Title

Development and Testing of Field-Deployable Real-Time Laser-Based Non-Intrusive Detection System for Measurement of True Travel Time on the Highway

Authors

Cheng, Harry H.
Shaw, Ben
Palen, Joe
et al.

Publication Date

2001-03-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Development and Testing of Field-Deployable Real-Time Laser-Based Non-Intrusive Detection System for Measurement of True Travel Time on the Highway

Harry H. Cheng, Ben Shaw, Joe Palen, Xudong Hu, Bin Lin, Jonathan E. Larson, Kirk Van Katwyk
University of California, Davis

California PATH Research Report
UCB-ITS-PRR-2001-6

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

Report for MOU 351

March 2001

ISSN 1055-1425

Development and Testing of Field-Deployable Real-Time Laser-Based Non-Intrusive Detection System for Measurement of True Travel Time on the Highway [1]

Harry H. Cheng
Ben Shaw
Joe Palen
Xudong Hu
Bin Lin
Jonathan E. Larson
Kirk Van Katwyk
Integration Engineering Laboratory
Department of Mechanical and Aeronautical Engineering
University of California, Davis
Davis, CA 95616

June 30, 2000

[1]The project is funded by the Caltrans through the California PATH Program

Contents

Abstract

We have developed a field prototype of a laser-based non-intrusive detection system for real-time measurement of delineations of moving vehicles for highway testing, based on our previous research on the laboratory prototype of the system. As described in the last report, the primary goal of this project is to develop a roadway detection system that can be used to gather reliable travel time data non-intrusively. The system uses a laser line that is projected onto the ground as a probe. The reflected light is collected and focused into a photodiode array by an optical system. Vehicle presence is detected based on the absence of reflected laser light. By placing two identical laser/sensor pairs at a known distance apart, the speed of both the front and rear of a vehicle is measured based on the times when each sensor is triggered. The length of each vehicle is determined by using these speed measurements and the residence time of the vehicle under each sensor. We have built a field prototype detection system which is used not only to proof the concept and algorithm of the detection system, but also to test the system in the real trafic environment. Several tests have been done with the field prototype system on the highway, and the test results further verified that the principle of our detection system is technically sound and indicated that the algorithm implemented in the software works in most cases. The software for the real-time data acquisition, data processing and graphic user interface has been developed in a real-time operating system. In the software, the speed, acceleration, and length of a detected vehicle can be calculated and displayed simultaneously. This document describes the design and implementation of each functional component and software of the field prototype system. The measurement and calculation of laser power has been performed to ensure that the system is safe to expose to public during the field tests and future operation on the highway.

Chapter 1

Introduction

Travel time is the most important aspect of the Intelligent Transportation System (ITS). Travel time is a good indicator of other direct constraints on ITS efficiency: cost, risk, and attentive workload. The importance of travel time is verified in ATIS user surveys which indicate that what travelers most want from a transportation system is (almost always) reduced travel time and higher reliability (e.g. reduced travel time variance and reduced risk) [1]. Every traveler must implicitly or explicitly make an assessment of these various travel time options before embarking on every trip, therefore this information is definitely of high value. Because trip travel time is the parameter the public most wants to minimize, this is the parameter that is most important for transportation service providers to measure and minimize.

Speed is commonly used as an indicator of the travel time across a link. In current practice, speed is measured at one or more points along a link and extrapolated across the rest of the link [1]. This extrapolation method is used regardless of the mechanism of detection. Example detection methods are loops—which determine speed from two elements twenty feet apart; radar—which can directly determine speed from the carrier frequency shift (Doppler effect); or video image processing—which tracks vehicles across the pixel elements within the field of view. The extrapolation from a point to a line is not necessarily valid. At the onset of flow breakdown, the speed variations along the length of a link can be quite large. Also, the onset of flow breakdown is when routing decisions are most time-critical and accurate information has the highest value, so inaccurate extrapolations could have detrimental effects to the traveler.

An alternate method to determine the traverse travel time (e.g. the true link speed) is to use Vehicles As Probes (VAP). A VAP system determines travel time directly by identifying vehicles at the start of the link and re- identifying them at the end of the link, with the time difference being the true travel time. The problem with VAP systems is that they require large numbers of both vehicle tags and tag readers to be effective, and the cost justification of such a system seems unwarranted in the light of other options. The key aspect to measuring the actual travel time is simply to identify some distinguishing characteristic on a vehicle at the beginning of a link and then to re-identify that same characteristic on the same vehicle at the end on the link. This is the basic idea of VAP, however the characteristic does not have to be entirely unique (as in a vehicle tag), and it does not necessitate the infrastructure set-up costs of VAP. If a characteristic can be found to separate the fleet into (say) 100 classifications, "the maximum probability fit" can be determined for the same sequence of classifications at the downstream detector as was identified at the upstream detector. This is what is currently being done in Germany with the low resolution imaging provided by (new high speed) loops [1]. If a higher-resolution detector is used so that it is possible to get a few thousand classes, then it should be quite possible to perform a 100% upstream-downstream Origin and Destination (O/D) (even if a significant percentage of the vehicles switch lanes) using time gating and other relatively straight-forward signal processing techniques. The mechanism of detection must allow highly resolved delineation between commonly available "commuter" vehicles, because commuter vehicles represent the majority of the vehicle stream during the period that traverse travel time information is most needed (e.g. the peak hours).

Any mechanism to measure travel time, by definition, is only determining the "past state" of the transportation system. Collecting data on what happened in the past has no utility except if it is used to infer what may happen in the future. All decisions, by definition, are based on an inference of future consequences. When a traveler learns that speed on a route is 50 MPH, the traveler generally infers that the speed will remain 50 MPH when she/he traverses it. This may or may not be a reasonable inference. Travelers want to know the "state" of the system (in the future) when they traverse it. In the simplest case, this is just a straight extrapolation of current "state." More sophisticated travelers may develop their own internal conceptual model of the typical build up and progression of congestion along routes with which they are familiar. A major benefit of ITS will be to provide travelers with a much more valid and comprehensive "look ahead" model of the (short term) future state of the transportation system. Validation of any traffic model requires (either implicitly or explicitly) traffic O/D data. The lack of valid O/D data has been the major impediment in the calibration, validation and usage of traffic models. In this research project we are developing a roadway detection system that can directly determine O/D data non-intrusively without violating the public's privacy (as in license plate recognition systems).

1.1 Comparison of Different Detection Schemes

Our detection system has a number of advantages over other systems currently in use. In current practice, vehicle features are most commonly measured using inductive loops or video image processing. An advantage of our system over loop detectors is the relative ease of installation and maintenance. Because loops are buried beneath the pavement, installation requires heavy equipment, and traffic must be re-routed [2]. It is for this reason that loops are expensive to install and repair. Because our system is mounted above the road, once installed, it can be maintained without disrupting the flow of traffic. More importantly, loop detectors cannot be relied upon to produce accurate speed (and therefore length) measurements because the inductive properties of the loop and loop detectors vary [2]. Video can be used to directly measure the length of vehicles, however the use of real time video image processing is problematic due to its computationally intensive nature. Our system operates on a simple "on/off" basis, requiring much less computation for vehicle detection, and consequently much less computational hardware. Because video is a passive system (gathering ambient light), video images are dependent on the lighting conditions. Vehicle length measurements taken from video, even on the same vehicle, may not produce consistent results depending on time of day and weather conditions. For truly site and time independent vehicle length measurements, video would require an external source of illumination. Because our system is active, it produces its own signals to be sensed and it does not suffer from these limitations.

One system that bears some similarity to the system we have developed is the Automatic Vehicle Dimension Measurement System (AVDMS) developed by the University of Victoria [3]. However, the AVDMS would not be suitable for our purposes. The AVDMS uses laser time-of-flight data to classify vehicles based on length, width, or height, and is based on the Schwartz Electro-Optics Autosense III sensor [4] [5][6][7]. The Schwartz systems are entirely dependent on time-of-flight laser measurements with moving parts, similar to conventional lidar (laser radar) in the principle of measurement. There are some significant functional differences between our system and Schwartz's. For example, the fundamental mechanism of detection is that the Schwartz detector determines the range (or distance) from the detector to the objects being detected. Our detector functionally does not determine the range (or distance) from the detector to the objects being detected. The laser of the Schwartz's detector reflects off the vehicle to determine the size, shape, and "presence" of the vehicle. In our detector, the laser reflects off the pavement. The lack of a reflection determines the size, shape, and "presence" of the vehicle. Therefore, our system will be more reliable because of its simplicity.

In comparison with other conventional traffic detection techniques, our system will offer the following

salient features:

- The system is mounted above the road and is relatively easy to install. Traffic need not be rerouted.
- The system is insensitive to ambient lighting conditions due to the active lighting source (the laser). It detects every passing object more than 46 cm (18 in) tall in all lighting conditions. No vehicles are missed, yielding nearly 100 % accuracy.
- The laser and detector have no moving parts, giving the system high reliability. The primary raw data gathered by the sensor are computationally easy to process.
- Not only does the detector produce local vehicle speed, vehicle volume, and vehicle classifications, but it also allows highly deterministic re-identification of vehicles between sites, even under high flow conditions. Point-to-point travel time, incident detection, and Origin/Destination data can easily be determined with this detector.
- The system has very low power and communication bandwidth requirements, allowing the development of a stand-alone detector untethered from hard-wired infrastructure.

1.2 Previous Works

A laboratory prototype was developed to verify the principle the detection system in the previous project. The detection system hardware was constructed using off-the-shelf parts for rapid assembly and easy modification. The laser chosen for use in the indoor prototype system was a Melles Griot model 56 DIL 452/P1 laser line projector. This product has a 30 mW red laser diode and all of the necessary optics for line projection built into a single package. In the laboratory prototype system, the current output from the sensor is amplified and conditioned for interfacing with a Delta Tau PMAC digital I/O board. The data are then collected in real-time by a computer for processing. Sensor software performs low-level gathering of the detector data and an application programmers interface (API) provides a standard way of accessing the data. Applications use this sensor API to display the data in various ways. The X window program shows the time history of the sensor data, displaying multiple detector values at regular intervals and profiles of vehicles as they pass under the sensor. The text based interface displays raw data. The laboratory prototype was tested in the laboratory and on the roof of a building in a simulation environment. The experimental results from laboratory system showed that the principle of the detection was feasible.

1.3 System View of Field Prototype

In the field prototype system, vehicle length is used as the primary identifying feature and is measured using two laser-based vehicle detectors. The system operates in the following manner, as illustrated in Figure 1.1. The basic detector unit consists of a laser and a spatially offset photodetector positioned above the plane of detection. The laser is a pulsed infrared diode laser that utilizes line-generating optics which project to a flat planar surface where objects are to be detected. The detector consists of imaging optics and a linear photodiode array. The offset photodiode array receives the laser light that is reflected back from the region of detection. The signal from the photodiode is amplified and sent to a computer for processing. Vehicle presence is detected based on the absence of reflected laser light. Two of these units are integrated and placed a known distance apart, allowing the velocity of the object and its residence time under each detector to be measured, giving the object's length and top-down outline profile.

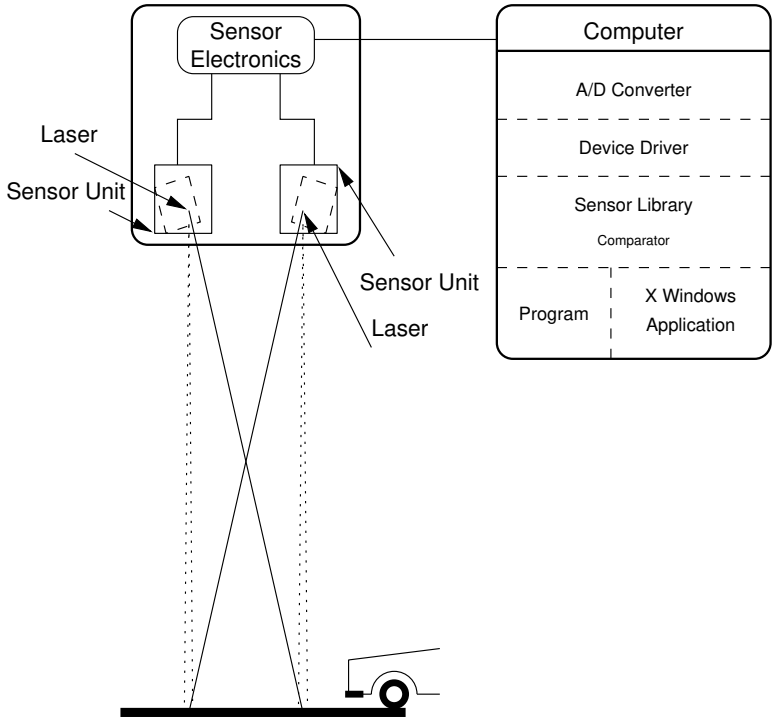


Figure 1.1: System Overview

Figure 1.2 shows the positioning of the system hardware. The detector is mounted at a distance of about 6.4 m (21 ft) (the height of a typical highway overpass) above the highway. The distance between each component of a laser/sensor pair is 30.5 cm (1.00 ft). The offset between the two sensor pairs is 10 cm (4 in). The sensors are mounted in a fixed vertical position, pointing downward, and are focused on the ground, forming two detection zones. The lasers are pointed towards the detection zones and are mounted at an adjustable angle, allowing the system to be mounted at different heights. The detection zones stretch across the width of the lane and are each about 13 mm (0.5 in) wide in the direction of traffic flow. In this configuration the minimum detectable object height, also called the critical height, is about 46 cm (18 in). This is lower than the bumper height of most common vehicles. For objects below this height, the laser line will still be visible by the sensor. This can result in the object remaining undetected or can cause signal spike due to reflections, depending on the surface properties and geometry or the object. In either case, for vehicle bumper heights below the critical height, the speed and length measurements will be incorrect due to the fact that one or more of the vehicle edges will be incorrectly found.

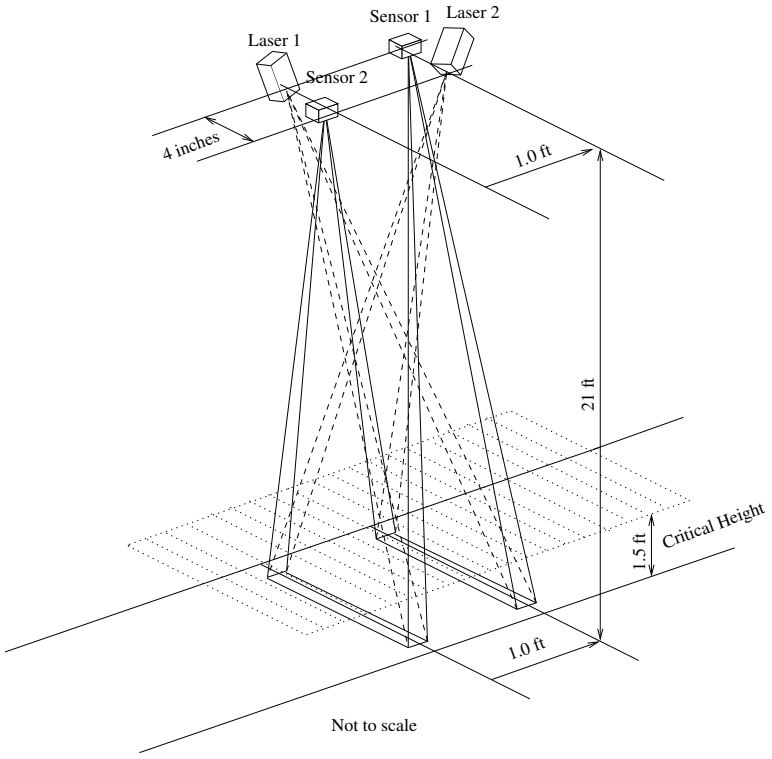


Figure 1.2: System Hardware Configuration

When a vehicle moves into a detection zone, it blocks the laser from being received by the sensor, as shown in Figure 1.3. When the first beam is blocked the current time is recorded. When the second beam is blocked, a second time is recorded. These times give the speed of the front of the car. In a similar manner, when each of the beams is no longer blocked, as shown in Figure 1.4, the times are recorded and the speed of the rear of the vehicle can be calculated. The time that each detector is blocked is also recorded and is used to calculate the vehicle length, assuming constant vehicle acceleration. A more detailed description of the speed and length measurement algorithms is presented in the software section. The assumption of constant acceleration is valid for free-flow traffic conditions, where there is negligible acceleration, and for conditions where the vehicle is accelerating or decelerating uniformly during the time it is in the detection zone. These cover the majority of situations, but there are a few situations, such as stop-and-go traffic, where this basic detection method will not work well.

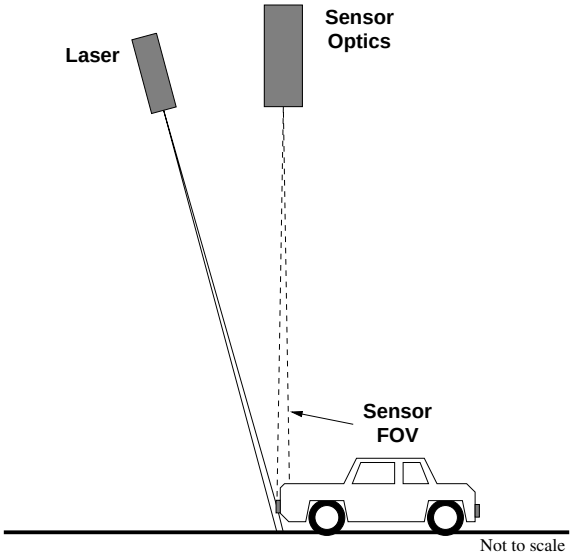


Figure 1.3: Vehicle Entering a Detection Zone

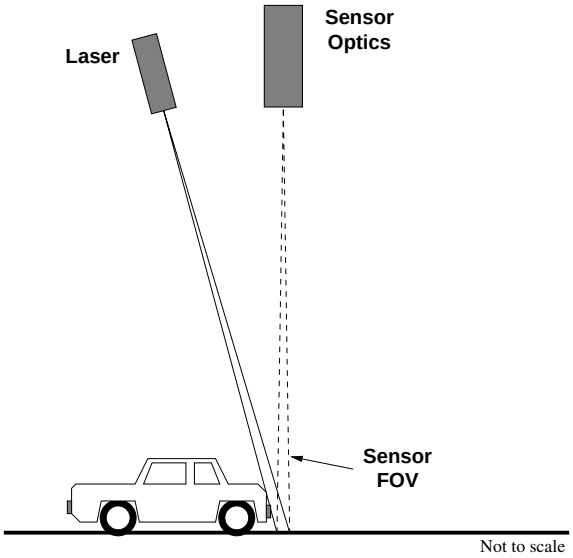


Figure 1.4: Vehicle Exiting a Detection Zone

Chapter 2

Field Prototype system

2.1 Laser and Optics

2.1.1 Laser

Two off-the-shelf integrated diode laser systems, ML20A15-L2 high power diode laser systems from Power Technology Inc are used as the laser source in the current prototype. This is an integrated laser system that incorporates a DC/DC voltage converter, voltage regulator, pulse generator, laser diode and line generation optics into a single unit. Operation requires only a +9V - +14.5V DC power supply and a trigger pulse. The system has a peak power output of 20W at 905 nm, with a pulse width of 15ns. It can be pulsed at a maximum rate up to 10 KHz. The line generating optics produce a beam with a full fan angle of 15 degrees. Its high performance and small size make it a good candidate for use in the field deployable prototype system.

Figure 2.1: ML20A15-L2 Integrated Laser System

A wavelength of 905 nm for the laser was chosen for a number of reasons. Infrared light has good transmittance through fog, giving the system better performance under a larger range of weather conditions. Furthermore, the intensity of sunlight around the wavelength of the laser is a local minimum, giving the system better rejection of noise due to sunlight. An infrared laser was also thought to be more appropriate for outdoor use because it is invisible to the human eye, and would therefore cause no distraction to passing motorists.

In the field-deployable prototype it is necessary that the laser be eye-safe. The ML20A15-L2 has a pulse width of 15 ns, a maximum output power of 20 W and a maximum frequency of 10 kHz. With the assumptions

that the laser is located 6 m above the roadway, the laser line is 4 m long and 5 mm wide, and that a potential observer is 2 m above the roadway as shown in Figure 2.2, the safety of the laser is verified in this section.

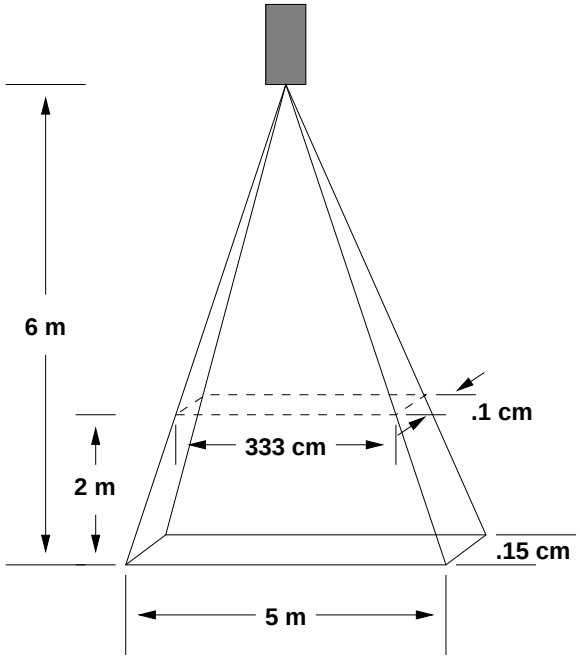


Figure 2.2: Projection Area of the Laser

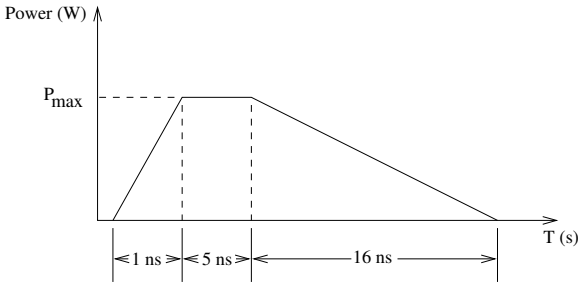


Figure 2.3: Laser Pulse Waveform

Assuming the laser pulse has the form shown in Figure 2.3 and the area of exposure is 267 cm x 0.33 cm, the radiant exposure per pulse of the laser system can be calculated by:

$$\begin{aligned} \frac{H_{exposure}}{pulse} &= \frac{(\frac{1}{2}\cdot 5ns + 15ns + \frac{1}{2}\cdot 16ns)\cdot 20W}{L\cdot W} \\ &= \frac{4.7\times 10^{-7}J}{267cm\cdot 0.33cm} \\ &= 5.33\times 10^{-9}\frac{J}{cm^2} \end{aligned} \tag{2.1}$$

For the pulse width of $32\times 10^{-9}s$, the single pulse maximum permissible exposure (MPE) for a 905 nm laser expressed as radiant exposure (H) is given by [10]:

$$MPE : H = 5 \cdot C_A \times 10^{-7} \frac{J}{cm^2} \tag{2.2}$$

with $C_A = 2.56$ for our laser, this becomes

$$H = 1.28 \times 10^{-6} \frac{J}{cm^2} \tag{2.3}$$

The MPE per pulse for repetitively-pulsed intra-beam viewing is $n^{-1/4}$ times the MPE for a single pulse exposure where n is the number of pulses found from the viewing time ($T_{viewing}$) and the pulse frequency (f_{pulse}). Assuming a pulse frequency of 10 kHz, the maximum safe viewing time can be found by equating the MPE per pulse and the radiant exposure per pulse.

$$\begin{aligned}
\frac{MPE}{Pulse} : H &= \frac{H_{exposure}}{pulse} \\
n^{-1/4} \cdot 1.28 \times 10^{-6} \frac{J}{cm^2} &= 5.33 \times 10^{-9} \frac{J}{cm^2} \\
n^{-1/4} &= 4.17 \times 10^{-3} \\
n^{1/4} &= 240 \\
n &= 3.32 \times 10^{9} \\
T_{viewing} \cdot f_{pulse} &= 3.32 \times 10^{9} \\
T_{viewing} \cdot 10000 \frac{1}{s} &= 3.32 \times 10^{9} \\
T_{viewing} &= 3.32 \times 10^{5} s \approx 92h.
\end{aligned} \tag{2.4}$$

The laser is safe for viewing times up to 92 hours. Although it is unlikely that a driver would stop on the highway and stare into the beam for this long, warning signs will be posted to prevent it.

In the June, 1999, we measured the laser power of our prototype detection system at both UC Berkeley and UC Davis, and performed extensive calculations using computer software for laser safety with the laser safety officer at UC Davis. The calculation results indicate that the laser power of our detection system is below the minimum national laser safety standard by a large margin.

2.1.2 Sensor Optics

The sensor optics consists of three main components: an imaging lens system, a telescopic lens system, and a filter.

The imaging lens system focuses the reflected laser light onto the active area of the sensor array. The imaging lens was selected based on the criteria that it should have an adjustable focal length within a range around the desired focal length, that it should have a field-of-view large enough to capture the width of an entire lane and that it should be compact for easy integration into the outdoor system.

Based on the assumptions that the lane width (h_o) is around 3.05 m (10.0 ft) and that the unit will be mounted about 6.40 m (21.0 ft) above the roadway (S_o) and given that the sensor is 7.5 mm (0.295 in) long (h_l), According to the Figure 2.4, an image distance (S_i) was calculated for the sensor using Equation 2.5, where it was determined that s= 15.8 mm (0.620 in).

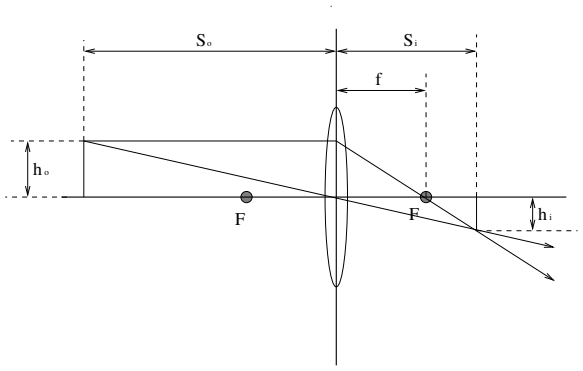


Figure 2.4: Lens Parameters

$$S_i = S_0 \frac{h_l}{h_o} \tag{2.5}$$

The desired focal length (f) of the lens was then calculated using Equation 2.6.

$$f = \frac{1}{\frac{1}{s_i} + \frac{1}{s_0}} \tag{2.6}$$

The focal length was calculated to be 15.7 mm (0.616 in). As a practical matter, the sensor array is placed at the focal point of the imaging lens system. Because so is large in comparison with si, f is nearly equal to si.

The lens we selected, a Tamron 23VM816, has an adjustable focal length of between 0.315 in (8 mm) and 0.630 in (16 mm) and was selected because of this feature. The Tamron lens is also suitably compact and has a field of view that should be large enough to capture the entire lane width. Any lens system that has the correct focal length and an acceptable field-of-view could be used.

The telescopic lens system is mounted in front of the imaging lens system. It is designed to restrict the field-of-view of the imaging lens along the width of the laser line, but not alter the field-of-view along the length of the line. Because the laser line is much longer than it is wide, use of the imaging lens alone would result in a much wider strip of pavement being visible to the sensor than is desired. The telescopic lens system is used to match the dimensions of the laser line image with those of the sensor array. Figure 2.5 shows the imaging lens and the cylindrical lenses used in our system.

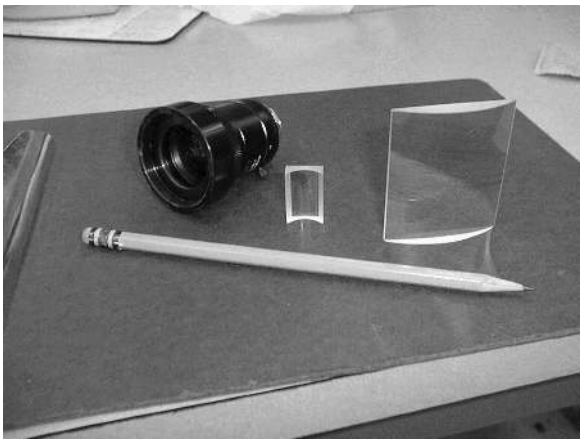


Figure 2.5: Imaging Lens and Cylindrical lenses

The telescopic lens system consists of one positive plano-cylindrical lens and one negative plano-cylindrical lens. The prototype uses a 150 mm focal length cylindrical lens and a -19 mm focal length cylindrical lens both manufactured by Melles Griot Inc. These lenses are positioned to form a Galilean telescope. When positioned correctly the cylindrical lenses will not effect the proper operation of the imaging lens. The ratio of the focal length of these lenses is approximately equal to the ratio of the uncorrected field-of-view of the width of the sensor to the desired field-of view. The desired field-of-view, X, is determined based on Equation 2.7, where Y is the separation of the sensor and laser, H is the height of the system above the road, and Hc is the desired minimum detectable object height, as shown in Figure 2.6 To insure reliable vehicle detection, it is important that Hc be below the bumper height of most common vehicles.

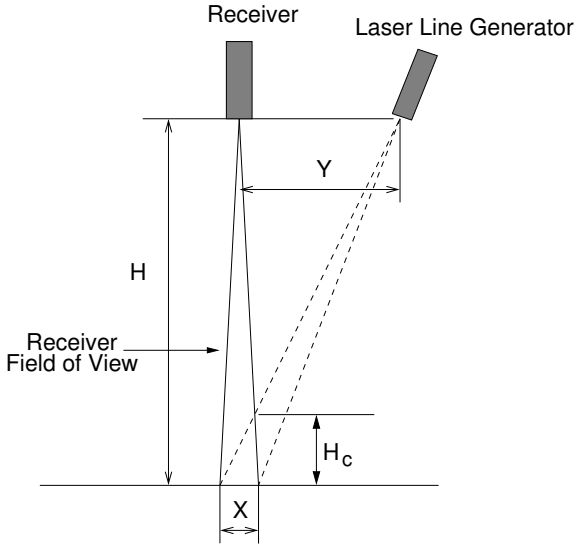


Figure 2.6: Minimal Object Height

$$Y = \frac{X(H - H_c)}{H_c} \tag{2.7}$$

The uncorrected field of view, about 13 cm (5 in), results in a critical height of about 1.8 m (6.0 ft). To ensure vehicle detection it is necessary to have a critical height somewhere below the bumper height of the

vehicles. A height of around 46 cm (18 in) was thought to be acceptable. To achieve this it is necessary to restrict the field of view X to about 2.3 cm (0.92 in). This is a factor of reduction of about 5. In our case, where f1 = 150 mm and f2 = -19 mm, the factor of reduction is equal to about -7.9 (the negative sign indicates an inverted image), giving us a field of view of about 16 mm (0.63 in). The factor of reduction is commonly referred to as the angular magnification of the system. As shown in Figure 2.7, a ray of light entering the system from the left at an angle θ_1 exits the system at the right at an angle θ_2 equal to $\theta_1 * (f1/f2)$. Because of this, objects to the left appear to be larger than they actually are. This is how the field of view is reduced. A sensor on the right of the telescopic system will have its field of view reduced by a factor equal to the angular magnification of the system. The telescopic system does not alter the position or focus of the image. Objects that are properly focused by the imaging lens remain in focus when the telescopic system is added.

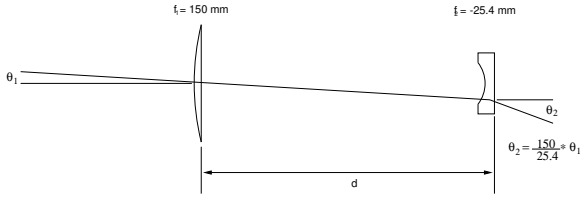


Figure 2.7: Telescopic Lens System

A bandpass filter that is matched with the wavelength of the laser is used to reduce the level of ambient light received by the sensor. The filter is mounted between the telescopic and imaging lens systems. The filter used in the prototype is manufactured by Omega Optical Inc. (model 904DF15). This filter is mounted on a ring that is threaded onto the front of the imaging lens.

2.1.3 System Electronics

A block diagram of the prototype hardware construct is shown in Figure 2.8. The hardware can be divided into five main parts: the power supply, the clock generator, the laser components, the sensor circuit, and the A/D converter and computer.

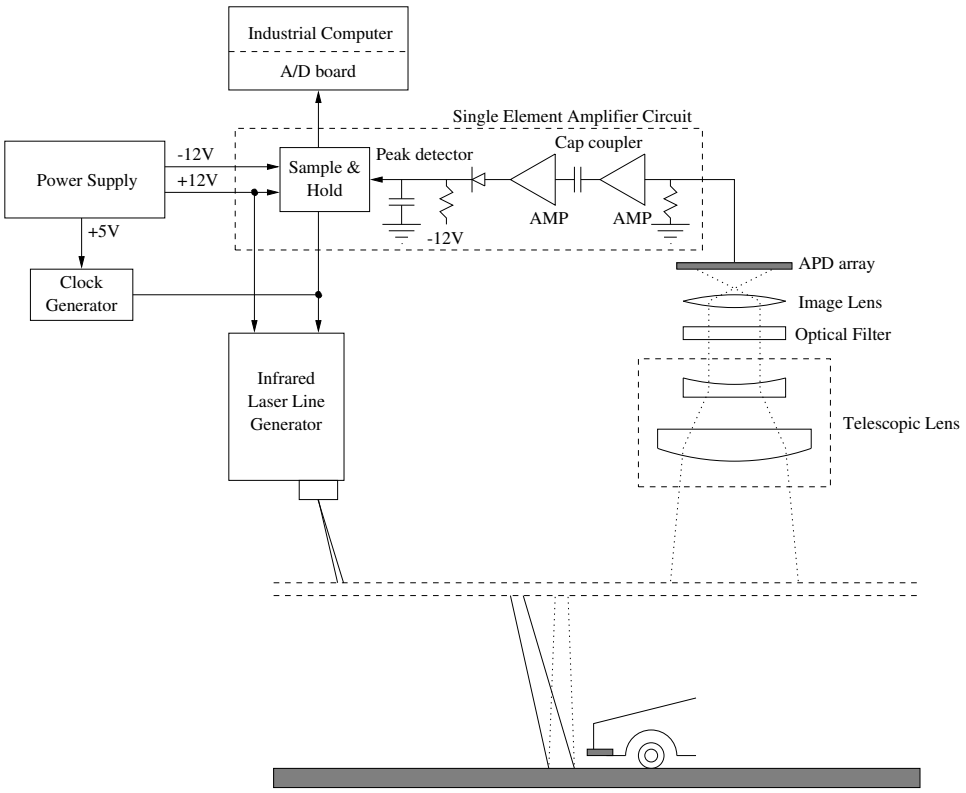


Figure 2.8: Diagram of Electronics Hardware

Power Supply Section

The power supply section delivers power to both the laser components and the sensor circuit. There are many different voltages needed by the system. A triple output power supply provides +12 V, -12 V and +5 V. The +5 V output is used to power the clock generator. The +12 V output supplies power to the laser system and the DC/DC converter required for the sensor array. A high voltage DC/DC converter changes 12 V DC to 250-350 V DC and is used to bias the sensor array to -290 V. The sensor circuit (except for the pre-amplifier) uses both the +12 V and -12 V outputs. For our triple-output DC power supply, the maximum output ripple is 5mV peak-to-peak value. This is a little large for a weak-signal amplifier power supply. The high voltage pulse generator for the laser diode is built in the laser unit which is powered by 12 volts DC. Because the pulse generator is isolated by a transformer and is well shielded, there is a very low noise induced by the pulse. In order to further reduce noise, a separate power supply for the pre-amplifier is used to increase the signal quality. In the system, a linear encapsulated power module, which produces +/- 5V, is used to power the pre-amplifier. The maximum output ripple of this power supply is 1mV peak-to-peak value.

Clock Pulse Generator Section

The clock generator provides a clock signal that is used to trigger the laser and to synchronize it with the sampling of S/H. In our system, an LM555 is used as the oscillator, as shown in Figure 2.9. A $4\mu s$ pulse-width, 2.2kHz clock signal was chosen to operate the laser system. The frequency and width of the pulse can be chosen by adjusting the values of resistors R_a and R_b. Increasing the value of R_a will increase the pulse frequency and increasing the value of R_b will increase the pulse width. In the near future, we will increase the clock frequency to 10 kHz.

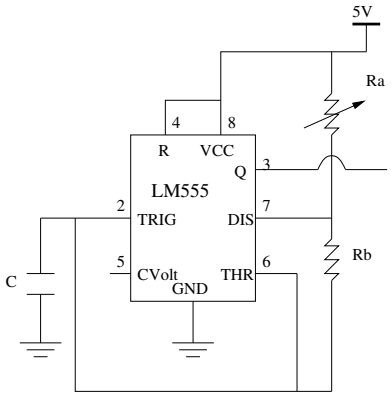


Figure 2.9: Clock Circuit

Sensor Circuit Section

A 25-element avalanche photodiode(APD) array is used as the sensor in our detection system, though currently only four elements of each array are used. In the future we plan to use all elements of the array. The sensor converts the reflected laser light into a current signal. The sensor circuit is the main part of the electronic hardware in the detection system. This circuit conditions and amplifies the signal produced by a single element of the sensor array so that it is suitable for sampling by the data acquisition board. Each element of the sensor that is used has its own circuit as described below.

The circuit can be divided into three stages: signal conditioning and amplification, peak detection, and sample/hold, as shown in Figure 2.10.

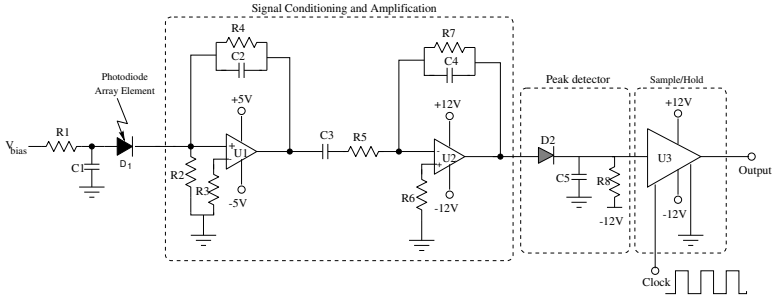


Figure 2.10: Sensor Circuit

The current produced by a sensor element is converted to a voltage by U1. U1 is a low-noise, high-speed amplifier and is used in a non-inverting configuration. C2 is a small value capacitor used for noise reduction. The signal is then passed through a resistor, R4, and capacitor, C7, which filters out the DC or low frequency signals components from the previous stage. U2 is used as an inverting amplifier. The signal is amplified to a suitable value for the computer to handle. Similar to the pre-amplifier stage, C9 is used to further reduce the noise. The amplified signal is a voltage pulse. The peak detector is needed to pick up the signal peak and deliver it to the sample & hold circuit. D2 and C14 consist of a peak detector. The output of the amplifier stage charges C14 through diode D2. The highest point of the output waveform of the amplifier is held by C14 while the diode D2 is back-biased. R7 and a -12V power supply are used to reset the capacitor C14. The detected peak is then input to the sample & hold circuit. This circuit uses the same clock pulse that is used as a trigger by the laser to synchronize the sampling of the signal with the laser pulse. C3, C4, C5, C6 and C10, C11, C12, C13 are de-coupling capacitors which filter noise from the power supply.

A/D Converter and Computer Section

The output from the sample & hold is an analog signal that must be digitized for processing. A PC-based, 16-channel A/D board installed in an industrial computer is used for this purpose. The converted digital data is then sent to the computer through the data bus for further handling. The main computer used in the system is an industrial Pentium computer. Custom software is used for processing of the data.

2.2 Mechanical Design

The test system is enclosed in a box mounted on a cantilevered structure (see Figure 2.11) that was designed for use during testing. The structure allows the system to be placed at various locations and is easy to assemble and modify for testing purposes. Figure 2.11 shows the system mounted on the cantilevered frame with box opened.

Figure 2.11: Testing prototype mounted on the cantilevered frame

The mechanical design of the test system is shown in Figure 2.12. Figure 2.13 is the picture of the prototype.

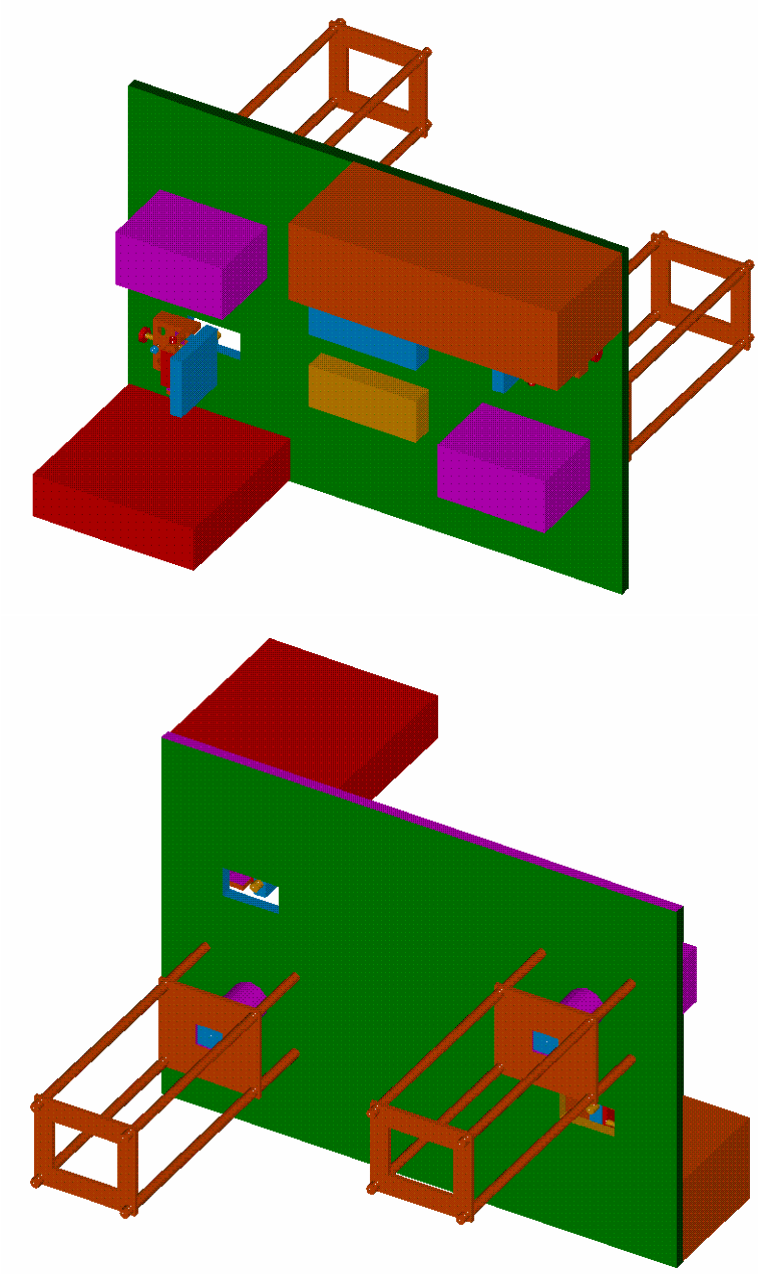


Figure 2.12: Testing prototype mechanical design

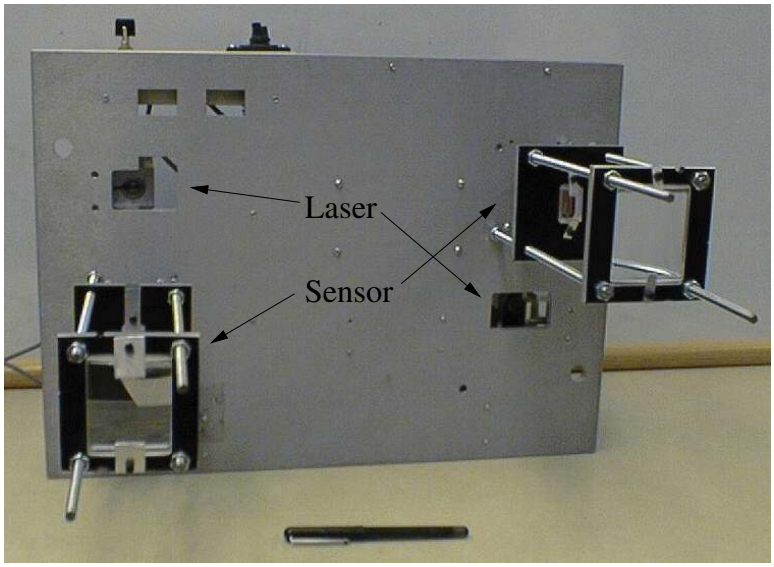


Figure 2.13: Picture of the testing prototype

To integrate the optical and electronic components, we have designed and manufactured several mechanical components. A laser stand was designed to hold the laser. It allows the angle of the laser to be adjusted. This swiveling mount (shown in Figure 2.14, Figure A.1, and Figure A.2) holds the laser with two extending arms while providing angular adjustments through the rotation of a threaded bolt.

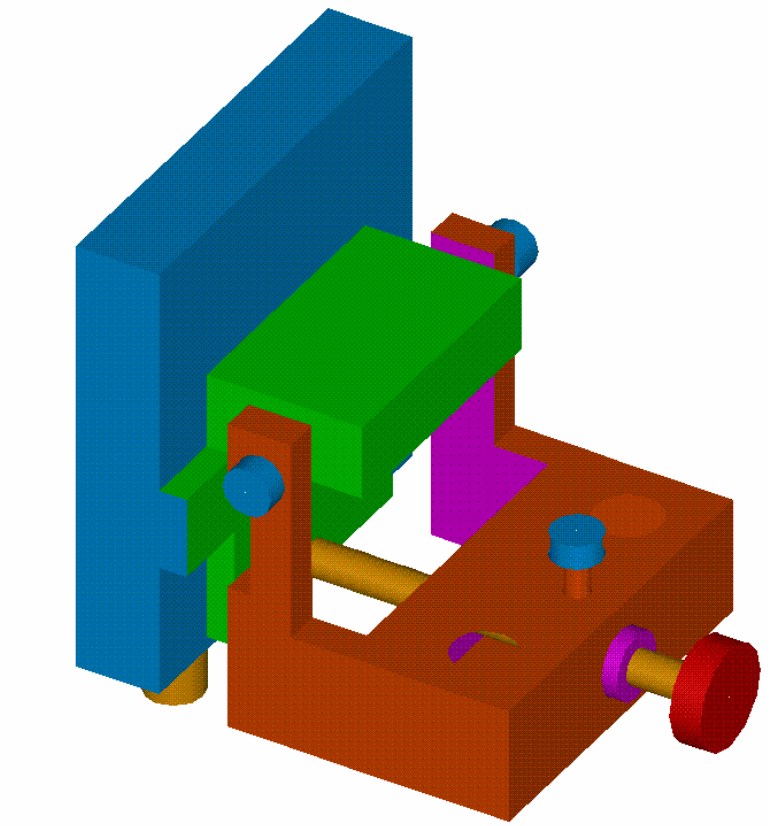


Figure 2.14: Laser stand

The receiving end of the detection system, consisting of a CCTV camera lens with two cylindrical lenses, is mounted directly on the bottom of the enclosure. The spacing of the cylindrical lenses can be adjusted. The photodiode array and circuitry are mounted above the CCTV lens inside the enclosure. The mechanical components draft are shown in Appendix A.

Chapter 3

Real-Time Data Aquisition and Processing Software

The purpose of the laser detector software was to collect, process and display vehicle delineation data, all in real time. The software is separated into layers as shown in Figure 3.1. Each layer performs a specific function and implementation is hidden from other layers. The layers interact and pass data by using function calls.

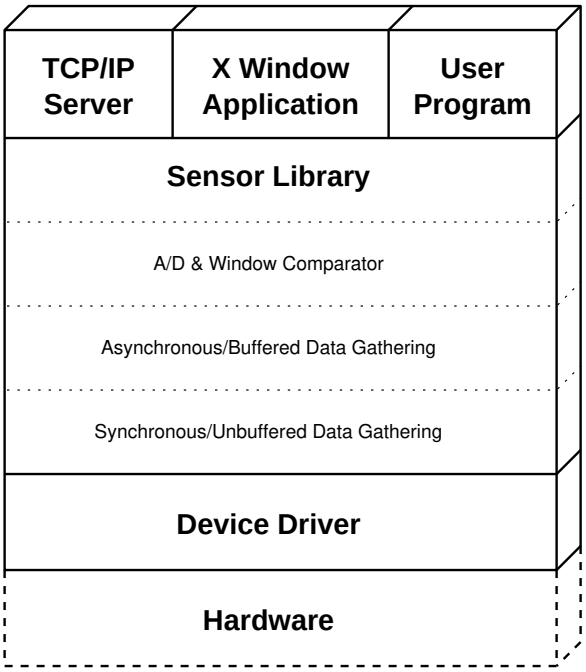


Figure 3.1: Component diagram of the Laser-Detector system software

The device driver is the first layer of software. It communicates with the computer hardware in a low-level fashion. The interface to the data acquisition device driver occurs through standard function calls, such as `open()`, `close()`, `read()`, `write()` and `ioctl()`. The device driver hides the low-level tasks necessary for communication between the hardware and the software. In this way, the rest of the sensor software does not have to be burdened with low-level communication. The device driver uses a circular queue to buffer the data while it is continuously collecting data from the data acquisition board. It does this so no data are lost between requests for data.

The sensor library layer requests detector data from the device driver and processes the data for applications and other software libraries. The sensor library is used by both the vehicle delineation library and the applications in the top layer of the software. A TCP/IP server is used to send the vehicle data over a network. X Windows System applications are used to display the detector and vehicle data in real-time. Finally, a user program is used to perform simple operations on the data.

3.1 Sensor Library

The sensor library is the largest portion of the software. It obtains detector data from the data acquisition device driver and processes the data for other applications and software libraries. The sensor library requests data from the device driver in a specified block size. The library converts the data from digitized values to values usable on other layers. The library consists of various levels of functionality. The library is multi-tasking and multi-threaded. It is multi-tasking because of the various tasks that must occur simultaneously. Multi-tasking is implemented by using threads that run at separate scheduling priorities. A higher priority thread blocks other threads from running. Using priority based threads increases the responsiveness of the software for real-time data acquisition.

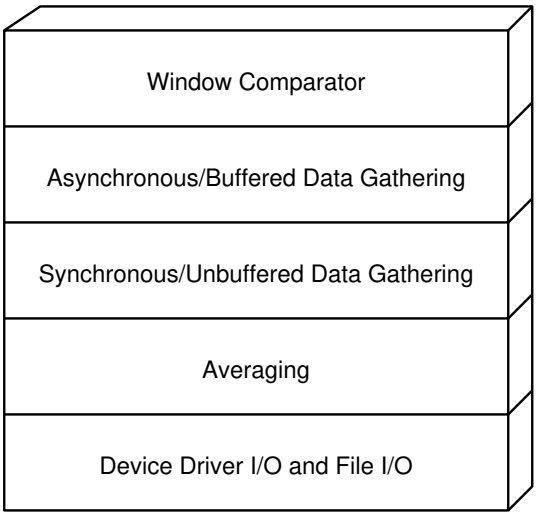


Figure 3.2: Component diagram for the sensor library

Figure 3.2 shows the separate layers of the sensor library. The bottom layer consists of the device driver function calls and the file input and output (I/O) function calls. The next layer computes a running average of the photodiode signal data. The Synchronous/Unbuffered Data Gathering layer sets up and collects data from the device driver. This layer contains a high priority thread that uses the averaging layer and the device driver and file I/O layer. The interaction between the high priority thread and the application is synchronized. No buffer exists between the thread and the application requesting the data.

The next layer is the Asynchronous/Buffered Data Gathering layer. This layer sets up and uses a buffer that allows asynchronous requests of data while the high priority thread reads the data and stores them in the buffer. This scheme is desirable because the application does not directly affect the gathering of data. As a result, no data are lost.

The top layer of the sensor library is the window comparator. The window comparator converts floating-point signal values to detection states, which indicate whether or not the laser is blocked. A blocked laser indicates the presence of a vehicle under a particular photodiode element. The comparator is configurable during run time to be responsive to changing signal conditions.

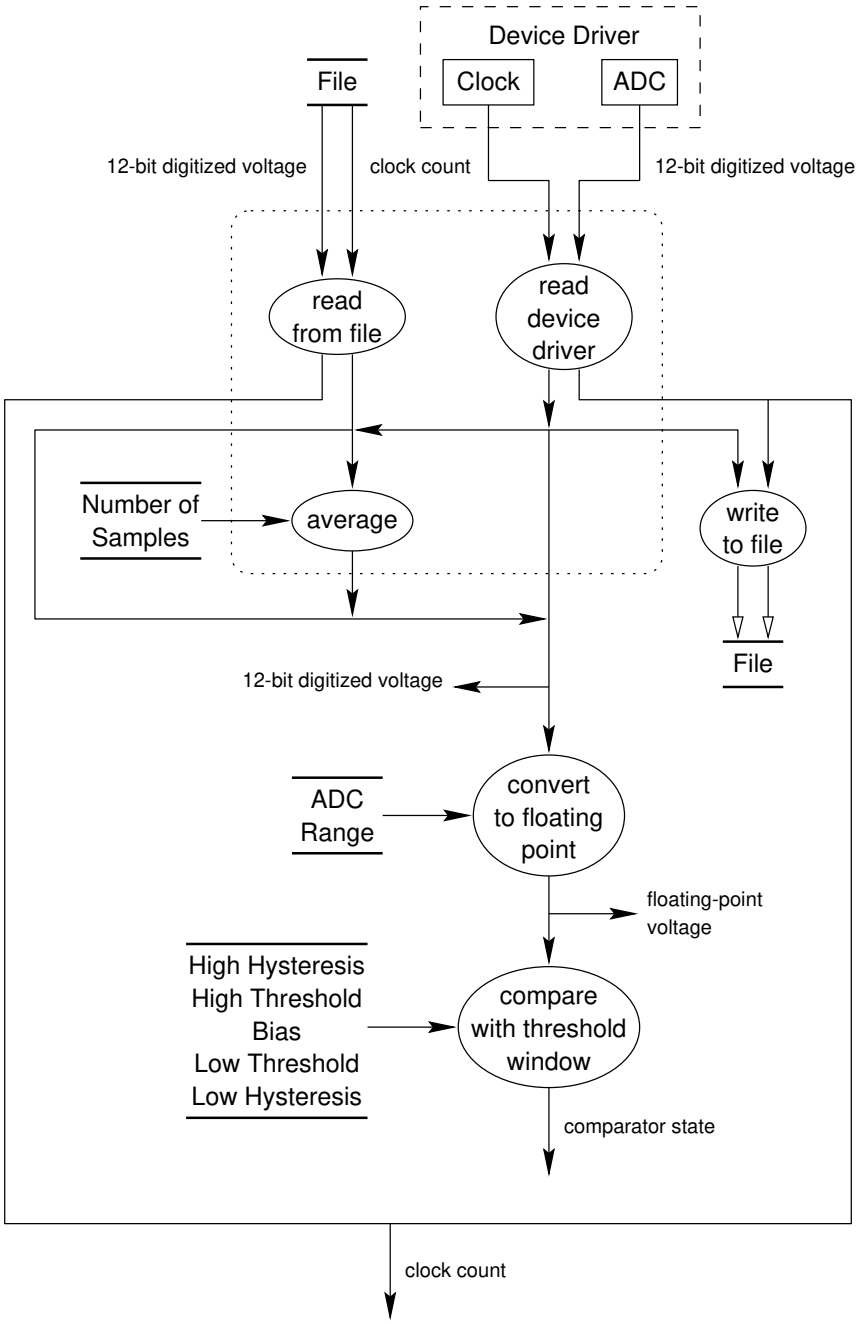


Figure 3.3: Sensor Library Functional Diagram

Figure 3.3 shows the data flow between the different functions of the sensor library. The sensor library requests clock data and ADC data from the device driver. Data from the ADC are in the form of 12 bit digitized voltage values. The clock count is a 32 bit number representing ticks of time. Each block of data contains a clock count so each sample of data can be linked directly to an instance in time. The sensor library has the option to save the clock counts and digitized values to a file for later playback or use the values directly from the device driver. The averaging function of the sensor library converts a specified number of samples to a single average voltage. This is done to eliminate random noise read from the ADC. A high priority thread reads the device driver or reads from a file and averages the data. A low priority thread saves data to a file

so the higher priority collection of data is not interrupted. All other functions are of the same priority as the application program using the sensor library.

Next, the digitized values are converted to floating-point voltages by dividing the digitized value by the total voltage range of the ADC. Once the data has been converted, the voltages can be used directly or passed through a window comparator. The comparator converts the voltage values to a comparator state. The comparator consists of a window bordered by a high threshold level and a low threshold level. The software compares the current voltage value with the high and low threshold values. When the value is between the two thresholds, the state of the comparator is false and when the value is above or below the threshold, the comparator state is true. Before the software compares the value to the threshold levels, it removes a bias voltage value. The bias voltage value is the level of the ADC when the laser is blocked. In other words, the software comparator removes the ambient light from the voltage value in order to compare the signal of the laser to the threshold levels.

One problem with window comparators is their inability to handle noise. Signals that make rapid transitions across threshold levels cause the comparator state to change rapidly. To remedy this problem, feedback or hysteresis is used to eliminate rapid changes of state due to noisy signals. An electrical hardware device used to solve this problem is the Schmitt trigger[11]. The trigger essentially widens the threshold "line" that the signal must pass through for the comparator to change states. The width of this line or envelope is dictated by the threshold level and the hysteresis level. The envelope size should be adjusted according to the noise level of the signal to eliminate rapid changes in comparator states.

3.1.1 Sensor Library Data

The data structure for the sensor library is shown in Figure 3.4. The complete sensor library code is listed in [12].

```
typedef struct _sensor {

    int         pmac;           /* PMAC ascii file descriptor */
    int         gather;         /* PMAC gather file descriptor */

    unsigned long status;       /* sensor status word */
    time_t      start_time;     /* start time (since Epoch 1/1/70) */

    unsigned long **data32;     /* 32 bit data queue */
    unsigned short *data16;     /* 16 bit data array */
    int         n_data;         /* size of data buffers in queue */
    int         dhead;          /* data queue head */
    int         dtail;          /* data queue tail */
    int         dsize;          /* data queue size */

                                /* File I/O */

    FILE        *file;          /* record/playback file */
    long        block_size;     /* size of data block */
    long        play_timeout;   /* playback timeout (ns) */
    long        curr_pos;       /* file position */

    long        cal_bias_pos;   /* bias signal position */
    long        cal_mean_pos;   /* mean signal position */
    long        data_pos;       /* start of data */

    void (*notify)(struct _sensor *);   /* called when array filled */

    int         n_scan;         /* number of scans */
    float       period;         /* sample period (ms) */

    short       ***queue;       /* queue of arrays of scans */
    int         qhead;          /* write location of queue */
    int         qtail;          /* read location of queue */
    int         qsize;          /* queue size */

    clock_t     *tick;          /* timestamp in clock ticks */

    int         n_average;      /* # of samples to average */

                                /* window comparator parameters */

    float       threshold_lo[N_DIODE];  /* low threshold level */
    float       threshold_hi[N_DIODE];  /* high threshold level */
    float       hysteresis_lo[N_DIODE]; /* low hysteresis level */
    float       hysteresis_hi[N_DIODE]; /* high hysteresis level */

    float       bias[N_DIODE];          /* signal bias */

    float       threshold[N_DIODE];     /* threshold percentage */
    float       hysteresis[N_DIODE];    /* hysteresis percentage */

} sensor_t;
```

Figure 3.4: The sensor library data structure

`int pmac`: The file descriptor for ASCII communication with the PMAC board and device driver.

`int gather`: The file descriptor for the gather node of the PMAC board and device driver.

`unsigned long statusd`: The sensor library status word. Each bit represents a mode or status of the sensor library.

`time_t start_time`: The time when the gathering of data was started.

`unsigned long **data32`: Low level circular queue for 32 bit or 4 byte data. This queue is used for reading from a file or a device driver. The queue acts as a buffer between the high-priority reading thread and the lower-level recording (writing to a file) thread.

`unsigned short *data16`: Array of 16 bit data for reading and writing to a file. File data after the 32 bit clock counter are saved in 16 bit format to reduce the file size since most analog-to-digital converters use 12 bit values.

`int n_data`: Number elements in each array of data (`n_scan*(1+N_DIODE)`).

`int dhead`: The entry point of the data queue.

`int dtail`: The exit point of the data queue.

`int dsize`: The number of arrays of data in the data queue.

`FILE *file`: The pointer for reading and writing to a file.

`long block_size`: The size of one block of file data.

`long play_timeout`: Time delay between reading a buffer of data from a file. The delay is used to simulate the amount of time during data gathering.

`long curr_pos`: Current read/write position in a file.

`long cal_bias_pos`: Position of bias calibration data in a file.

`long cal_mean_pos`: Position of averaged calibration data in a file.

`long data_pos`: Position of the start of the data in a file.

`void (*notify)(struct _sensor *)`: A pointer to a function that is called when sensor library has read one data buffer. This function can be an internal function or a supplied function. The function is specified as an argument to `sensorGatherArray()`.

`short ***queue`: Circular queue of data buffers. Each photodiode element is one 12-bit analog value (`short`). An array of `N_DIODE` photodiode elements (`short*`) is one scan. An array of scans (`short**`) is one buffer of data. A queue is an array of data buffers (`short***`).

`int n_scan`: The number of scans in each data buffer (`short**`).

`float period`: The actual sample period in milliseconds as dictated by the hardware A/D board.

`int q_head`: The entry point of the queue.

`int q_tail`: The exit point of the queue.

`int q_size`: The total number of buffers in the queue.

`clock_t *tick`: An array of clock ticks representing the time when each data buffer was gathered.

`int n_average`: Number of samples to average in running-average filter.

`float threshold_lo[N_DIODE]`: An array of low threshold values for the window comparator. Each element in the array represents the low threshold value for one photodiode element.

`float threshold_hi[N_DIODE]`: An array of high threshold values for the window comparator. Each element in the array represents the high threshold value for one photodiode element.

`float hysteresis_lo[N_DIODE]`: An array of low hysteresis values for the window comparator. Each element in the array represents the low hysteresis value for one photodiode element.

`float hysteresis_hi[N_DIODE]`: An array of high hysteresis values for the window comparator. Each element in the array represents the high hysteresis value for one photodiode element.

`float bias[N_DIODE]`: An array of bias voltages for the window comparator. Each element in the array represents the bias voltage signal for one photodiode element. The bias is removed from the input signal before the signal is passed through the window comparator. The bias signal is removed so that the detector only examines the laser induced portion of the photodiode signal.

`float threshold[N_DIODE]`: An array of percentage threshold values for calibrating the window comparator. Each element in the array represents a percentage that is used for calculating both the high and low threshold levels. The threshold levels are based on mean signal values for each photodiode element and are obtained during the calibration of the window comparator. The percentages are expressed in non-negative values where 1 represents 100 percent of the average photodiode signal.

`float hysteresis[N_DIODE]`: An array of percentage hysteresis values for calibrating the window comparator. Each element in the array represents a percentage from the associated threshold level. The percentages are expressed in non-negative values where 1 represents 100 percent of the high and low threshold levels.

3.1.2 Sensor Library General Routines

`sensor_t* sensorOpen()`

This function allocates memory for and initializes the sensor data structure, opens communication to the device driver and initializes the hardware. It returns an address to the sensor structure used by all sensor library routines.

`void sensorClose(sensor_t* s)`

This function resets the hardware, closes communication to the device driver and frees allocated memory.

`int sensorNotifyOff(sensor_t* s)`

This function stops the sensor library from updating the circular queue. New data will not be written to the queue; however, data already in the queue can be read. The function returns 0 if successful and -1 if not.

`int sensorNotifyOff(sensor_t* s)`

This function updates the circular queue. New data will be written to the queue. The function returns 0 if

successful and -1 if not.

3.1.3 Averaging Routines

```
int sensorSetAverage( sensor_t* s, int n_samples )
```

This function sets the size of the window for the running-average filter. A size of 0 turns averaging off. The function returns 0 if successful or -1 if an error occurs.

```
int sensorGetAverage( sensor_t* s )
```

This function returns the window size of the running-average filter. It returns -1 if an error occurs.

3.1.4 Buffered Data Gathering Routines

```
float sensorGather(sensor_t* s, int n_scans, float period)
```

This function allocates buffer memory and starts the data gathering process by invoking the low-level function `sensorGatherArray()`. The size of each buffer is specified by the `n_scans` argument. The `period` argument specifies the time interval in milliseconds between each scan. This high-level function invokes the low-level function `sensorGatherArray()`. The function prints out an error message and terminates the application program if there is an error. It returns the actual sample period in milliseconds dictated by the PMAC board if no error occurs.

```
short** sensorGetData(sensor_t* s, clock_t *tp)
```

This function returns an address of a two-dimensional array of gathered data. The clock value at the time the data was collected is stored in the address `tp`. Scans of data are stored in a circular queue. The sensor library reads the device driver and places the scans of data at one end of the queue. At the same time, the application program reads the opposite end of the queue. If the queue is empty, the function waits until data are available. The scans of data are two-dimensional arrays of photodiode signal levels. One scan is defined as a `short*` array that contains `N_DIODE` number of 12-bit digitized values (`short`) for each photodiode element. The `short**` array is an array of scans. The sensor library stores the clock value at the time the data was collected in the address `tp`, if it is not `NULL`.

```
int sensorSetQueueSize( sensor_t *s, int n )
```

This function sets the queue size. The size must be set before queue memory is allocated. The function returns -1 if the queue's memory is already allocated and 0 if setting the queue size was successful.

```
int sensorGetQueueSize( sensor_t *s )
```

This function returns the size of the data queue. If the sensor instance argument is invalid, a -1 is returned to indicate an error.

3.1.5 Unbuffered Data Gathering Routines

`float sensorGatherArray(sensor_t* s, short **scans, int n_scans, float period, void (*notify)(sensor_t*))`

This function is the low-level interface for collecting and storing data. The `short** scans` argument is an array of scans, where `int n_scans` is the size of the array. One scan is defined as a `short*` array that contains `N_DIODE` number of 12-bit digitized values (`short`) for each photodiode element. The `float period` argument specifies the time interval in milliseconds between each scan. The `void (*notify)(sensor_t*)` argument is the function that is called when the `short **scans` array is filled. The following steps are performed by `sensorGatherArray()`:

1. Creates thread attributes and raises thread priority. The priority is raised to ensure that data are always read before the application program processes the data.
2. Initializes sensor library data for gathering. The exact sample period is calculated based upon the PMAC servo update interval. If a name of a function is specified as an argument to `sensorGatherArray()`, a bit is set in the library status word to direct the thread to call the specified function.
3. If direct access to data is desired, the circular queue, `short ***queue`, is set to the address of the two-dimensional array passed as the argument, `short **scans`.
4. Setup and start the device driver to gather data. Pass to the device driver the number of scans, number of servo cycles between scans and the PMAC memory address of the source of data. The device driver starts gathering data.
5. Start clock. The clock is used for time stamping each array of scans.
6. Create, start and detach the high-priority sensor thread. The `sensorThread()`, is started and detached from the current process. The thread is detached because it does not return.

If any one of these steps is unsuccessful, the function prints out an error message and terminates the application program. If successful, the function returns the exact sample period in milliseconds, as dictated by the PMAC board.

3.1.6 Window Comparator Routines

`int sensorConvertToFloat(sensor_t *s, short** analog, float** f)`

This function converts the array of gathered data, `short** analog`, from 12-bit digitized values to floating-point voltages. The `float*` array contains `N_DIODE` number of voltage values (`float`) for each photodiode element. The `float** f` array is an array of scans. The function returns -1 if an error occurred and 0 if successful.

`int sensorComparator(sensor_t *s, short** analog, char** binary)`

This function passes data through a window comparator or detector. The function converts the array of gathered data, `short** analog`, from 12-bit digitized values to floating-point voltages and then converts them to binary detector states. The `char** binary` array is a two-dimensional array of detector states. The `char*` array contains `N_DIODE` number of detector states (`char`), one for each photodiode element. The function returns -1 if an error occurred and 0 if successful.

`int sensorSetWindow(sensor_t *s, int element, float threshold, float hysteresis)`

This function sets the threshold levels for the window comparator. The `int element` argument specifies the desired photodiode element. The `float threshold` argument specifies the distance at which the threshold levels should be set from the average signal, in terms of a percentage of the average signal. The average signal level is determined during calibration. The `float hysteresis` arguments specifies the distance at which the hysteresis levels should be set from the threshold levels, in terms of a percentage of the threshold level. The function returns -1 if an error occurred and 0 if successful.

`int sensorGetWindow( sensor_t *s, int element, float *threshold, float *hysteresis )`

This function returns the threshold and hysteresis values. The function returns -1 if an error occurred and 0 if successful.

`int sensorCalibrate(sensor_t *s)`

This function starts calibration of the window comparator. Calibration of the threshold and hysteresis levels are dictated by the `threshold` and `hysteresis` percentage values passed to function `sensorSetWindow()`. The actual threshold and hysteresis levels are based on an average signal level. The function returns -1 if an error occurred and 0 if successful.

`int sensorSetHighThreshold(sensor_t *s, int element, float threshold)`
`int sensorSetLowThreshold(sensor_t *s, int element, float threshold)`

These functions set the high and low threshold levels of the window comparator. The `int element` argument specifies the desired photodiode element. The `float threshold` is the threshold voltage value. The function returns -1 if an error occurred and 0 if successful.

`int sensorSetHighHysteresis(sensor_t *s,int element,float hysteresis)`
`int sensorSetLowHysteresis(sensor_t *s, int element,float hysteresis)`

These functions set the high and low hysteresis levels of the window comparator. The `int element` argument specifies the desired photodiode element and the `float hysteresis` is the hysteresis voltage value. The function returns -1 if an error occurred and 0 if successful.

`int sensorSetBias(sensor_t *s, int element, float bias)`

This function sets the bias of the window comparator. The `int element` argument specifies the desired photodiode element and the `float bias` is the bias voltage value. The function returns -1 if an error occurred and 0 if successful.

`float sensorGetBias(sensor_t *s, int element)`

This function returns the bias for the window comparator. The `int element` argument specifies the desired photodiode element and the function returns the bias value, in volts.

```
float sensorGetHighThreshold(sensor_t *s, int element)
float sensorGetLowThreshold(sensor_t *s, int element)
```

These functions return the high and low threshold levels for the window comparator. The `int element` argument specifies the desired photodiode element and each function returns the associated threshold value, in volts.

```
float sensorGetHighHysteresis(sensor_t *s, int element)
float sensorGetLowHysteresis(sensor_t *s, int element)
```

These functions return the high and low hysteresis levels of the window comparator. The `int element` argument specifies the desired photodiode element and each function returns the associated hysteresis value, in volts.

```
int sensorCompressData(sensor_t *s, char** old, unsigned long* new)
```

This function takes the detector data in a two-dimensional array and compresses the data into a one-dimensional array. The `char** old` argument is the two-dimensional array of binary states, where each element (`char`) represents the state of the detector. The `char*` array contains `N_DIODE` number of detector states (`char`), one for each photodiode element. The data is converted and stored in the `unsigned long* new` array. Each photodiode element is represented as one bit in each `unsigned long` element of `new`. Photodiode element 0 is the right most bit (LSB) and subsequent elements are to the left (MSB). The function returns -1 if an error occurred and 0 if successful.

```
int sensorSaveParameters( sensor_t *s, char *filename )
```

Saves window comparator parameters to filename. Threshold and hysteresis window parameters are saved to a file that can be later used to restore the window comparator configuration.

```
int sensorReadParameters( sensor_t *s, char *filename )
```

Reads window comparator parameters from filename. Threshold and hysteresis parameters are read from a file to restore a previous window comparator configuration.

3.1.7 Data Recording and Playback Routines

Data read from the device driver can be saved to a file to be retrieved and played back at a later time. A circular queue is used to buffer the data between the device driver and the file. The following routines are used to record and playback the collected data. If not otherwise specified, each routine below returns a zero when successful or non zero when an error has occured.

```
int sensorFileOpen( sensor_t *s, char *filename )
```

This function opens the file `filename` if it already exists, or creates it if it does not exist. The file header is read and compared to the current sensor configuration if the file exists and written to the file if it is new. Next, the file positions are set and the file pointer is positioned at the starting data block. Finally, a timeout value is set to estimate the time it takes to collect data from the device driver.

```
int sensorFileClose( sensor_t *s )
```

This function simply stops recording or playing and closes the file.

```
int sensorFileBegin( sensor_t *s )
```

This function stops recording or playing and rewinds the file pointer back to the first sensor data block.

```
int sensorFileEnd( sensor_t *s )
```

This function stops recording or playing and moves the file pointer to the last sensor data block.

```
int sensorFileBack( sensor_t *s, int n_block )
int sensorFileForward( sensor_t *s, int n_block )
```

These function rewind and fast forward the file pointer `n_blocks` data blocks, respectively.

```
int sensorFilePause( sensor_t *s )
```

The function stops recording or playing.

```
int sensorFileStop( sensor_t *s )
```

This function stops recording or playing and moves the file pointer back to the starting data block.

```
int sensorFileRecord( sensor_t *s )
```

This function starts the recording of the sensor data to a file. An error occurs if the file is currently being played.

```
int sensorFilePlay( sensor_t *s )
```

This function starts the playing of the sensor data from a file. An error occurs if the file is currently being recorded.

```
long sensorFilePosition( sensor_t *s )
```

This function returns the position of the file pointer.

```
int sensorFilePlayEOF( sensor_t *s )
```

This function returns a true (1) if the end of file has been reached, false (0) if not.

```
int sensorFileSetPlayTimeout( sensor_t *s, long timeout )
long sensorFileGetPlayTimeout( sensor_t *s )
```

These functions set and get the play timeout value, respectively. The timeout value is used to mimic the delay associated with reading the device driver. The units are in nanoseconds.

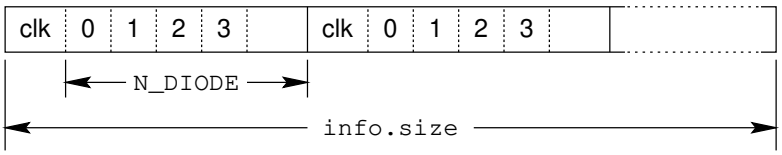
3.1.8 Internals of the Sensor Library

Internal routines are only used by the sensor library and cannot be used by applications.

`void *sensorThread(sensor_t *s)`

This function is the high priority thread that collects the sensor data from the data acquisition board and places the data in buffers for lower priority threads to process. It runs at high priority so that it is scheduled before any other thread in the data pipeline to prevent the loss of data during the collection process. The library keeps track of the time when data was collected and prints out a warning if data are lost.

Data from the PMAC device driver are stored in a low-level circular queue, `data32`. The queue is used to buffer the data between the device driver and a disk file. The data from the device driver enters at the head of the queue in the same format of the PMAC buffer, a single dimensional array as shown in 3.5. The `info.size` size calculates the largest buffer size that is a multiple of `(N_DIODE+1)`. Data exits the queue at the tail by the data recording thread, `sensorRecordThread()`. The head and tail move independently of each other, but a warning is displayed if the head catches up to the tail. When the library is not saving data to a file, neither the head nor the tail move.




```
info.size = (N_DIODE+1)*(int)( GATHER_MAX_SIZE
                                /(float)(N_DIODE+1));
```

Figure 3.5: Data format in PMAC memory

After data are saved in the low-level `data32` queue, the first clock tick of the block of scans is stored into an array `ticks`. The subsequent clock ticks for the rest of the block are ignored. For buffered data acquisition, photodiode signal values are stored in a circular queue of two dimensional arrays, shown in Figure 3.6, one dimension indexed with the scan and the other dimension indexed with the photodiode element. Each clock tick in the array of clock ticks are associated with each block of queued scans.

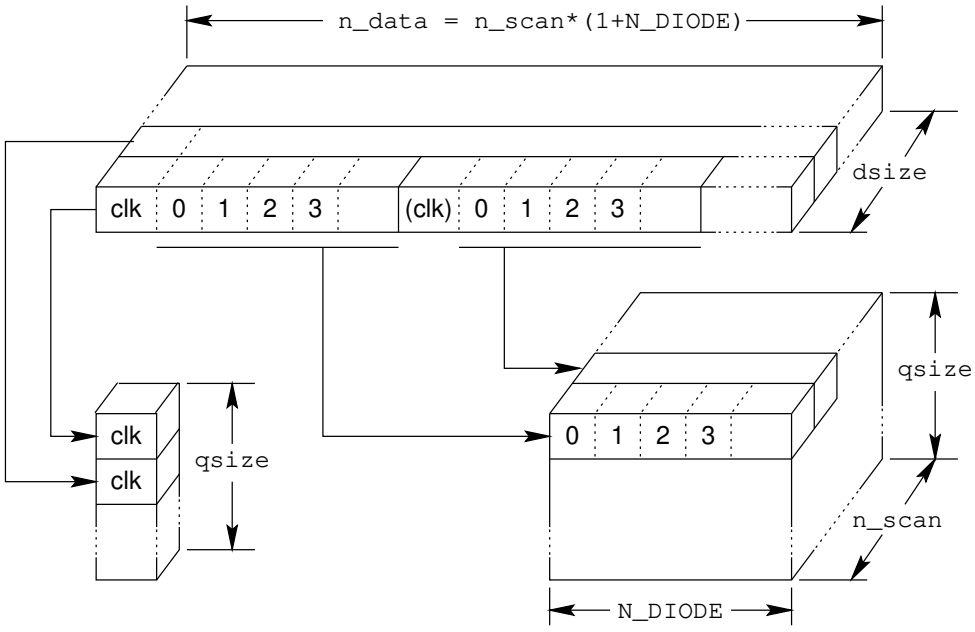


Figure 3.6: Low-level 32 bit data format, array of clock ticks and high-level data queue

The clock ticks and the photodiode signal values are stored at the head of the queue by the high priority thread. A specified function passed as an argument to the function `sensorGatherArray()` is then called to indicate that data has been written to the queue, only if a bit is set in the `status` word. The setting and clearing of this bit is done in functions `sensorNotifyOn()` and `sensorNotifyOff()`, respectively. For buffered data acquisition, the `sensorNext()` function is specified by the sensor library and is decribed below.

Averaging is also performed in the high-priority thread if a `status` bit is set by the function `sensorSetAverage()`. A circular queue is used to calculate a running average by storing `n_average` number of signal values, summing them up and converting them to a single average value, for each photodiode element. The head and tail are spaced `n_average` data points apart. Data points enter the queue at the head and leave at the tail. A running sum is used for each photodiode element. Each signal value at the head of the queue is converted to a floating point number and added to the sum. Each signal value at the tail of the queue is converted to a floating point number and subtracted from the sum. An average is calculated, converted back to a 12 bit digitized value and stored at the head of the high-level queue. If the value of `n_average` number of samples has been changed the running sum for each photodiode is set to zero and the values between the head and tail are set to zero. The tail moves away from the head for larger values of `n_average` and the tail moves closer to the head for smaller values.

```
void *sensorNext(sensor_t *s)
```

This function is called from the high priority thread, `sensorThread()`, to move the head of the queue to the next buffer during buffered data gathering, where data enters. A warning is printed out to the standard error terminal if the head reaches within one buffer of the tail, called overrun, indicating that a loss of data is possible. Increase the size of the queue if overruns occur frequently. Next, the function signals a semaphore to indicate that data are ready to be read from the queue at the tail. The queue is drained at the tail during buffered data collection by the function `sensorGetData()` by waiting on this same semaphore.

`int sensorCalibration( sensor_t *s )`

This function is used to read and/or store a block of data scans used for calibrating the window comparator for a specified number of photodiode elements. The high priority thread `sensorThread()` calls this function when a bit is set to perform the calibration in the `status` variable. The `status` variable also contains bits for each of the desired photodiode elements to calibrate.

Calibration in conducted with two sets of data. The first set of data is collected without the laser signal present, to determine bias signals for each specified photodiode element. The bias signals are used to remove the portion of the signal that is not from the laser (e.g. sunlight). The routine then calculates the average bias signal for each specified photodiode element. In addition, bias signal minimum, maximum and standard deviation are calculated and printed out for informational purposes only. The next set of data is collected with the laser signal present and is used to calculate the window comparator threshold and hysteresis levels. The bias signal is removed and the remaining signal values are averaged for each specified photodiode element of the data. The threshold and hysteresis levels are determined based on the specified percentage of the voltage range of the analog-to-digital converter. The signal minimum, maximum, mean, standard deviation, threshold levels and hysteresis levels are printed out for each of the desired photodiode element to the standard error terminal.

`void *sensorReadData(sensor_t *s)`

This function is used by the high priority thread `sensorThread()` and the function `sensorCalibration()` to read low-level data from the device driver or from a file.

If the data comes from a file, the first data point is read as the 32 bit clock tick value for the entire block of scans. The rest of the block of data is read as 16 bit data, which is converted to 32 bit data. Finally, a delay is added to mimic the collecting time of the data acquisition board.

The data comes from the device driver if the data does not come from a file. If a file is used to store the data, a semaphore for the low-level `data32` queue is signaled, indicating that previous queued data are available for saving to a file. The head is then moved to the next empty buffer in the data queue. If the head meets up with the tail, an overrun warning is issued indicating the loss of data. Increase the value of `DATA_QUEUE_SIZE` if overruns occur frequently during data recording to a file. Lastly, the data from the device driver are placed at the head of the data queue. Note that the signaling of the data semaphore and the moving of the head occurs before the device driver is read because current data will be processed by the high priority thread (e.g. calibration, averaging and placing the data in a high-level queue).

`void *sensorRecordThread(sensor_t *s)`

This function is a low priority thread that saves data from the low-level `data32` queue to a file. The high priority `sensorThread()` thread reads the data from the driver and places them at the head of the `data32` circular queue. A semaphore is signaled to write the data to file one data block at a time so it does not interrupt the data gathering process. By writing only one data block at a time, the low priority thread can be switched out of context by the higher priority thread and resumed when the high priority thread is idle.

The data file is comprised of three sections, as shown in Figure 3.7. The first section is the header, which specifies the configuration of the sensor that is associated with the file. The header contains the number of photodiodes that the data was collected, the calibration block data size, the number of scans per block of data and the time. The next section of calibration data consists of two sections, the bias data section and the mean data section. The bias data contains the photodiode signal levels when the laser is off. The mean data contains averaged photodiode signals with the laser on. The last section contains an unlimited number of

sensor data blocks from the recording process. Each block of data contains a 4 byte (32 bit) clock value, 2 bytes (16 bits) for each photodiode element, followed by groups of 2 byte clock values, which are ignored, and photodiode values until `n_scan` number of blocks have been collected.

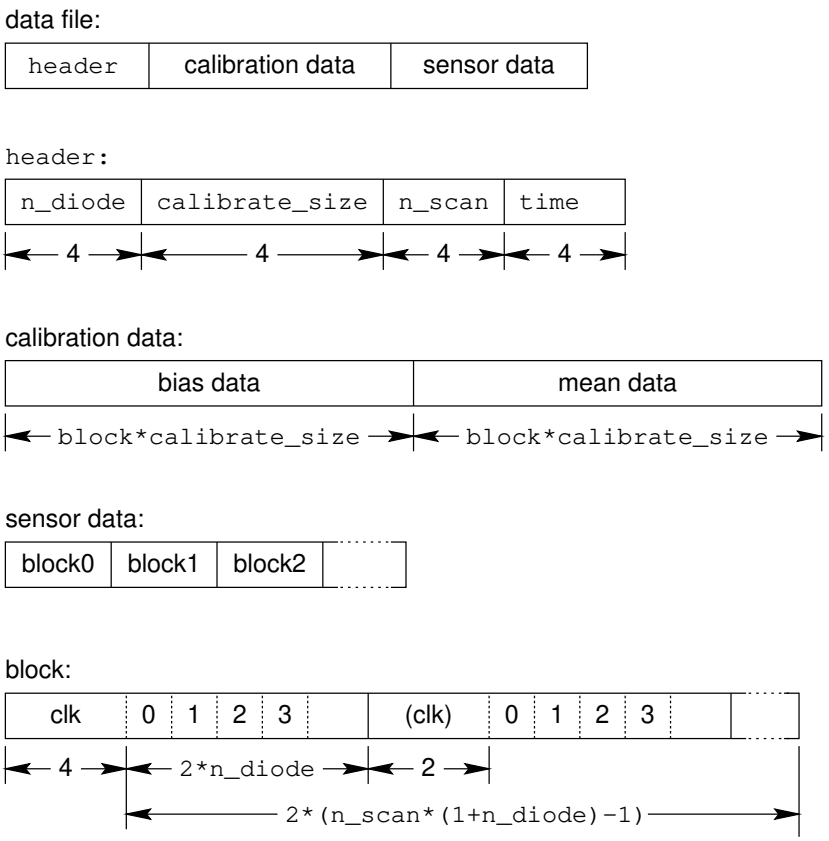


Figure 3.7: Data Recording File Format. Sizes are in bytes.

3.2 Vehicle Delineation Library

The purpose of the vehicle delineation library is to convert sensor library data to vehicle delineation data, such as vehicle front velocity, rear velocity, average acceleration and ultimately length. Figure 3.8 shows the functional diagram of the vehicle delineation library, indicating the flow of data between the functional aspects of the library.

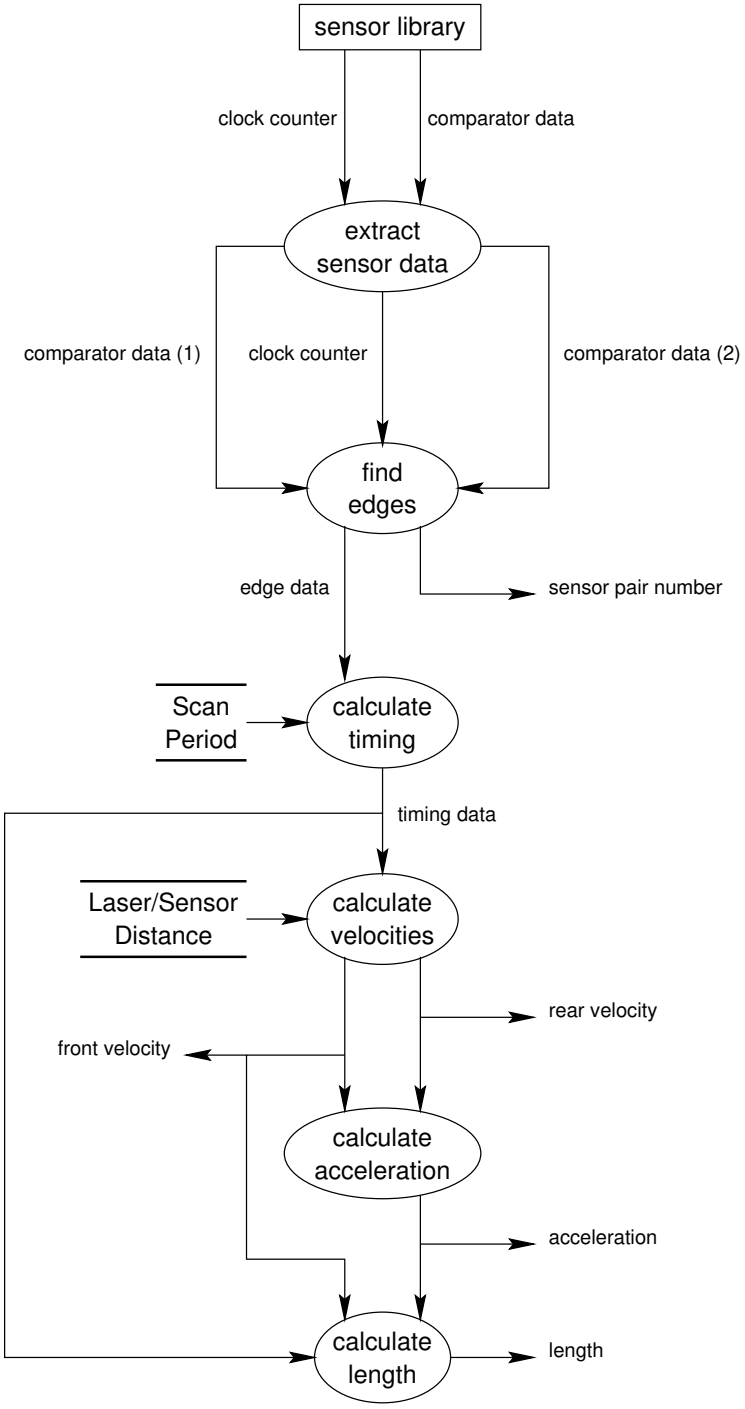


Figure 3.8: Vehicle Delineation Library Functional Diagram

The vehicle library obtains photodiode data from the sensor library. The photodiode data are comprised of two separate photodiode sensors combined into one block of data. The block of photodiode data is then passed through the sensor library comparator and converted to comparator states. A comparator state indicates if a photodiode signal is outside or between two threshold levels.

Next, the software finds the timing of the front of the vehicle and the rear of the vehicle by using comparator data from the two separate photodiode sensors. When a vehicle passes under a particular photodiode

element, the comparator state changes, creating an "edge." Every photodiode element for each sensor is coupled with another element, forming a "pair", (e.g. element 1 for the first sensor corresponds to element 1 for the second sensor).

The software looks for four edges for each pair of photodiodes, corresponding to the front of the vehicle and the back of the vehicle. Once the edges are found, the software calculates the time between the first two edges and the last two edges using the clock count from the sensor library. A clock count is associated with each block of data so each instant of time for every sample of data can be calculated. Figure 3.9 shows the timing of the two sensors. The front of the vehicle is indicated by times t_0 and t_1 and the rear of the vehicle is indicated by t_2 and t_3.

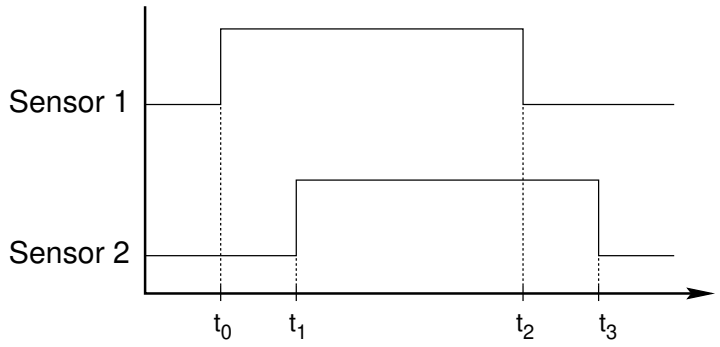


Figure 3.9: The timing diagram when a vehicle passes under two sensors.

The timing data are used by the vehicle library to calculate the front velocity and the rear velocity of a detected vehicle. The software uses the time interval of the two sensors and the distance between them to calculate the front velocity v_0,

$$v_0 = \frac{d}{t_1 - t_0}$$

where d is the distance between the two sensors. The rear velocity is calculated in a similar fashion

$$v_1 = \frac{d}{t_3 - t_2}$$

The velocities are used to calculate the average acceleration of the vehicle as it passed under the two laser sensors. The calculation is based on the front velocity, v_0, and the rear velocity, v_1, and the time of the first rising edge, t_0, and the time of the second rising edge, t_2.

$$a = \frac{v_1 - v_0}{t_2 - t_0}$$

The length of the vehicle is determined from the front velocity, the timing of the first edges and the average acceleration

$$l = v_0 (t_2 - t_0) + \frac{1}{2} a (t_2 - t_0)^2$$

The vehicle delineation library obtains the current time and then groups all of the above calculated parameters into one block of data and makes the data available for reading by an application. The group contains the timing information, the front and rear velocities, the average acceleration and finally the calculated length of the vehicle.

3.2.1 Vehicle Data Format

Figure 3.10 shows the data structures used by the vehicle delineation library. The complete vehicle library code is listed in [12].

```
typedef struct vehicle_data {

    char        pair;   /* which photodiode pair is this for? */

    float       v[2];   /* velocity (m/s) */
    float       a;      /* average acceleration (m/s/s) */
    float       l;      /* length */

    time_t      time;   /* time (s) since the Epoch (1/1/70) */

} vehicle_data_t;


typedef struct _vehicle {

    sensor_t    *sensor;        /* sensor instance */

    unsigned long status;                       /* module status */
    unsigned long timing_status[N_PAIR];        /* timing status of each
                                                 * photodiode pair */

    int         n_scans;        /* buffer size */
    float       period;         /* actual time between scans */

    float       distance;       /* between laser lines */

    short       ***data;                /* photodiode data queue */
    int         dsize;                  /* size of data queue */
    int         dtail;                  /* reading end of queue */
    int         dhead;                  /* writing end of queue */

                                        /* vehicle data queue */
    vehicle_data_t      *queue;         /* circular queue */
    int                 qsize;          /* queue size */
    int                 qtail;          /* reading end of queue */
    int                 qhead;          /* writing end of queue */

    struct timeval      timeout;        /* semaphore wait timeout */

    void (*notify)(struct _vehicle *v); /* called when data ready */

} vehicle_t;
```

Figure 3.10: The vehicle library data structures

The first data structure contains the vehicle delineation data that is available to applications. Vehicle delineation data for each photodiode pair is grouped into a single structure. Each member of the structure is described below:

`char pair`: The number that identifies the photodiode pair for this set of delineation data.

`float v[2]`: An array of velocities associated with the calculated front and rear velocities of the vehicle.

`float a`: The calculated average acceleration of the vehicle.

`float l`: The calculated length of the vehicle.

`time_t time`: The calendar time when the vehicle passed under the sensor.

The following data structure is internal to the vehicle library.

`sensor_t *sensor`: The instance of sensor. This pointer points to the internal data structure used by the sensor library.

`unsigned long status`: The vehicle library status word. Individual bits of this word are used by the library to indicate the state or mode of the library.

`unsigned long timing_status[N_PAIR]`: An array of timing status word for each photodiode pairs. This word is used to indicate the timing edges found by the library.

`int n_scans`: The number of scans read from the sensor library per scan.

`float period`: The actual sample period in milliseconds as dictated by the hardware A/D board returned from the sensor library.

`float distance`: The distance in meters between the laser lines on the road. This value is used to determine the velocity of the front and rear of the passing vehicle.

`short ***data`: A circular queue of photodiode data. The queue consists of two-dimensional arrays obtained from the sensor library. One index of the array is one scan of photodiode signals and the other index of the array is a photodiode element.

`int dsize`: The size of the circular queue. The size determines the number of scans of data the library stores.

`int dtail`: The location where data exits the circular queue. The tail moves through the queue as data are used by the vehicle library.

`int dhead`: The location where data enters the circular queue. The head moves through the queue when data are read from the sensor library.

`vehicle_data_t *queue`: The circular queue of vehicle delineation data. The queue is a single dimensional array of data of vehicles that were detected.

`int qsize`: The size of the circular queue of vehicle delineation data. The size specifies the amount of data the library buffers.

`int qtail`: The circular queue exit point of vehicle delineation data. The tail moves through the queue as the data are requested from the application program.

`int qhead`: The circular queue entry point of vehicle delineation data. The head moves through the queue as vehicles are detected.

`struct timeval timeout`: The wait timeout period for vehicle delineation data. The `vehicleGetData()` function waits for the specified time if no vehicle data is available.

`void (*notify)(struct _vehicle *v)`: A low-level function to signal that vehicle delineation data are available. This function is invoked by the vehicle library thread to indicate that vehicle data can immediately be processed. This function is internal to the library for buffered data processing.

3.2.2 Vehicle Delineation Library Routines

A function interface is provided to allow the application program to setup, start and stop the vehicle detection process. The routines interact with the underlying sensor library so the application does not have to. Raw sensor data are obtained from the vehicle library as well. Each routine is described below:

`vehicle_t *vehicleOpen( float distance )`

Opens communication with the vehicle detection library. The `distance` argument specifies the distance between the two laser lines on the road. The function allocates memory for the library data structure, initializes members of the structure, opens communication with the sensor library and returns the memory address of the structure. NULL is returned if an error occurs.

`void vehicleClose( vehicle_t *v )`

Closes communication with the vehicle detection library. Communication is closed with the sensor library and memory used by the vehicle library is freed.

`float vehicleDetectData( vehicle_t *v, int n_scans, float period, vehicle_data_t *vehicle_data, void (*notify)(vehicle_t *) )`

Initialize and start unbuffered vehicle detection. A low-level circular queue of `n_scans` are allocated in memory. The desired `period` between scans is passed to the sensor library and the sensor library starts collecting data. The vehicle detection thread is created and started. The thread places detected vehicle data in the `vehicle_data` array and calls the `notify` function to indicate that data has been placed in the array. The function terminates the application program if an error occurs or returns the exact period dictated by the PMAC servo update rate when it is successful.

`int vehicleSetQueueSize( vehicle_t *v, int n )`

Sets the circular queue size of detected vehicle data. Returns a zero if successful or a -1 if an error occured.

`float vehicleDetect( vehicle_t *v, int n_scan, float period )`

Initialize and start buffered vehicle detection. A circular queue of detected vehicle data is allocated in memory based on the current queue size. The arguments `n_scan` and `period` are passed to the `vehicleDetectData()` function, which starts the data collection and vehicle detection process. The exact data collection rate dictated by the PMAC servo update rate is returned if the function was successful or terminates the application if an error occurs.

`vehicle_data_t *vehicleGetVehicle( vehicle_t *v )`

Returns detected vehicle delineation data if they available in the circular queue or NULL if no data are available after a timeout period. The function waits on a data semaphore. If no data are available, it waits for a period of time dictated by the `timeout` library variable and returns NULL. If data are available, the function reads from the tail of the circular queue and returns an address location of the `vehicle_data_t` data structure, containing delineation data for the detected vehicle.

`short **vehicleGetData( vehicle_t *v )`

Returns low-level sensor data that was obtained from the sensor library. The function waits on a data semaphore for an unlimited amount of time. When data becomes available, the function reads from the tail of the `data` queue and returns an array of scans.

`int vehicleDataReset( vehicle_t *v )`

Resets the detection state of vehicle library. All previous vehicle timing data are cleared. The function returns a zero if successful or a -1 if an error occurs.

`int vehicleEnable( vehicle_t *v, int number )`

Enables the photodiode pair `number` for vehicle detection. The vehicle library will detect edges on the `number` pair of photodiodes. The function sets a `status` bit and returns zero if successful and a -1 if an error occurred.

`int vehicleDisable( vehicle_t *v, int number )`

Disables the photodiode pair `number` for vehicle detection. The vehicle library will not detect edges on the `number` pair of photodiodes. The function clears a `status` bit and returns zero if successful and a -1 if an error occurred.

3.2.3 Vehicle Library Internals

The following functions are internal to the vehicle library and cannot be used by application programs. Details of the functions are described below.

`void *vehicleThread( void *arg )`

A high-priority thread that reads photodiode data, processes the data, finds vehicle delineations based in the data and places the delineation data in a buffer. The thread priority should be lower than the data collection thread, but higher than the data recording thread in the sensor library. The thread detects vehicle delineations by using two separate photodiode sensors that are focused a specified `distance` apart. The time differential in the signals allows the thread to calculate the front and rear velocity of the vehicle, the average acceleration and the length. The time of the detected signal is stored and the delineation data are stored at the head of a high-level circular queue. Vehicle data can be obtained from the queue by an application program.

The thread obtains photodiode signal data from the sensor library and places the data at the head of a low-level circular queue. This queue is used to buffer the raw photodiode signals so an application program may request data periodically from the tail of the queue. The photodiode signal data are passed through a window comparator that converts the data to binary states. Basically, the comparator indicates if the laser is blocked (true) by a vehicle or not (false). The thread moves the head of the low-level queue to the next buffer and signals that photodiode signals are available. The thread checks a reset `status` bit and clears the detection states if the bit is set.

After the photodiode signal is queued and converted to comparator states, the thread finds the changes of state for each element of the two photodiode sensors. Consecutive elements of each sensor are associated in pairs. The sensor library reads both photodiode sensors as one scan, one sensor before the other. For each scan of photodiode data, the thread examines each photodiode element pair. The thread ignores pairs that are disabled for detection.

The thread then finds any change in comparator states, called edges. The edge detection algorithm was written so it does not matter which sensor is the leading sensor, as long as they are consistent. Once an leading edge is detected on a particular sensor, that sensor must be the first trailing edge. As a result, this algorithm supports a vehicle passing backwards through under laser detector. The time of the scan is calculated based on the first scan clock tick once two edges have been detected. Once all four edges have been found, the front

velocity of the vehicle is calculated based on the specified `distance` between the laser lines and the time difference between the first two edges. The rear velocity is calculated in the same manner, based on the last two edges. Average acceleration is calculated from the two velocities and then the length is calculated from the front velocity, the time difference between the first edge and the third edge and the average acceleration.

After the length is calculated and stored, the current time is stored as well as the pair number that the calculations were based upon. The thread then calls the specified `notify` function that indicates that data are available in the queue buffer for further processing. The edge detection algorithm loops again for each photodiode pair in each scan of data.

```
void vehicleNext( vehicle_t *v )
```

This internal `notify` function is called from the vehicle library thread to move the head of the vehicle data queue to the next available buffer, where data enters the queue. A semaphore is signaled to indicate that data are available in the queue for the application program to read from the tail. A warning is displayed if the head meets up with the tail, which indicates that data is lost. If these warnings occur frequently, increase the amount of `QUEUE_SIZE` defined `sensor.h` file of the sensor library.

3.3 X Windows System Applications–Xvehicle

X Windows System applications are used to display the photodiode sensor and vehicle delineation data in real time. They use the sensor and vehicle libraries to collect and process the data and display the data in various ways. The applications use common visual interfaces to the window comparator, the file recording and playback operations and the configuration averaging filter. The typical application program is the Xvehicle.

The Xvehicle application is used to display analog photodiode sensor signals and the vehicle delineation data in real time. The signals are displayed on the left of a strip chart that scrolls from right to left, showing the time history of the signals, as shown in Figure 3.11. At the bottom of Xvehicle window is the strip chart signal bar, which shows a colored button for each corresponding signal in the strip chart. Clicking on a signal button turns on or turns off the signal in the strip chart. A solid button indicates the corresponding signal is on. Vehicle delineation data is displayed for each pair of photodiodes as the application receives them. The data includes the front velocity (`v1`) average acceleration (`a`), rear velocity (`v2`) and the resulting length (`l`).

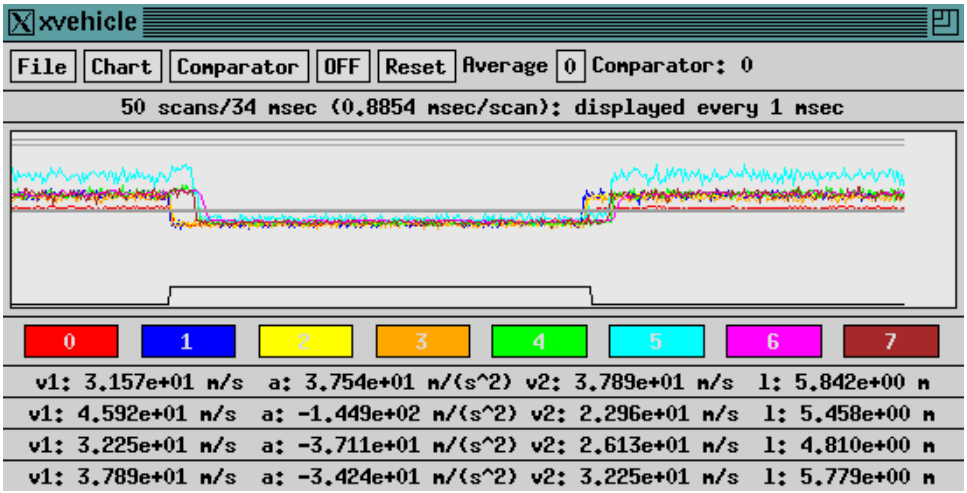


Figure 3.11: Xvehicle

3.3.1 Common Window Interfaces

All applications contain a menu bar, a status window and a strip chart. The `File` menu button in the menu bar contains a quit entry that exits the application and an entry that displays a file recording and playback window. The `Chart` button in the menu bar raises a window to configure the strip chart. The `Comparator` menu button contains entries to read, save, start, configure and adjust the window comparator calibration. The next button in the menu bar is the `ON/OFF` button that switches the collection of data and the strip chart on or off. The next button is the displays the number of photodiode signal samples that are averaged in the strip chart. This number is changed by clicking on the button and entering the desired number of samples in a pop-up window. Below the menu bar is a status window that displays data collection parameters and scan display rate. The data collection parameters include the number of scans per desired scan period and the actual scan rate dictated by the PMAC board. The scan display rate is the rate at which the application requests data from the sensor library. But it does not mean that the data are not actually displayed at that rate because the application waits until data are available.

3.3.2 Stripchart Configuration Window

The `Chart` button raises window for strip chart configuration, as shown in Figure 3.12. A selected window comparator state for one photodiode element can be displayed at the bottom of the xsensor or xvehicle strip chart. A strip chart grid can be turned on or off with the grid `ON/OFF` button and the strip chart scale displayed in the scale button can be changed using the scale menu. The chart voltage offset can be changed by entering a number in the offset field. The `Done` button applies any changes and closes the window. The `Cancel` button closes the window without changing the strip chart. The `Apply` button applies any changes without closing the window.

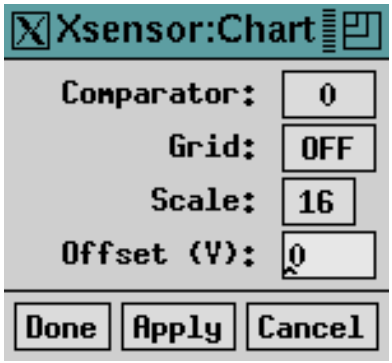


Figure 3.12: Chart configuration window

3.3.3 Comparator Calibration Windows

The `Calibration` menu contains an entry to calibrate the window comparator of the sensor library. When the `Calibrate Element...` entry is selected, the window shown in Figure 3.13 is displayed. The desired element to calibrate is shown at the top in a menu button. A new element can be selected by clicking on the button and selecting another element in the menu. The threshold and hysteresis fields are used to change their respective percentages of analog-to-digital converter voltage range. The `Cancel` button closes the window without changing the comparator configuration. The `Reset` button changes the values back to previous values. The `Set` button sets the parameters in the sensor library. The `Apply` button sets the parameters and starts calibration for the displayed photodiode element. The `Done` button is the same as the `Apply` button, but also closes the window.

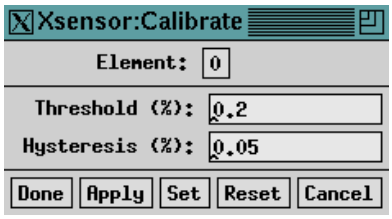


Figure 3.13: Comparator calibration window

When the `Adjust Element...` entry of the `Calibration` menu raises the window shown in Figure 3.14. This window is used to adjust window comparator parameters for a desired photodiode element. The top of the window shows the desired photodiode element. The element can be changed by clicking on the element menu button and selecting the desired element to adjust. Text fields are used to change the individual window comparator parameters. The `Cancel` button closes the window without changing the parameters. The `Reset` button changes any changes parameters back to previous values. The `Set` button sends the parameters to the sensor library and the `Done` button applies the parameters and closes the window.

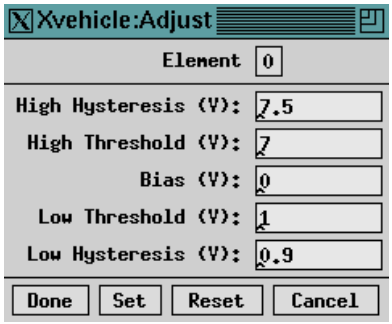


Figure 3.14: Comparator adjust element window

3.3.4 The Record & Playback Window

For each X windows application, the `Record/Play...` entry in the `File` menu button displays the window in Figure 3.15. The `File` menu button contains entries for closing the window as well as creating, opening and closing a data file. The area to the right of the `File` menu button is used to display the name of the current data file, which is blank if no file is open.

Below the `File` menu button two playback speed buttons surrounding a file byte counter. The two speed buttons, − and +, lower the playback speed and increase the playback speed, respectively. Holding these buttons down is the same as repeatedly clicking on them.

Below the counter and speed buttons are the play/record buttons. Starting with the left button, the stop button (with the square) stops the playing or recording of the data file. The play toggle button starts playing the file and the record toggle button starts data recording. The pause toggle button stops the playing or recording of the data file. Below the play/record buttons are the rewind and fast forward buttons. The move to start button, the rewind button, the fast forward button and the move to end button are used to move the file pointer during data file playback. Holding down rewind and fast forward buttons increase the speed of the file pointer.

Figure 3.15: Data record/play window

Chapter 4

Experimental Results

4.1 Sensor Field of View

During indoor resting, each of the 25 sensor elements was tested to verify the size of the field of view and location of each element. To test this a reflective stripwas moved in a small increments across the entire length of the laser line placed approximately 18 feet from the sensor, and the sensor signal level was recorded. The sensor produced a negative voltage in response to the laser, so reflections can be seen as large negative values. Figure 4.1 shows the signal for each of the middle 13 elements plotted as a function of reflector location. By examining these data in more detail it can be seen that the measured field of view of each element is in agreement with the theoretical value. Figure 4.2 shows the signals for three representative elements. As can be seen, the separation between adjacent points of maximum reflection (labeled f, g, and h in Figure 4.2) along the length of the laser line is approximately 4 inches, consistent with our previous calculations. Note that the signal level was recorded with a resolution of 0.1 volt using an oscillascope.

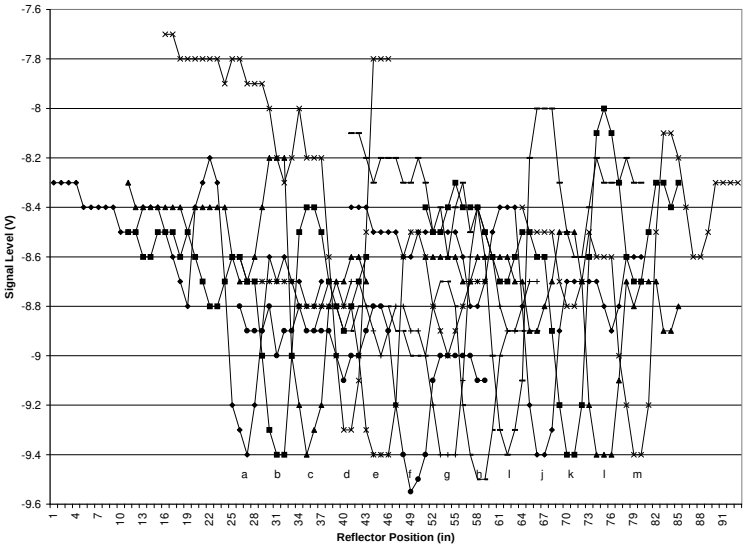


Figure 4.1: Sensor signal for 13 elements

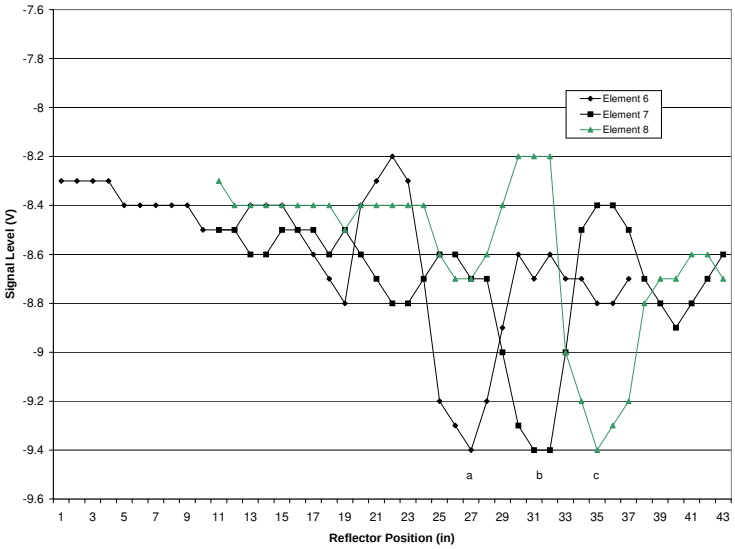


Figure 4.2: Sensor signal for selected elements

4.2 Field Test Results

Field tests have been made on the highway with real traffic at the junction of highway I-5 and I-80 in Sacramento. In the tests eight elements (four elements from each sensor array) were used. The test results shown that the system can collect data in the real traffic environment. The front and rear speeds, length and acceleration can be calculated according to the measured data. From the test results, we found that the front and rear speeds of the four element pairs are different. One reason for this error is because the separating distance of two laser-sensor set are too small. In the current version, the separating distance is 1 foot and sample rate is 1.11 KHz. Assuming a vehicle pass through detector zone at speed of 60 MPH, there are only 12 samples during this distance. This caused the relatively big error in speed measurement. The other cause of error is the slow transition time of the analog signal from system hardware. The transition time is comparable to the time the vehicle crosses two laser lines. This problem will be fixed in the next version field prototype by using digital output. Figure 4.3 is a picture of the test site and the mount of detection system on the bridge acrossing the highway. Figure 4.5 shows the process of testing and correspoding signals and data displayed on the screen. From this picture we can see that the changes in the signal are clear when the vehicle passed the detection system. The picture also shows that the signals and data are stable.

Figure 4.3: Detection system mounted above the highway

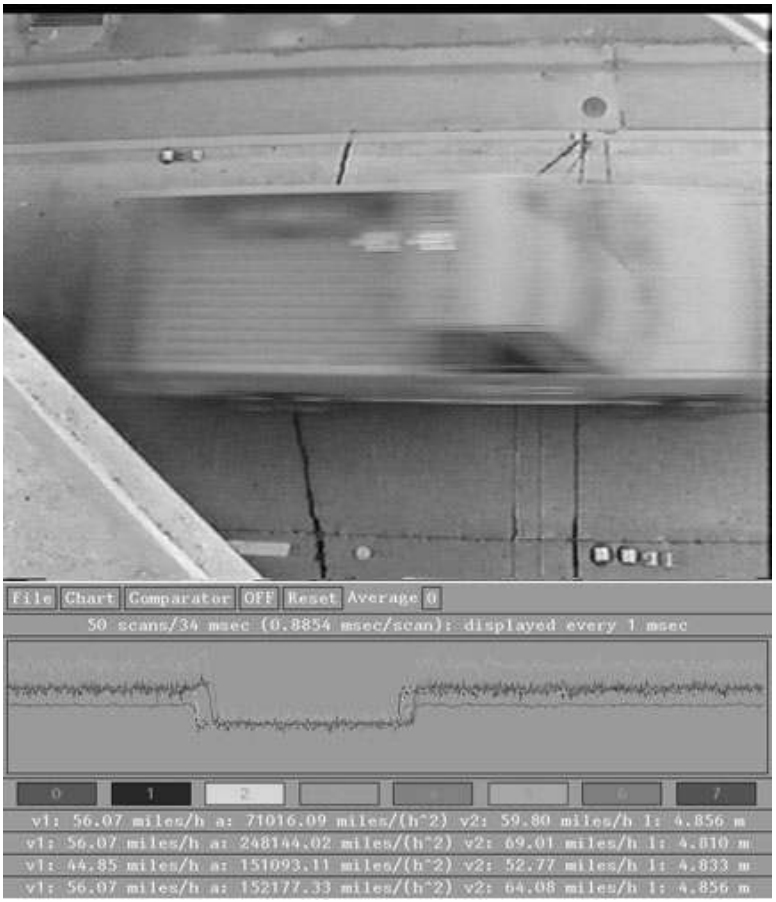


Figure 4.4: Test results(1)

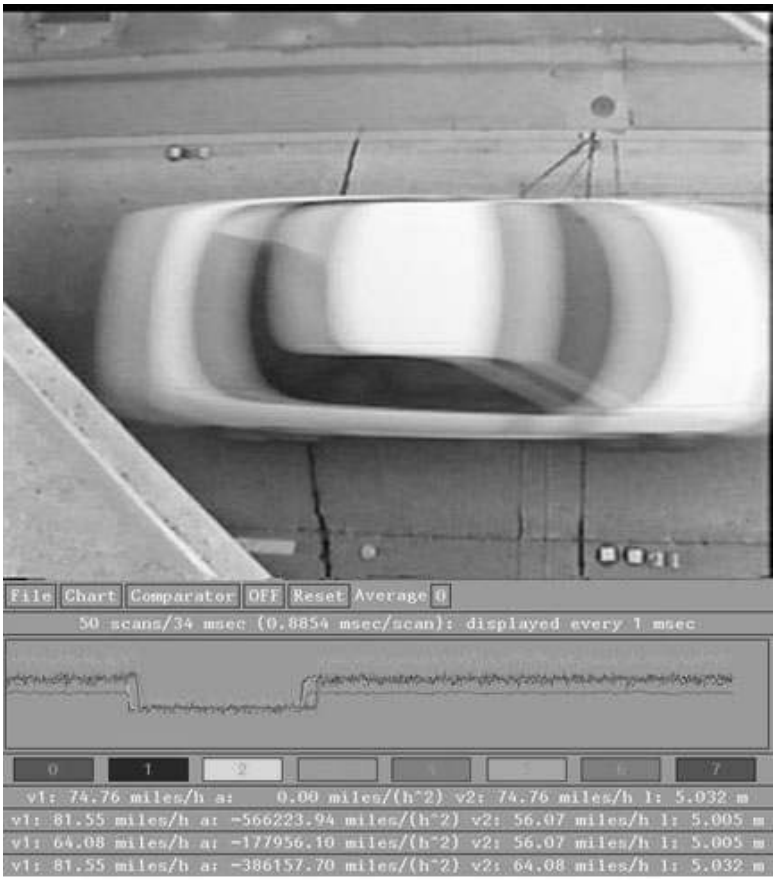


Figure 4.5: Test results(2)

Chapter 5

Current Work

We are continuing work on the improvement of the field prototype system, according to the problems that surfaced in the highway tests. We are designing and building a new version system. A new optical configuration and new signal amplification and processing electronics are being designed and built. We are using low-cost amplifier chips with suitable bandwidth which can meet the high demand of our detection system for signal amplification and without unnecessary features. The time response of the new circuit provides a good match to the pulse length of the laser. The high-frequency signal is amplified effectively and stably without oscillation. The new circuit increases the signal-to-noise ratio by a factor of 5 relative to the previous one. A new method using a TTL logic circuit, instead of a sample-and-hold amplifier, has been designed to handle the short pulse of the signal. By this method, a TTL logic circuit is triggered by the amplified signal and outputs a digital signal. This new design simplifies the circuitry and solves the problem of slow time response caused by the slow sampling rate of the sample-and-hold in the previous version system. Using a digital signal as output from the sensor electronics will significantly reduce the burden of signal processing in the software. The implementation of this method is based on the high signal-to-noise ratio of the new circuitry. Because surface mounted chips and simplied circuits are used, the new version electronics has a compact size. This makes it possible to place all amplifiers for 24 channels on a small printed circuit board. This is important to eliminate the noise, because the signal from the photodiode array is very weak. The new version of the system allows us to gather more channels of the signal, for example up to 24 channels, which is necessary to obtain precise profile of moving vehicles. The cost of the new version circuitry is only one fifth of that of the previous one. This will be helpful for the commercialization of our system in the future.

We are developing software for the new version system using a Microsoft platform because of its compatibility, low-cost and rich support from third party. In order to ensure the high real-time performance of the software, a real-time subsystem RTX based on the Microsoft Windows NT is used. It allows deterministic real-time and non-real-time processing within the same computer. National Instrument data acquisition boards will be used in our detection system for signal input hardware. An AT-MIO-16E-10 board will be used for analog input and a PCI-DIO-32SH or PCI-DIO-96 digital I/O board as digital input.

Chapter 6

Conclusions

Over the past year we have developed a field prototype of a laser-based real-time, non-intrusive detection system for measurement of delineations of moving vehicles in real traffic environments. A field prototype has been constructed and some tests have been done on the highway with real traffic. The test results further verified that the principle of our detection system is technically sound and indicated that the algorithm implemented in the software works in most cases. The simple method of detecting vehicle presence based on the absence of reflected laser beam works reliably in the real traffic environment. The real-time data acquisition software has been developed to gather and process the data from system hardware. The vehicle delineation library wss developed to convert data from the sensor library, which was developed in previous project, to vehicle delineation data, such as vehicle velocity, real velocity, average acceleration and length. The speed, acceleration, and length of a detected vehicle can be displayed on the screen simultaneously. All these data of the detected vehicle can be saved with a time stamp to a disk in real time. The data can later be analyzed to improve the performance of our detection system. A significant problem of this version of the field prototype are a slow time response and not high enough signal/noise ratio. These problems will be solved in the current work toward a new version field prototype.

Bibliography

[1] Palen, J., Roadway Laser Detector Prototype Design Considerations, personal communications,1996.

[2] Tyburski, R.M., A Review of Road Sensor Technology for Monitoring Vehicle Traffic, *ITE journal*. Vol. 59, no. 8 (Aug. 1989)

[3] Halvorson, G. A., *Automated Real-Time Dimension Measurement of Moving Vehicles Using Infrared Laser Rangefinders*, MS Thesis, University of Victoria, 1995.

[4] Olson, et al., *Active Near-Field Object Sensor and Method Employing Object Classification Techniques*, *U.S. Patent* No. 5,321,4990, 1994.

[5] Wangler, et al., *Intelligent Vehicle Highway System Sensor and Method*, *U.S. Patent* No. 5,546,188, 1996.

[6] Wangler, et al., *Intelligent Vehicle Highway System Sensor and Method*, *U.S. Patent* No. 5,757,472, 1998.

[7] Wangler, et al., *Intelligent Vehicle Highway System Multi-Lane Sensor and Method*, *U.S. Patent* No. 5,793,491, 1998.

[8] Graeme, J., *Photodiode Amplifiers: Op Amp Solutions*, McGraw-Hill, New York, 1996.

[9] Horowitz, P. & Hill, W., *The Art of Electronics, 2nd Ed.*, Cambridge University Press, New York, 1989.

[10] *American National Standard for the Safe Use of Lasers*, Laser Institute of America, Orlando, 1986.

[11] Horowitz, P. & Hill, W. *The Art of Electronics*, Cambridge University Press, 2nd edition, 1989.

[12] Kirk Van Katwyk, *Design and Implementation of Real-time Software for Open Architecture Integration of Mechantronic System*, thesis for Master of Science, Department of Mechanical Engineering, University of California at Davis, 2000.

Appendix A

Detail Drafts of Mechanical Components

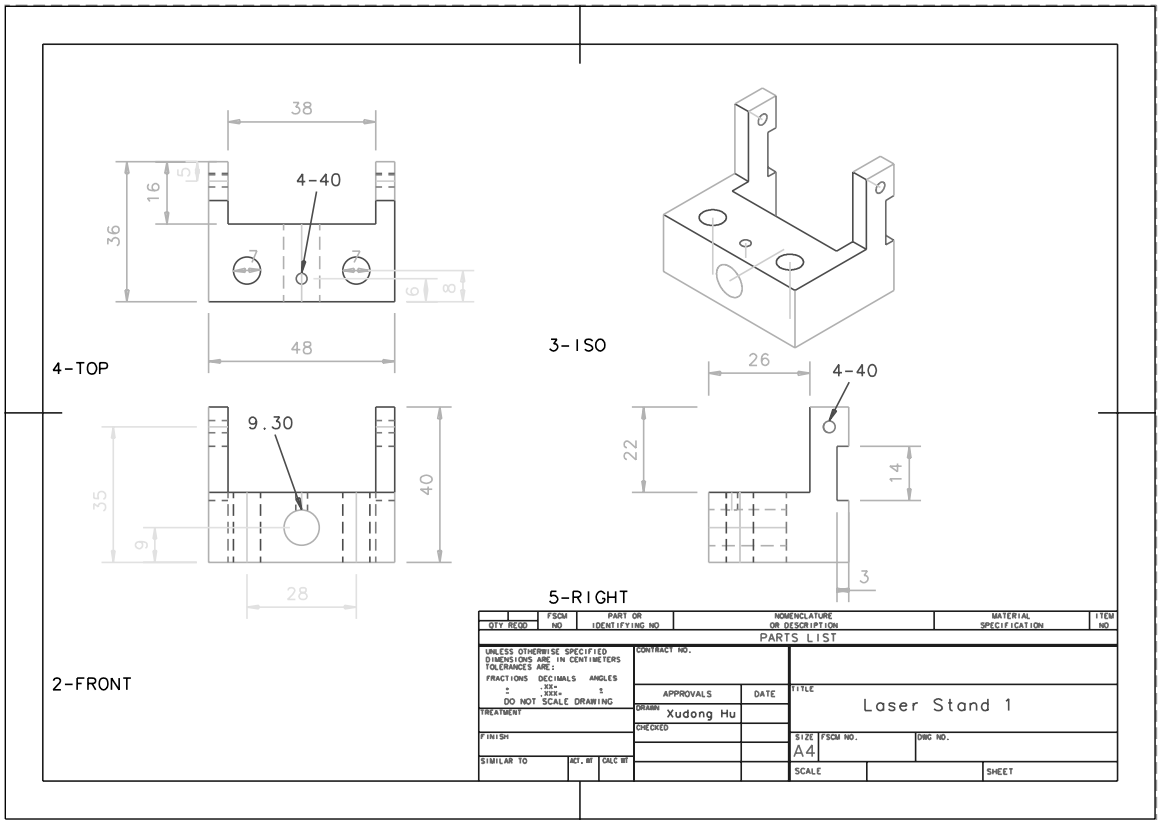


Figure A.1: Laser stand 1

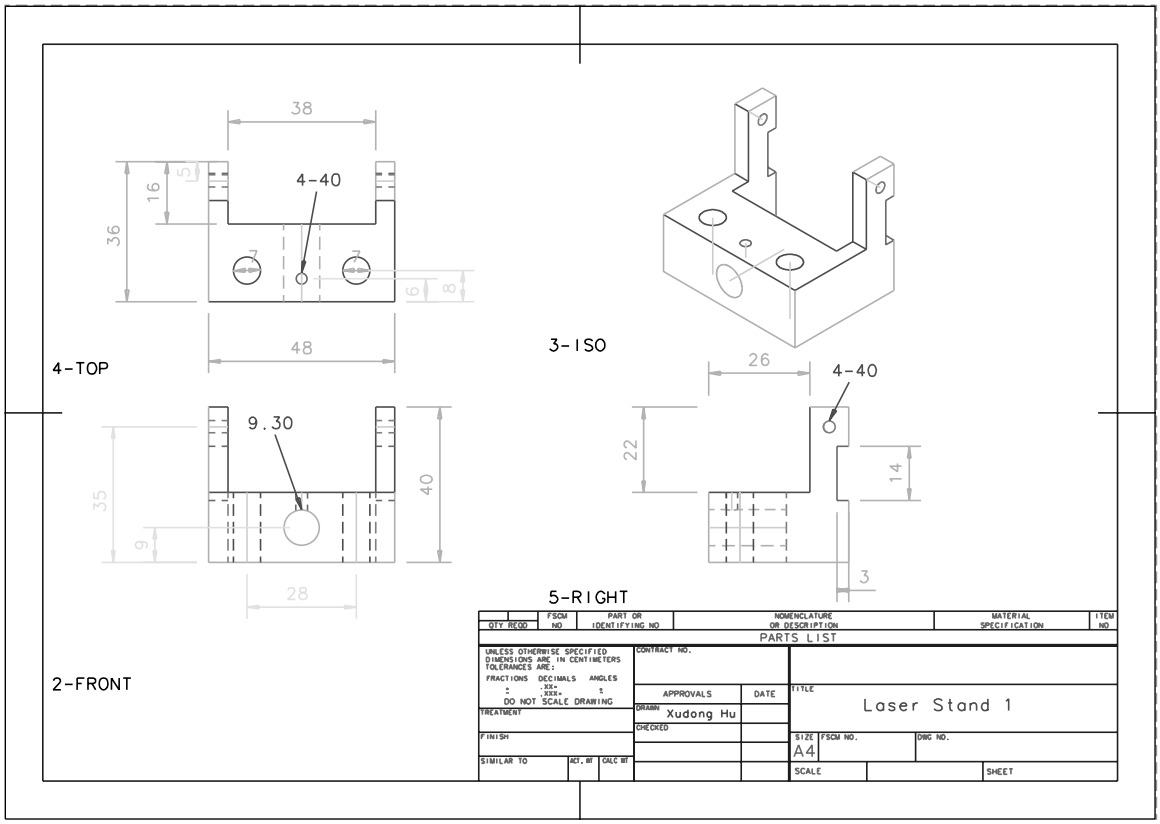


Figure A.2: Laser stand 2

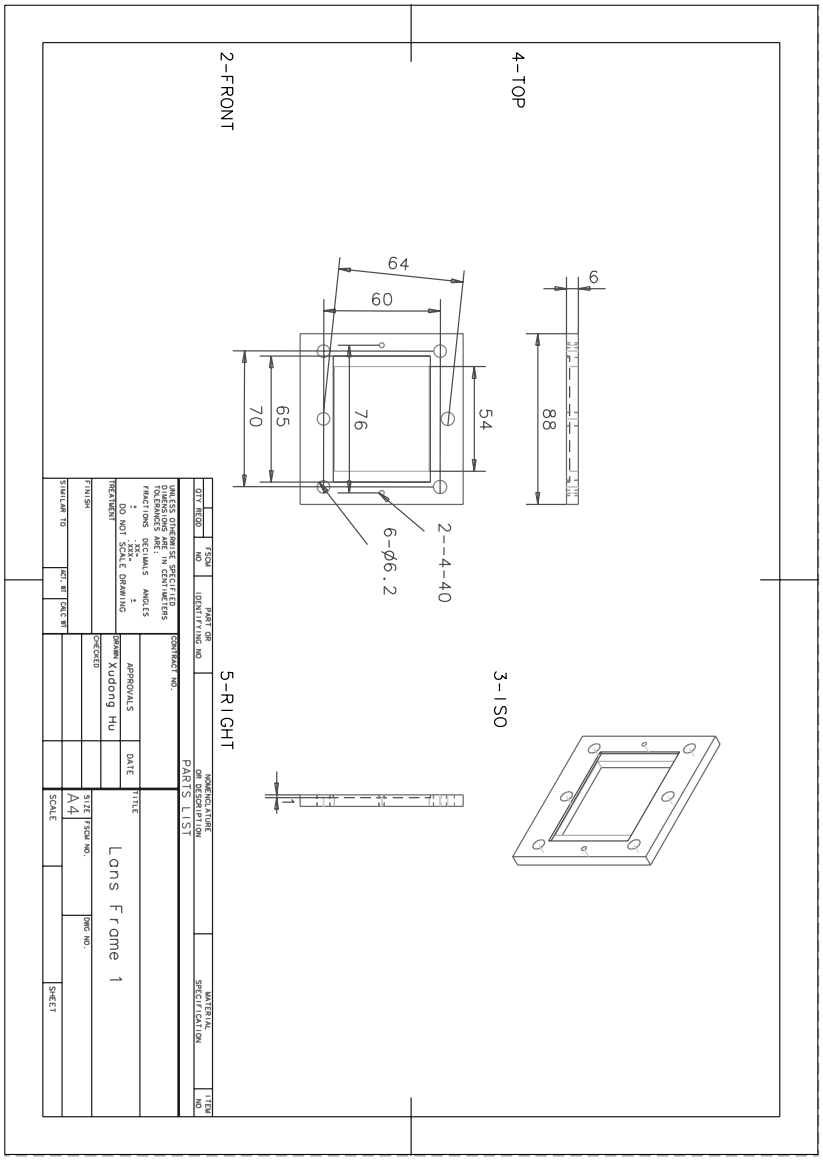


Figure A.3: lens Frame 1

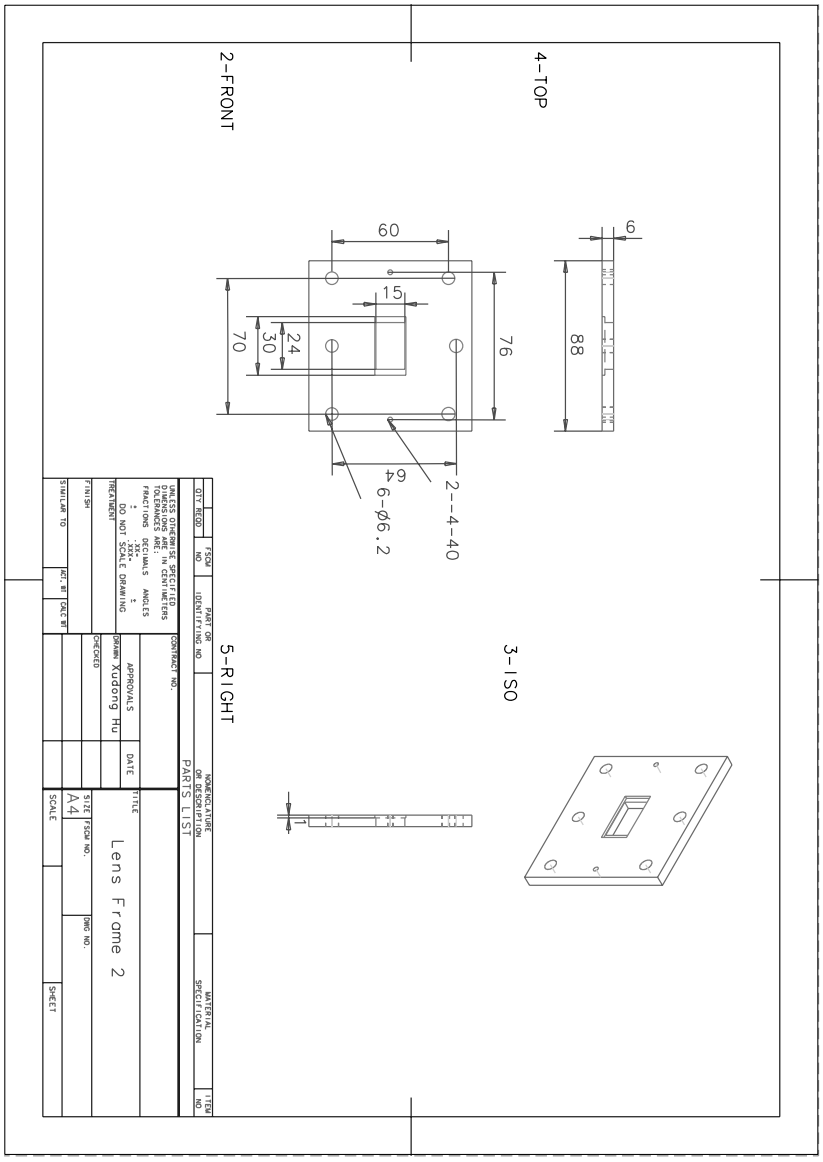


Figure A.4: lens Frame 2

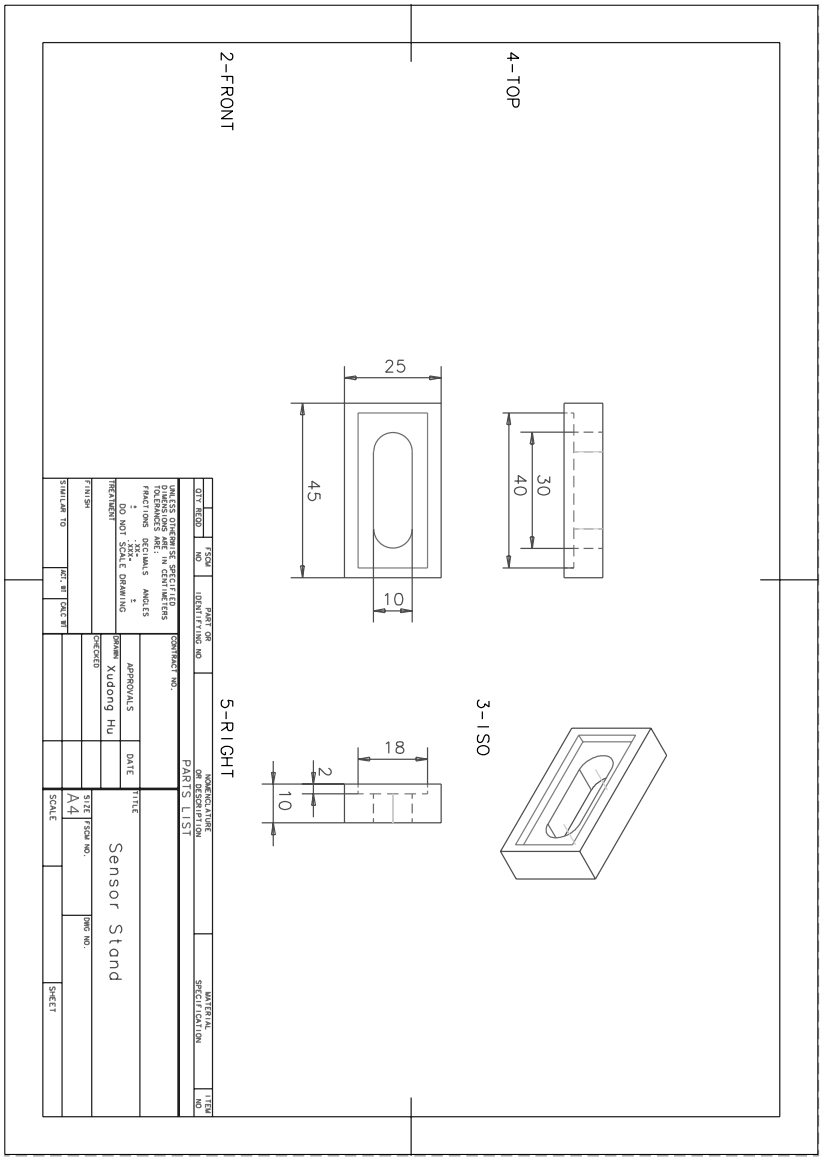


Figure A.5: Sensor mounter