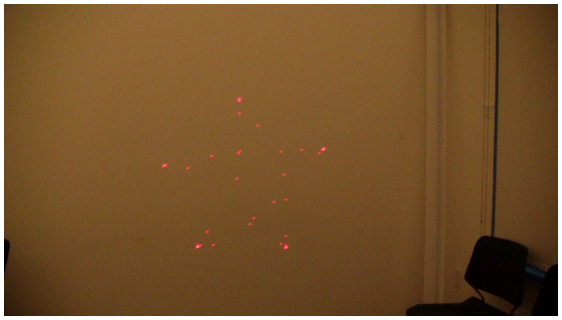
Figure 4-11: Step 4. The user clicks the 'Draw Laser' button, and clicks a series of points in a pattern on the world surface. Notice the order of the points clicked.

This procedure allows the user to easily calibrate the camera and laser coordinate systems. A demonstration of the procedure in action is shown in Figure 4-13. This should only need to be done once for the final product, but depending on how static all of the parts are, can be done occasionally and very quickly to keep the two systems calibrated. The laser drawing functionality will allow the officer to design a pattern of virtual flares on the road surface with a simple series of clicks, saving him or her time and energy on placing real flares on the road.

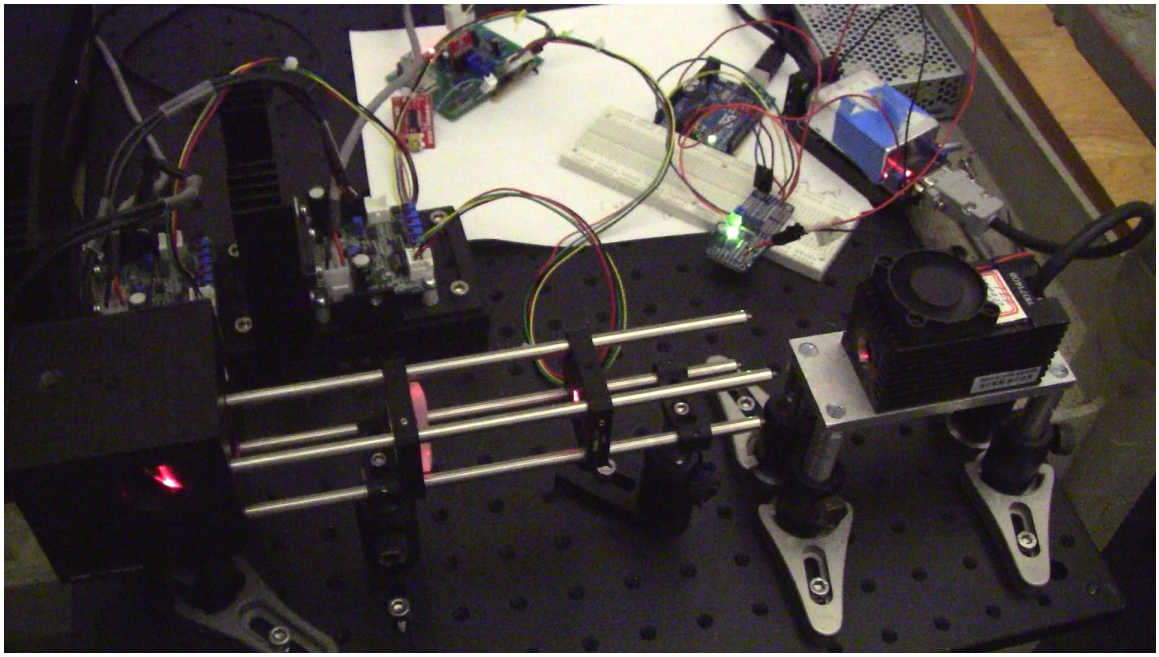
Figure 4-12: Step 5. The user clicks the 'Send Laser' button, and the galvanometer command angles corresponding to the pattern are sent to the laser. Notice the star pattern. The laser in its current state moves between laser points slower than the final laser will, and some intermittent laser points are produced in between each point. Also notice the curve of the lines between each laser point. This is due to the angular nature of the galvanometers, and so the shortest path between two mirror positions does not necessarily translate to the shortest path on the world surface. Finally note that this example of the system uses a flat vertical wall, instead of a horizontal road. This should not make any difference, as the system only assumes a flat surface, and the code has been run on a horizontal surface and works just as well.



(a) A user running the calibration procedure on a laptop.



(b) The lasers projected on a wall.



(c) The laser system. The laser source is on the right, and the laser travels through a series of optical lenses in the middle, then enters the galvanometers on the left and is projected out of the hole on the left facing forward.

Figure 4-13: A demo of the laser calibration procedure.

Chapter 5

Results and Contributions

5.1 Results

This section will present some results, my contributions, and future work to be done.

5.1.1 Specifications

The computer running the code was a Dell Precision Mobile M4600, with a 2.70 GHz Quad Core Intel Core i7-2960XM processor, with 8.0 GB RAM. The cameras are Point Grey Dragonfly2 V1.3 cameras. Figure 5-1 shows the group testing at the BAE facility in Merrimack, NH.

5.1.2 Performance

The tracker system and the road lines system each run concurrently in real time on camera logs that are running at 20 frames per second. The system takes about 4 ms per frame to find the relevant pixels that could be headlights, about 32 ms per frame to pull out the headlights from these relevant pixels and manage the union find data structure, and about 8 ms per frame to run all of the road lines algorithms and display the image and tracks to the user. This adds up to about 44 ms per frame, which is under the 50 ms between frames, but it is close. The efficiency of the headlight detector should be improved in the future, so that any performance hiccups

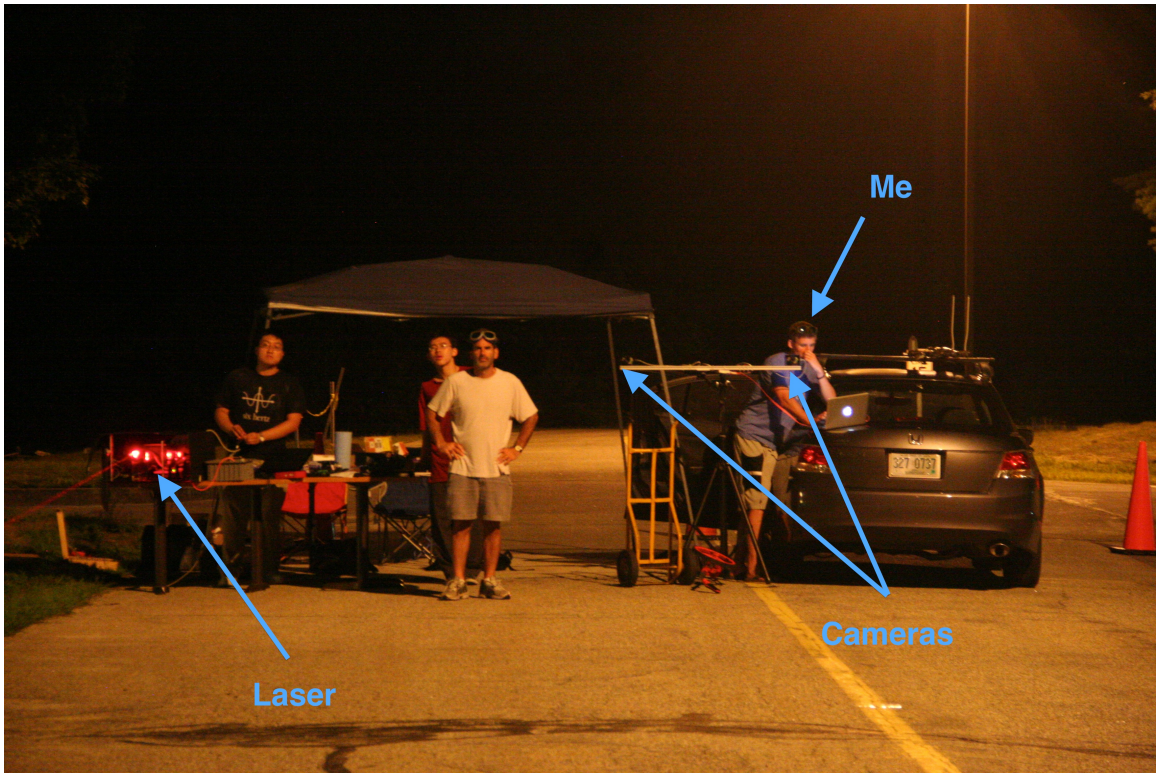


Figure 5-1: The group testing the laser and camera systems at the BAE facility in Merrimack, NH.

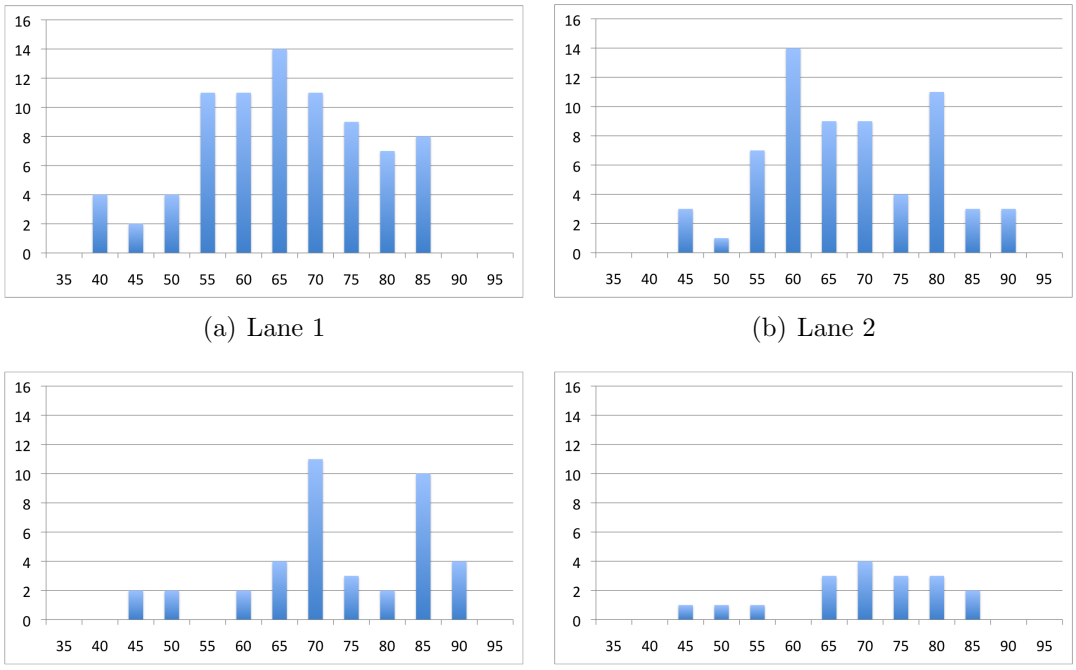
will not cause any problems.

5.1.3 Lane Characterization

As a demonstration of the correctness of the system, I recorded the z-direction velocities of headlights and compared them to the lane that the system deduced the headlights belonged to. I recorded these values and correlations for a 93-second LCM log of footage taken on a police ride along on a highway about 20 minutes outside of Boston, with relatively heavy traffic and a speed limit of 65 mph. The average speed breakdown for each lane is shown in Table 5.1, and Figure 5-2 shows the distribution of the velocity data for each lane.

Lane	1	2	3	4
Number of Headlights	81	64	41	18
Average Velocity (mph)	68.09	69.71	75.65	72.66

Table 5.1: The number of headlights and average velocity for headlights detected in each lane. Lane 1 is the rightmost lane closest to the shoulder and the camera.



(a) Lane 1

(b) Lane 2

(c) Lane 3

(d) Lane 4

Figure 5-2: Velocity distributions for each lane of the road. The velocities on the x-axis are in miles per hour.

Analysis

This experiment is not the most rigorous test of the system, as there are a few small problems with the experiment and the results. First, it only records velocities from tracked headlights, rather than headlight pairs. The system currently can track headlight pairs, but does not do so as accurately as with single headlights, and finds many false positive pairs due to the high traffic and clustering of headlights. Headlights also are dropped occasionally, and so headlight pairs disappear when either headlight is dropped; the pair tracker thus has double the drop rate as headlights.

Second, as mentioned some headlights are dropped, in which case if they reappear in subsequent frames they will be double counted. This seems to happen consistently

across lanes, so should not muddle the distributions of the data.

Third, because of the traffic, the system detects fewer headlights in the farther lanes, as many of them are obscured by closer vehicles, and their headlights are relatively smaller then those of vehicles in the closer lanes. This accounts for the decreasing number of headlights in farther lanes. This should be fine though, because we are really just concerned with vehicles that pose a danger to the officer, which are almost always vehicles in the rightmost lane.

The speed limit on the stretch of highway where this log was taken is 65 mph, so these numbers make sense, as most drivers drive slightly above the speed limit. As the lane gets farther away from the shoulder, there is an increase in average speed (for the most part), which also makes sense. The distributions are somewhat Gaussian, which is a good sign, and their deviation from Gaussian may be due to the relatively small sample size.

5.2 Contributions

In this thesis I made four main contributions:

1. I designed and implemented a laser calibration procedure, which takes click input from the user to determine a homography between the camera coordinate system and the laser coordinate system. After calibration, I added functionality to allow the user to draw an arbitrary pattern on the screen with mouse clicks, then be able to send the pattern to the laser to fire points into the corresponding points on the road surface.

2. I built a vehicle detection and tracking system that looks back at oncoming cars from the road shoulder and detects oncoming headlights. The system then tracks the headlights, pairs them up, and tracks the resulting vehicle headlight pairs.

3. I developed a software system displays the camera image of the road, takes road lines as input from the user, and calculates the transformation from the image

coordinate system to the world coordinate system. This allows the system to determine the real world coordinates of vehicles on the road based upon their coordinates in the image. I then integrated this system with the tracker, and implemented additional functionality such as lane characterization and velocity calculation for tracked headlights.

4. I devised and ran an experiment to test the correctness of the software system that recorded velocities of vehicles in a heavy traffic log, and characterized them by lanes. I then analyzed this data.

5.3 Future Work

There is much left to do on this project. The tracker needs a lot of honing to become as accurate and robust as the application calls for. There is much to be done on characterizing dangerous vehicles, which will be a critical function of the final product. The user interface for the final design will need to require much less user input and interaction. The final product will need to go through a vigorous process of real-time performance evaluation, in conjunction with the Massachusetts State Police.

5.3.1 Vehicle Detection and Tracking

The vehicle detection and tracking system works, but has some issues. Headlights are often dropped, where for a frame the detecter will not see the blob of light, either because it is to small or not bright enough or some other bug. In this case a headlight will disappear from the tracker for a frame, and when it reappears will have a new ID.

Similarly, occasionally the tracker will give a headlight that was actually also detected in the previous frame a new ID. This occurs because the blob detection algorithm uses a union find data structure, and so if the two images of the blob do not overlap, then the system will have no way of knowing that they are in fact the same headlight. This happens a lot when the headlights are close to the camera and

moving fast, and so moving many pixels per frame. This is not a huge problem, as by the time a vehicle is that close to the camera the decision of whether the vehicle is dangerous will have already been made to give the officer enough time to get out of the way.

We could also perform more sophisticated techniques, such as cross-sensor cueing and many-to-many association of detections, to improve the robustness of the tracks.

5.3.2 Headlight Pairing

We currently only pair headlights based on a few simple factors, such as vertical proximity and total pixel distance. This leads to headlights belonging to multiple pairs. Other factors that could be used to pair headlights are velocity factors and spacing factors. We could track the velocity of each headlight, and if a possible pair has similar velocities, then this would make them more likely to be an actual pair. Similarly, if we tracked the pixel spacing between two pairs, this should strictly increase as the two headlights approach the camera. If this does not happen, this decreases the likelihood of being an a actual pair.

A more robust way of doing this, which we have considered but not implemented, is to weight each possible pair based on all of the factors above, then run a constantly updating maximum weighted matching algorithm to determine the most likely set of actual headlight pairs. This may provide some possible performance issues, and added complexity, so we decided against implementing it, but it could be very useful in future iterations.

It is much more difficult to detect vehicles with their headlights off. For this we could use the long exposure camera by trying to find what is moving in the field of view. We could use optical flow, which is the pattern of motion in a visual scene, and the well-understood techniques for motion detection and object segmentation using optical flow, to find potential vehicles. The cameras should also have variable, software controlled exposure times, so that the system can adjust based on lighting conditions.

5.3.3 Characterizing Dangerous Vehicles

There are many ways to characterize the tracks of vehicles with varying complexities. Simple characterization techniques, such as examining speed and direction, can classify a vehicle as moving too fast or directly toward the camera, and therefore classify that vehicle as dangerous. Contextual information such as models of the roadway, lane geometry, and vehicle size can be used to determine vehicle speed and acceleration in real-world units, rather than pixels. Optical flow can be used to calculate a simple time-to-collision estimate. A slightly more complex technique is to look at night for vehicles with their headlights off, by finding objects that have a large amount of directed motion, but that do not appear in the images from the short exposure camera.

Even more complex techniques include machine learning algorithms to infer "normalcy" of non-dangerous traffic. This involves storing all of the trajectories of traffic that passes by and triggering an alert when an oncoming vehicles path varies in some significant way from the norm, such as by swerving or decelerating an exaggerated amount. Normalcy can also be modulated by other contextual information, such as known speed limits, road curvature information from GPS-referenced databases, number of lanes, weather conditions, and current traffic advisories. Additional information, such as turn signals, fog lights, or hazard lights, can inform us of anomalous behavior, such as motion inconsistent with the turn signal.

Other indicators, such as a turn signal or other blinking lights, could also be detected and tracked on disjoint but globally consistent trajectories. For example, turn signals appear as dotted lines. We could use co-occurrence analysis and geometric reasoning to group these indicators with the vehicles that they belong to, and use this information as an even more detailed classification technique.

We could use a combination of all of these techniques to characterize oncoming vehicles. We could determine the most effective combination of all of these techniques based on comparing the results during testing to some ground truth conditions.

5.3.4 User Interface

We allow the police officer to manually adjust the road lines visual interface. We could combine visual lane boundary detection with contextual information such as GPS-referenced maps to determine the location and orientation of the cruiser and generate the safety zones automatically. Ideally the officer will only have to power on the system, and the code will automatically produce virtual flares in the right places and begin tracking and characterizing vehicles.

5.3.5 OAM Performance Evaluation

We currently have tested only logs of video data, but in the future we would like to evaluate the performance of the OAM prototype by live testing under controlled conditions on a closed course with the collaboration of trained police officers. The police officers would repeatedly drive around the course and drive by a vehicle with the MDM/OAM detection device. The police testers would drive in a number of patterns that should or should not be classified as dangerous as determined by us. We would tabulate these situations, as well as the ground truth for each situation, or whether we believe it should have been classified as dangerous. We would compare the outcomes of the detection device with our ground truths, and hope to minimize the number of false positives and missed detections. We would gather these statistics over hundreds of individual tests, with varying specifications, such as along a long straight stretch of road, near a curve on either side of the cruiser, near a section of lane merging, and near a section of lane splitting. We would use the results of these tests to analyze and improve the performance of the OAM and the algorithms controlling it.

Appendix A

Setup Instructions

A.1 Checkout the Repository

Here is the svn checkout command: svn checkout svn+ssh://<csail_id>@login/afs/csail.mit.edu/group/rvsn/repositories/nij nij.

There will be a few points where it runs into an error, and says that it is missing some function. Google the function and find out what package the function is in, then sudo apt-get install that package.

After you get the whole repository checked out, go into nij/software/ and type "make". This will build the code, and again you will probably run into errors where you need to apt-get install some packages.

Once this finishes, you are ready to run the code. All of the tracker/roadlines code is in nij/software/headlight_tracker/src/. After making changes, type "make" in nij/software/headlight_tracker/ to build.

A.2 Run the Headlight Tracker Code

For this go into software/build/bin/ and type "./nij-headlights -c dragonfly". An LCM input must be running for this code to run.

LCM input options:

1. Playback an LCM log file. For this go into the directory of the log file, then type "lcm-logplayer-gui [log name]". A good example log file is logs/Ride_Along_November/log5.

2. Run on input from the cameras. Plug the cameras into the firewire port of the laptop. In build/bin/ type "./camview" and add Point Grey Research Dragonfly2 to the chain. Then add OpenGL to see the camera image, then add LCM Image Publish to publish the image to the code. Set the channel of the short-exposure camera to CAMLCM_IMAGE and the long-exposure camera to CAMLCM_IMAGE_HIGH. (If the image from the camera is split, lower the packet size to around 2600 to fix it)

A.3 Run Laser

The laser code is in /nij/software/firmware/serial_interface. To run the laser to listen to LCM commands from the tracker code type "sudo python listener.py".

A.4 Run Alert code

The alert code is written in Java, so open Eclipse. The code that listens for LCM messages from the tracker, and makes an alert sound when necessary is the file MySubscriber.java. Just run this file in Eclipse.

Appendix B

Code

Here is a portion of the headlight tracker code. This shows the basic process of detecting and tracking headlights.

```
// called every frame, true if there is a new image from LCM
if (camGrab_pointer->getCameraData(greyImage, greySize) == true)
{
    // finds all pixels above a threshold
    filter::relevantPixels rp;
    filter_pointer->getRelevantPoints(greyImage, &rp);

    // clusters them into a union find data structure
    headlightEstimator::idxAndBlobIDArray newBlobIndices;
    filter_pointer->colorSegmentsVoting(&rp, &newBlobIndices);

    // extracts headlights and pairs from relevant pixels
    h_estimator_pointer->extractFromInput(&newBlobIndices);

    // updates headlights and pairs data every frame
    h_estimator_pointer->updateHeadlights();
    h_estimator_pointer->updateHeadlightPairs();
```

```
    // stores relevant tracking data to be passed to roadlines code
    headlightEstimator::idxAndBlobIDArray headlightBlobIndices;
    headlightEstimator::idxAndBlobIDArray headlightPrevBlobIndices;
    headlightEstimator::idxAndBlobIDArray pairBlobIndices;
    headlightEstimator::idxAndBlobIDArray pairPrevBlobIndices;
    h_estimator_pointer->publishHeadlightTracking(
        &headlightBlobIndices, &pairBlobIndices,
        &headlightPrevBlobIndices, &pairPrevBlobIndices);
    rlines_pointer->setTracking(&newBlobIndices,
        &headlightBlobIndices, &pairBlobIndices,
        &headlightPrevBlobIndices, &pairPrevBlobIndices);

    free(rp.pix);
    free(newBlobIndices.array);
}
```

Here are some of the data structures that are used in the tracking code.

```
typedef struct {
int blob_ID;
coordinates_2D centroid;
int framesMissing;
bool isDeleted;
} headlight;

typedef struct {
headlight headlight_1;
headlight headlight_2;
coordinates_2D midpoint_2D;
coordinates_3D midpoint_3D;
```

```
float velocity;
float pixelDistanceBetweenLights;
bool isDeleted;
} headlightPair;

typedef struct {
int ID;
int centroidX;
int centroidY;
int* previousCentroidsX;
int* previousCentroidsY;
int R_average; //average radius for the blob size
} blob;

typedef struct {
headlightPair *headlightPairs;
int num_pairs;
int num_pairs_active;
headlight *headlights;
int num_headlights;
int num_headlights_active;
std::map<int, int> blob_IDs;
std::map<int, int> pair_IDs;
} frame;
```

Here is a selection of the machine vision algorithms used in the road lines code.

```
float phi; //yaw about Y axis (horizontal), with right positive
float theta; //pitch about X axis (vertical), with down positive
float xi_left; //angle of image line of left road edge from y-axis
float xi_right; //angle of image line of right road edge from y_axis
```

```
void roadlines::calculateVelocityForPair(int pair_ID,
double x_p, double y_p, double &v_x, double &v_z)
{
    v_x = 0.0;
    v_z = 0.0;
    // check to see if this pair was in the previous frame
    for (int i = 0; i < pairsPrevTrackIDArray->size; i++) {
        if (pairsPrevTrackIDArray->array[i].val == pair_ID) {
            double cam[3], world[3], x_prev, y_prev, x_prev_p,
                y_prev_p, cam_prev[3], world_prev[3];
            this->convertPixelToMM(x_p, y_p, cam[0], cam[1]);
            cam[2] = focal_length;
            this->convertCamToWorldCoordinates(cam, world,
                cam_height - headlight_height);

            this->convertIndexToPixel(
                pairsPrevTrackIDArray->array[i].idx, x_prev, y_prev);
            this->convertCameraToCamera(
                x_prev, y_prev, x_prev_p, y_prev_p);
            this->convertPixelToMM(
                x_prev_p, y_prev_p, cam_prev[0], cam_prev[1]);
            cam_prev[2] = focal_length;
            this->convertCamToWorldCoordinates(
                cam_prev, world_prev, cam_height - headlight_height);

            v_x = world[0] - world_prev[0];
            v_z = -(world[2] - world_prev[2]);
}
}
```

```
}

// find distance to the road from the car, and the lane width,
// based on camera height and user input roadlines
void roadlines::findDistances()
{
    // tan(xi) = (w/h)*(cos(theta)/cos(phi)) + tan(theta)*sin(phi)
    cam_to_road = (tan(xi_left) - tan(theta)*sin(phi)) *
        cos(theta)/cos(phi) * cam_height;
    cam_to_lane = (tan(xi_right) - tan(theta)*sin(phi)) *
        cos(theta)/cos(phi) * cam_height;
    lane_width = cam_to_lane - cam_to_road;
}

// find transforms between the camera coordinate frame
// and the world coordinate frame based on angles
// found from the vanishing point
void roadlines::findWorldToCamTransforms()
{
    /* rotations between world and camera coordinates
    double world_to_cam_rotation[3][3] = {
        {cos(theta),            0,        sin(theta)},
        {-sin(theta)*sin(phi),  cos(phi), cos(theta)*sin(phi)},
        {-sin(theta)*cos(phi), -sin(phi), cos(theta)*cos(phi)}};
    double cam_to_world_rotation[3][3] = {
        {cos(theta),           0,        -sin(theta)},
        {-sin(theta)*sin(phi), cos(phi), -cos(theta)*sin(phi)},
        {sin(theta)*cos(phi),  sin(phi), cos(theta)*cos(phi)}};
    */
```

```
    world_to_cam_rotation[0][0] = cos(theta);
    world_to_cam_rotation[0][1] = 0;
    world_to_cam_rotation[0][2] = sin(theta);
    world_to_cam_rotation[1][0] = -sin(theta)*sin(phi);
    world_to_cam_rotation[1][1] = cos(phi);
    world_to_cam_rotation[1][2] = cos(theta)*sin(phi);
    world_to_cam_rotation[2][0] = -sin(theta)*cos(phi);
    world_to_cam_rotation[2][1] = -sin(phi);
    world_to_cam_rotation[2][2] = cos(theta)*cos(phi);

    cam_to_world_rotation[0][0] = cos(theta);
    cam_to_world_rotation[0][1] = 0;
    cam_to_world_rotation[0][2] = -sin(theta);
    cam_to_world_rotation[1][0] = -sin(theta)*sin(phi);
    cam_to_world_rotation[1][1] = cos(phi);
    cam_to_world_rotation[1][2] = -cos(theta)*sin(phi);
    cam_to_world_rotation[2][0] = sin(theta)*cos(phi);
    cam_to_world_rotation[2][1] = sin(phi);
    cam_to_world_rotation[2][2] = cos(theta)*cos(phi);
}

// convert a point in the laser coordinate frame (x_l, y_l)
// to the two galvo command angles theta1, theta2
void roadlines::convertLaserToCommandAngles(
    double x_l, double y_l, double &theta1, double &theta2)
{
    //-2 * theta1 = atan(x/sqrt(f^2 + y^2))
    // 2 * theta2 = atan(y/f)
    theta1 = -atan(x_l/sqrt(focal_length * focal_length + y_l * y_l))/2;
    theta2 = atan(y_l/focal_length)/2;
```

```
}

// find transformation between image frame and
// laser frame based on 4 correspondences.
// Called at the end of calibration_state 2
void roadlines::findCalibration()
{
    if (num_corrs >= 4) {
        // Each correspondence gives us 2 linear equations:
        // m11*x_l + m12*y_l + m13 - m31*x_i*x_l -
        // m32*x_i*y_l - m33*x_i = 0
        // m21*x_l + m22*y_l + m23 - m31*y_i*x_l -
        // m32*y_i*y_l - m33*y_i = 0
        // These are the rows:
        // (x_l, y_l, 1, 0, 0, 0, -x_i*x_l, -x_i*y_l) lhs and
        // x_i rhs because m33 = 1
        // (0, 0, 0, x_l, y_l, 1, -y_i*x_l, -y_i*y_l) lhs and
        // y_i rhs because m33 = 1
        double** lhs = (double**)malloc(
            2 * num_corrs * sizeof(double*));
        for (int i = 0; i < 2 * num_corrs; i++) {
            lhs[i] = (double*)malloc(8 * sizeof(double));
        }
        double* rhs = (double*)malloc(
            2 * num_corrs * sizeof(double));
        for (int j = 0; j < num_corrs; j++) {
            lhs[2*j][0] = corrs[j][0];
            lhs[2*j][1] = corrs[j][1];
            lhs[2*j][2] = 1;
            lhs[2*j][3] = 0;
```

```
            lhs[2*j][4] = 0;
            lhs[2*j][5] = 0;
            lhs[2*j][6] = -corrs[j][2] * corrs[j][0];
            lhs[2*j][7] = -corrs[j][2] * corrs[j][1];
            rhs[2*j] = corrs[j][2];

            lhs[2*j+1][0] = 0;
            lhs[2*j+1][1] = 0;
            lhs[2*j+1][2] = 0;
            lhs[2*j+1][3] = corrs[j][0];
            lhs[2*j+1][4] = corrs[j][1];
            lhs[2*j+1][5] = 1;
            lhs[2*j+1][6] = -corrs[j][3] * corrs[j][0];
            lhs[2*j+1][7] = -corrs[j][3] * corrs[j][1];
            rhs[2*j+1] = corrs[j][3];
        }

gauss_solve(lhs, 8, rhs);
// solution is now in rhs
laser_to_image_transformation[0][0] = rhs[0]; //m11
laser_to_image_transformation[0][1] = rhs[1]; //m12
laser_to_image_transformation[0][2] = rhs[2]; //m13
laser_to_image_transformation[1][0] = rhs[3]; //m21
laser_to_image_transformation[1][1] = rhs[4]; //m22
laser_to_image_transformation[1][2] = rhs[5]; //m23
laser_to_image_transformation[2][0] = rhs[6]; //m31
laser_to_image_transformation[2][1] = rhs[7]; //m32
laser_to_image_transformation[2][2] = 1;      //m33
// image_to_laser_transformation is just the inverse of
// laser_to_image_transformation
```

```
matrix_inverse(
    laser_to_image_transformation, image_to_laser_transformation);
free(lhs);
free(rhs);
    }
}
```

Bibliography

[1] Kenneth R. Agent and Jerry G. Pigman. Accidents involving vehicles parked on shoulders of limited access highways. *Transportation Research Record, 1270*, pages 3–11, 1990.

[2] A. Basharat, A. Gritai, and M. Shah. Learning object motion patterns for anomaly detection and improved object detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[3] Michael T. Charles, John Crank, and David N. Falcone. *A Search for Evidence of the Fascination Phenomenon in Roadside Accidents*. AAA Foundation for Traffic Safety, 1990.

[4] R. T. Collins. Mean-shift blob tracking through scale space. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages II–234–40. IEEE Comput. Soc, 2003.

[5] Ingemar J. Cox and Sunita L. Hingorani. An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(2):138–150, August 2002.

[6] Richard Hartley and Sing Bing Kang. Parameter-free radial distortion correction with center of distortion estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29:1309–1321, August 2007.

[7] Berthold K. P. Horn. *Robot Vision*. MIT Press, 1986.

[8] Berthold K. P. Horn. Recovering baseline and orientation from 'essential' matrix. `http://people.csail.mit.edu/bkph/articles/Essential.pdf`, 1990.

[9] Berthold K. P. Horn. Relative orientation. *International Journal of Computer Vision*, 4(1):59–78, January 1990.

[10] Berthold K. P. Horn. Relative orientation revisited. *Journal of the Optical Society of America, A*, 8:1630–1638, October 1991.

[11] Berthold K. P. Horn. Projective geometry considered harmful (exterior orientation). `http://people.csail.mit.edu/bkph/articles/Harmful.pdf`, 1999.

[12] Berthold K. P. Horn. What is wrong with so-called 'linear' photogrammetric methods? `http://people.csail.mit.edu/bkph/articles/Orientation_2D_Illustration.pdf`, 1999.

[13] Albert S. Huang, David Moore, Matthew Antone, Edwin Olson, and Seth Teller. Multi-sensor lane finding in urban road networks. In *Proceedings of Robotics: Science and Systems*, Zürich, Switzerland, June 2008.

[14] Albert S. Huang and Seth Teller. Lane boundary and curb estimation with lateral uncertainties. In *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, St. Louis, Missouri, Oct. 2009.

[15] Lightweight communication and marshalling. `http://code.google.com/p/lcm/`.

[16] Mobileye. EyeQ. http://www.mobileye.com.

[17] Stephen S. Solomon and George L. Ellis. *Emergency vehicle accidents: Prevention and reconstruction.* Lawyers and Judges Press, Tucson, AZ, 1999.

[18] Chris Stauffer and W. Eric L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22:747–757, August 2000.

[19] Chris Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:2246, 1999.

[20] Thorlabs. *GVS011 and GVS012 Mirror Diameter Scanning Galvo Systems User Guide*, 7th edition, July 2011.

[21] U.S. Department of Justice, Federal Bureau of Investigation, Criminal Justice Information Services Division. Law Enforcement Officers Killed and Assaulted (Table 61: Law Enforcement Officers Accidentally Killed; Circumstance at Scene of Incident, 2000-2009). `http://www2.fbi.gov/ucr/killed/2009/data/table_61.html`, 2009.

[22] Shaohua K. Zhou, R. Chellappa, and B. Moghaddam. Visual tracking and recognition using appearance-adaptive models in particle filters. *Image Processing, IEEE Transactions on*, 13(11):1491–1506, 2004.

[23] Yue Zhou, Shuicheng Yan, and Thomas S. Huang. Detecting anomaly in videos from trajectory similarity analysis. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1087–1090. IEEE, July 2007.