

UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

OPTOELECTRONIC THREE-DIMENSIONAL TRACKING SYSTEM

FOR COLLISION RISK MODEL

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

YIH-RU HUANG
Norman, Oklahoma
2009

OPTOELECTRONIC THREE-DIMENSIONAL TRACKING SYSTEM
FOR COLLISION RISK MODEL

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

DR. JOHN FAGAN, CHAIR

DR. HILLEL J. KUMIN

DR. HONG LIU

DR. JAMES J. SLUSS

DR. JOSEPH P. HAVLICEK

© Copyright by YIH-RU HUANG 2009
All Rights Reserved.

ACKNOWLEDGEMENTS

Thank you, Dr. Fagan; I cannot finish this dissertation without your guidance and support. This is a very impressive experience in my life. Thanks for giving me this opportunity.

I would like thank everyone in the lab, especially Dr. Wen, Evan, Ben, Rix and Jacob. This research cannot be done without your help. Thanks FAA-OKC for all the supports and helps, especially Dean Alexander.

Finally, a special thank to my parents. You are always helpful whenever I need. Thanks for everything.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	IV
TABLE OF CONTENTS	V
LIST OF TABLES.....	VII
LIST OF FIGURES.....	VIII
ABSTRACT	1
CHAPTER 1 INTRODUCTION AND BACKGROUND.....	3
<i>1.1 Background.....</i>	<i>3</i>
<i>1.2 The Task.....</i>	<i>5</i>
<i>1.3 Dissertation Outline.....</i>	<i>8</i>
CHAPTER 2 CRM SYSTEM ARCHITECTURE	9
<i>2.1 CRM-Box System Implemented Architecture</i>	<i>11</i>
<i>2.2 CRM System Remote Control Server</i>	<i>26</i>
<i>2.3 CRM System Database Server</i>	<i>30</i>
CHAPTER 3 METHODOLOGY AND ALGORITHM.....	32
<i>3.1 CRM-Box System Runway Sighting.....</i>	<i>32</i>
<i>3.2 Data Acquisition</i>	<i>36</i>
<i>3.3 System Calibration.....</i>	<i>38</i>
<i>3.4 Digital Image Processing.....</i>	<i>41</i>
<i>3.5 TSPI Construction.....</i>	<i>44</i>

CHAPTER 4 RESULTS	47
4.1 CRM System Evaluation and Validation.....	47
4.2 CRM System Errors Analysis.....	51
4.3 Results.....	64
CHAPTER 5 CONCLUSIONS	67
REFERENCES	69
APPENDICES.....	72
Appendix A: CRM-Box CCD Camera Aiming Tool C# Code	72
Appendix B: CRM System Raw Data to CSV Tool C# Code.....	85
Appendix C: CRM_BoxRL_ProcessAutoCal.m MATLAB Code	88
Appendix D: CRM-Box System Watchdog C code.....	93
Appendix F: KOUN Runway 17 CRM-Box System Sighting.....	121
Appendix G: CRM Server Linux Perl Scripting Language	123

LIST OF TABLES

TABLE 2.1 CRM WATCHDOG COMMAND CODES.	18
TABLE 4.1 TSE MEAN OF THE CALIBRATION FLIGHT.	59
TABLE 4.2 STANDARD DEVIATION OF THE ABOVE CALIBRATION FLIGHT.	59
TABLE 4.3 MEAN ERROR FOR 6 CALIBRATION FLIGHTS.	60
TABLE 4.4 STANDARD DEVIATION FOR THE ABOVE 6 CALIBRATION FLIGHTS.	60
TABLE 4.5 ICAO AIRCRAFT APPROACH CATEGORY (KNOTS).	62

LIST OF FIGURES

FIGURE 1.1 TYPICAL LIS GUIDED APPROACH.	4
FIGURE 1.2 CCD VIEW OF APPROACH LANDING LIGHTS.....	7
FIGURE 2.1 ANAGLYPH APPROACHING CRM IMAGE.....	10
FIGURE 2.2 CRM SYSTEM ARCHITECTURE.....	11
FIGURE 2.3 KOUN MASTER CRM-BOX.....	12
FIGURE 2.4 CRM-BOX FUNCTION DIAGRAM.....	13
FIGURE 2.5 CRM-BOX SYSTEM COMPONENTS.....	14
FIGURE 2.6 WATCHDOG PCB IN THE CRM-BOX SYSTEM.	15
FIGURE 2.7 CRM-BOX COMPUTER POWER MANAGEMENT FLOW CHART.....	17
FIGURE 2.8 CRM BOX SOLAR CHARGING FLOW CHART.....	18
FIGURE 2.9 CRM-BOX SYSTEM COMMUNICATIONS.....	20
FIGURE 2.10 VIA C3 EMBEDDED SYSTEM COMPUTER.....	23
FIGURE 2.11 OV9121 CCD WITH 50MM LENS.	24
FIGURE 2.12 OV9121 LIGHT RESPONSE.....	25
FIGURE 2.13 IRC30 IR FILTER TRANSMISSION VALUE.....	26
FIGURE 2.14 CRM-BOX SYSTEM REMOTE CONTROL SERVER WEB.....	28
FIGURE 2.15 CRM-BOX REMOTE CONTROL SERVER WEB.....	29
FIGURE 2.16 CRM RAW DATA STRUCTURE.....	30
FIGURE 3.1 CRM BOX CCD VIEW COVERAGE.....	33
FIGURE 3.2 KOUN CRM BOXES RUNWAY SIGHTING.....	34
FIGURE 3.3 CRM SYSTEM RUNWAY SURVEY.....	35
FIGURE 3.4 LANDING LIGHT IMAGE CENTROID.....	37
FIGURE 3.5 ASHTECH Z-XTREME DGPS RECEIVERS.....	39
FIGURE 3.6 CRM CAMERA AIMING SOFTWARE.....	40
FIGURE 3.7 CRM DSP FLOW CHART.....	42
FIGURE 3.8 CRM-BOX CAMERA LEFT AND RIGHT IMAGES.....	44
FIGURE 3.9 CRM TSPI ALGORITHM ILLUSTRATION.....	46

FIGURE 4.1 ECEF-XYZ AND ENU LOCAL TANGENT PLANE.	49
FIGURE 4.2 FLIGHT TRACK ENU (LEFT) TO NORMALIZED ENU (RIGHT).	50
FIGURE 4.3 CRM SYSTEM LEFT AND RIGHT VIEW OF CCD.	52
FIGURE 4.4 CRM SYSTEM CCD AND TRUTH SYSTEM ANGLE MEASUREMENT.	54
FIGURE 4.5 CRM AND TRUTH SYSTEM HORIZONTAL AND VERTICAL TSPI PLOT.	55
FIGURE 4.6 KOUN CRM 3D TSPI PLOT.	57
FIGURE 4.7 CRM SYSTEMS TSE WITH DISTANCE EAST.	58
FIGURE 4.8 CRM-BOX SYSTEMS AND TRUTH SYSTEM DIFFERENCE IN TIME.	61
FIGURE 4.9 CRM-BOX SYSTEMS CCD RESOLUTION.	63
FIGURE 4.10 CRM TSE DISTRIBUTIONS HISTOGRAM.	66

ABSTRACT

The purpose of this dissertation is to develop a new, novel and low cost three-dimensional tracking system that can produce a Time and Space Position Information (TSPI) database for developing an FAA Collision Risk Model (CRM) for the final phase of flight. This Collision Risk Model TSPI database will then help the FAA define a better and safer Terminal En-Route Procedures (TERPS) for air-traffic control in the National Air Space (NSA). The Federal Aviation Administration (FAA) has attempted to develop a simple and economic solution to analyze the approaching aircraft's behavior during the time immediately after leaving the instrument approach to landing, in the visual segment, during Instrument Meteorological Conditions (IMC). Normally FAA could use laser or radar tracking, but it is expensive, does not acquire sufficient data for a meaningful analysis, and is hampered by the weather itself. The reason this tracking technique is being researched is that little is known of the aircraft's behavior upon leaving the instrument segment of flight in IMC conditions and transitioning to a visual form of flight to the touch down point during a landing.

The new CRM tracking system uses a pair of stationary CCD cameras to record the landing lights of the approaching aircrafts at two sides of the runway. The concept is to apply the left and right pictures from the two cameras to create a stereoscopic image. The stereoscopic images are then used to triangulate the position of the approaching aircraft, which becomes their TSPI data. Every CRM

tracking system on a runway includes two CRM tracking units and links to a central CRM data base server computer by GSM (Global System for Mobile Communications) wireless network.

The CRM system is new and novel concept and is the first successful low cost attempt to visually track a high speed approaching aircraft. The system satisfies the requirement to provide a large volume of track data on an area of the approach that had not been examined for the risk for collision of each type of aircraft on final approach after leaving IMC condition. Controller and pilot error in the critical phase of flight can be determined in order to implement new FAA procedures for the final approach to landing. This CRM tracking system has proved its functional integrity and has successfully produced high accurate TSPI data for the FAA at the University of Oklahoma Westheimer Airpark (KOUN) and Oklahoma City Will Rogers World Airport (KOKC). This dissertation discusses the details of the CRM tracking system concept, implementation, including software and hardware development. This dissertation also includes the system's function development, calibration, and system definition errors as well as a sample of the data produced by the new system.

Chapter 1 Introduction and Background

This dissertation introduces a new and novel engineering solution for tracking the aircraft on approach to landing during the visual segment of flight. Until now, the Collision Risk Model (CRM) provides approach obstacle clearance risk analysis before the aircraft reaches the Decision Height (DH) which is a part of approach by instrument flight. The contribution of this dissertation is the development of a system for the Federal Aviation Administration (FAA) with a methodology to complete the FAA's Collision Risk Model (CRM) by finishing the visual segment of the approach to the landing. This research builds on previous work which developed a partial flight tracking system. The research re-directs an ongoing research project by developing a new, compact, and unique image tracking algorithm and hardware host for tracking aircraft on approach in the visual segment. This dissertation will detail the Collision Risk Model (CRM) tracking system, which includes software and hardware algorithm development and implementation and how the system harvested data will complete the FAA's current CRM. [1] The research will also detail the algorithm functional development, calibration measure, and a definition of the system error.

1.1 Background

The Federal Aviation Administration (FAA) has attempted to develop a simple and economic solution to analyze the approaching aircraft's behavior

during the time immediately after leaving the instrument approach to landing, in the visual segment, during Instrument Meteorological Conditions (IMC). Usually, the FAA could use laser or radar tracking system but it is much more expensive and does not acquire enough CRM data on a wide variety of aircraft and airport environment, and is hampered by the weather itself. The reason this tracking technique is being developed is that little is known of the aircraft's behavior upon leaving the instrument segment of flight in IMC conditions and transitioning to a visual form of flight to the touch down point during an approach to landing.

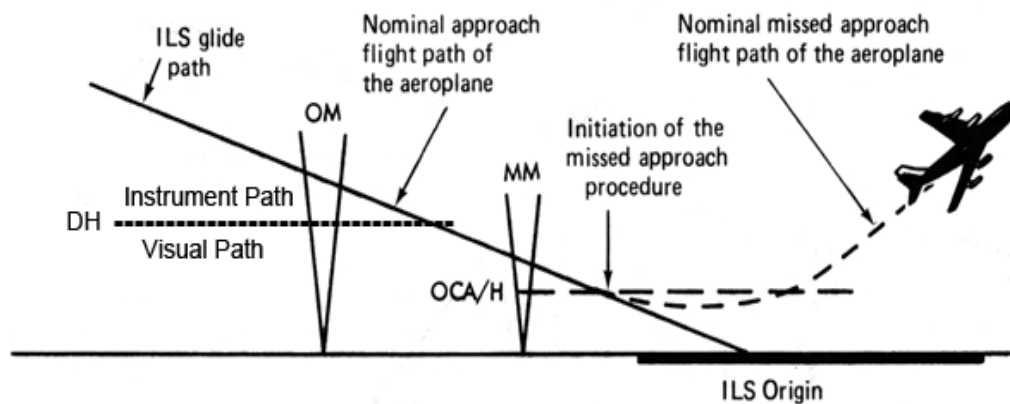


Figure 1.1 Typical ILS Guided Approach.

This dissertation develops a three-dimensional tracking system to produce a Time and Space Position Information (TSPI) database for Collision Risk Model (CRM) analysis. This Collision Risk Model TSPI database will help the FAA to define better and safer Terminal En-Route Procedures (TERPS) for air-traffic

control in the airspace service volume. Figure 1.1 is a typical approach path for an airplane on final approach of flight to landing. ILS and DH represent Instrument Landing System and Decision Height respectively. ILS is a ground based navigation system that provides the localizer and glide-slope signal for the aircraft during an approach. The localizer signal is an outward horizontal beam which helps align the aircraft in the horizontal plane with the centerline of approaching runway. The glide-slope signal is a vertical beam that helps keep the aircraft on an optimal vertical descent path. The DH is an altitude above the ground level on which a pilot makes a decision whether continue approach to landing or perform a missed approach. In order to continue the approach, the pilot must be able to see the runway environment. OM and MM represent the Outer Marker and Middle Marker beacons respectively. The Outer Marker and Middle Marker are aligned with approach line in front of a runway and support additional information about the distance to the runway threshold. The standard instrument approach is performed using ILS or GPS before the aircraft reaches the Decision Height, and after the Decision Height is the visual segment in which the pilot completes the approach to landing or the missed approach in a manual mode.

[1]

1.2 The Task

The concept of this dissertation revolves around developing a pair of CRM-Box systems, located at the left and right sides of the subject runway, to

capture a stereoscopic image of an approaching airplane's landing light. The horizontal angle from the left and right landing light images can determine the 2D (East and North) horizontal distances of the approaching aircraft. Adding the vertical angle measurement from either left or right images with the horizontal distance can be used to calculate the altitude (Up) of the approaching aircraft. Merging 2D horizontal distances and vertical altitude continuously results in a three-dimensional East, North, Up (ENU) TSPI track for the CRM visual segment. The CRM visual segment research is a marriage of the wisdom of the past tracking concepts with new technology available today.

The work presented in this document solves two major problems encountered in tracking the visual segment. The challenge of developing a system platform is to design a completely independent and reliable system, where these units require no airport infrastructure. The system must be minimally invasive to the airport environment and not interfere with the mission of the airport. The hardware of the CRM system must have a watchdog power management system and the solar charging system to collect and store energy to overcome no sun during the operational place. The watchdog system must guarantee the integrity and reliability of the CRM-Box system energy source and remote control by the CRM server at all times.

There is no current equipment research on tracking multiple identical landing lights in such large scale at such a low cost. The only similar science is

calculating the astronomical unit (AU) in the early age. [2] Therefore, it is necessary to create a solution for an automatic tracking algorithm. Figure 1.2 shows a compost set of left and right approach track of more than one aircraft's approach landing lights and ambient city light recorded by the CRM-Boxes' CCD cameras.

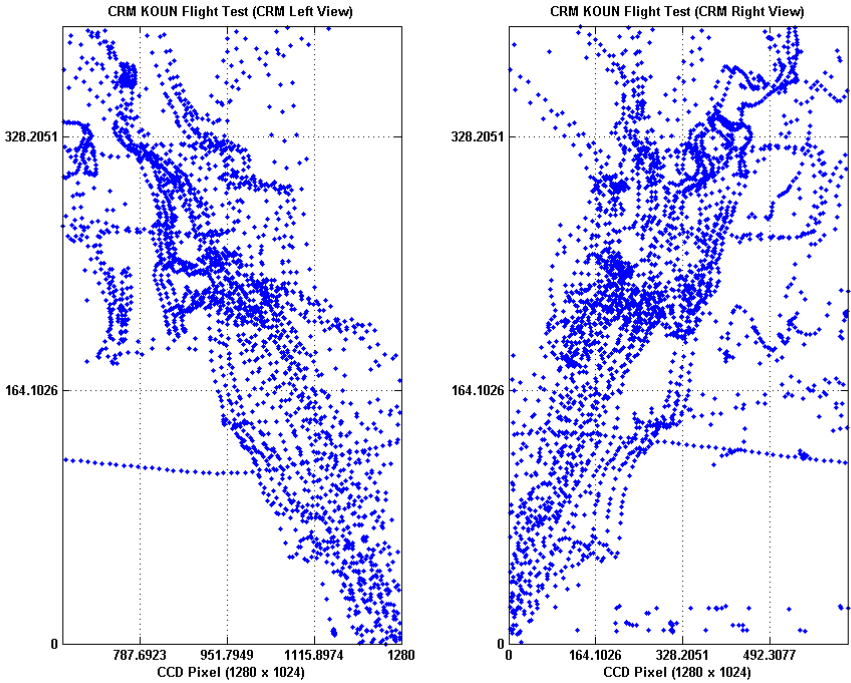


Figure 1.2 CCD View of Approach Landing Lights.

Each blue dot is represents a snapshot of the landing light in time. The CRM left and right binary images are the combined one night's capture of the approaching aircrafts' landing lights at KOUN. The goal of this research is to develop a unique algorithm that creates a stereoscopic image from the left and

right pictures that triangulates the approaching aircraft's three dimension positions for its TSPI data. The problem is also to develop a stable algorithm that matches the left and right landing lights images on a multiple image environment which indicates the same approaching aircraft and produces a 3D Time and Space Position Information (TSPI) database for Collision Risk Model (CRM) analysis. [3]

1.3 Dissertation Outline

Chapter 1 presents the purpose and concept of a CRM tracking system and points out the dissertation's contributions, background, and a path to the solution of the problem. Chapter 2 describes the conceptual system function and architecture. Chapter 3 presents the methodology and the developed algorithms for an effective solution to tracking multiple aircraft for Collision Risk Model (CRM) analysis. Chapter 4 presents the CRM system analysis and evaluation of the CRM system error. Chapter 5 presents the conclusions and foreseeable future research.

Chapter 2 CRM System Architecture

To achieve the final part of the visual segment of the CRM research, a complete data acquisition system was developed, which relies on a unique and robust system architecture. The CRM system has three major components which include the CRM-Box systems on the airport, the CRM system remote control server, and the CRM database server. These three components of the CRM system form a powerful network to collect and sort the CRM data automatically in the database for FAA CRM analysis.

The function of a pair of CRM-Box systems is to capture the stereoscopic images from an airplane's landing light using a self provided energy source and a pair of wireless links to the CRM server. A pair of CRM-Box systems is placed at the left and right side of the runway. The CRM-Box systems use the horizontal phase shift angle to determine the distance between the runway threshold and an approaching aircraft's horizontal position. Figure 2.1 shows the anaglyph approaching images taken from the CRM-Box system at KOKC. The anaglyph image builds from a single approaching aircraft with two horizontal phase shift angle images. The vertical angle measurement and the horizontal distance are used to calculate the altitude of the approaching aircraft. With horizontal position and vertical altitude, the CRM system can build an East, North, Up (ENU) TSPI database for this approaching aircraft.



Figure 2.1 Anaglyph Approaching CRM Image.

The CRM system remote control server is a web based network remote control system. The remote server acquires the necessary information such as weather conditions at each airport and the CRM-Box system's health status to determine if the CRM-Box system needs to be powered on or off. The purpose of the CRM system remote control server is to reduce the CRM-Box's system power consumption and only acquire the CRM data during Instrument Meteorological Conditions (IMC). The CRM database server organizes the CRM data and sorts CRM byte package by date and airport. Figure 2.2 shows the CRM architecture diagram with the three major components.

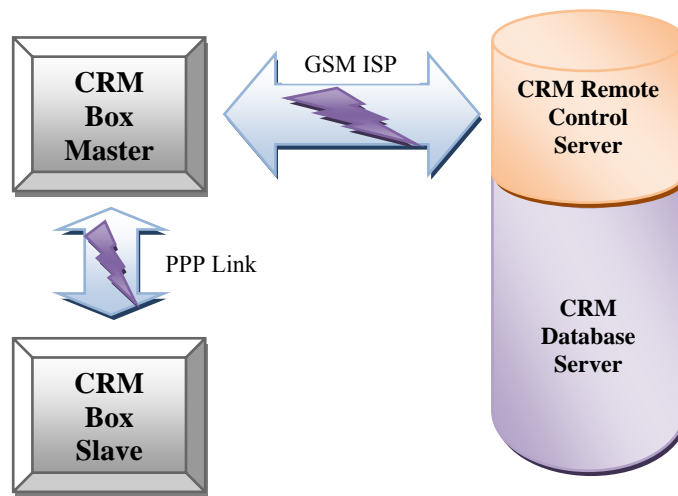


Figure 2.2 CRM System Architecture.

2.1 CRM-Box System Implemented Architecture

The CRM-Box system chassis is built in a 30"x24"x12" industrial enclosure. One side of the box has a glass window for observation and an aluminum frame structure on the top for holding a solar panel. Each box was designed with aluminum panel that is fixed to the bottom of the box for placing the electronic components and batteries. Each CRM-Box system is equipped with a CRM watchdog board developed as part of this research, a 3.5" VIA-C3 Eden mini-board computer with BusyBox Linux Embedded System, a charge coupled device (CCD) with 50mm lens and IR filter, a patch directional antenna, a lightning surge arrester, three 12-volt batteries, and one 220-watt solar panel module. Figure 2.3 shows the master CRM-Box system at right side of runway 17 at the University of Oklahoma Max Westheimer Airport (KOUN).



Figure 2.3 KOUN Master CRM-Box.

The master and slave wireless communication interface and solar charging controller are also integrated into the CRM watchdog PCB. The master CRM-Box system has an additional GSM cell phone for wireless internet data link to connect to the CRM remote control and database server. The slave CRM-Box system was developed to use a local wireless link to the master CRM-Box system and share the internet service provide by the master CRM unit. Figure 2.4 shows the CRM-Box system function diagram and figure 2.5 shows the components inside the CRM-Box system.

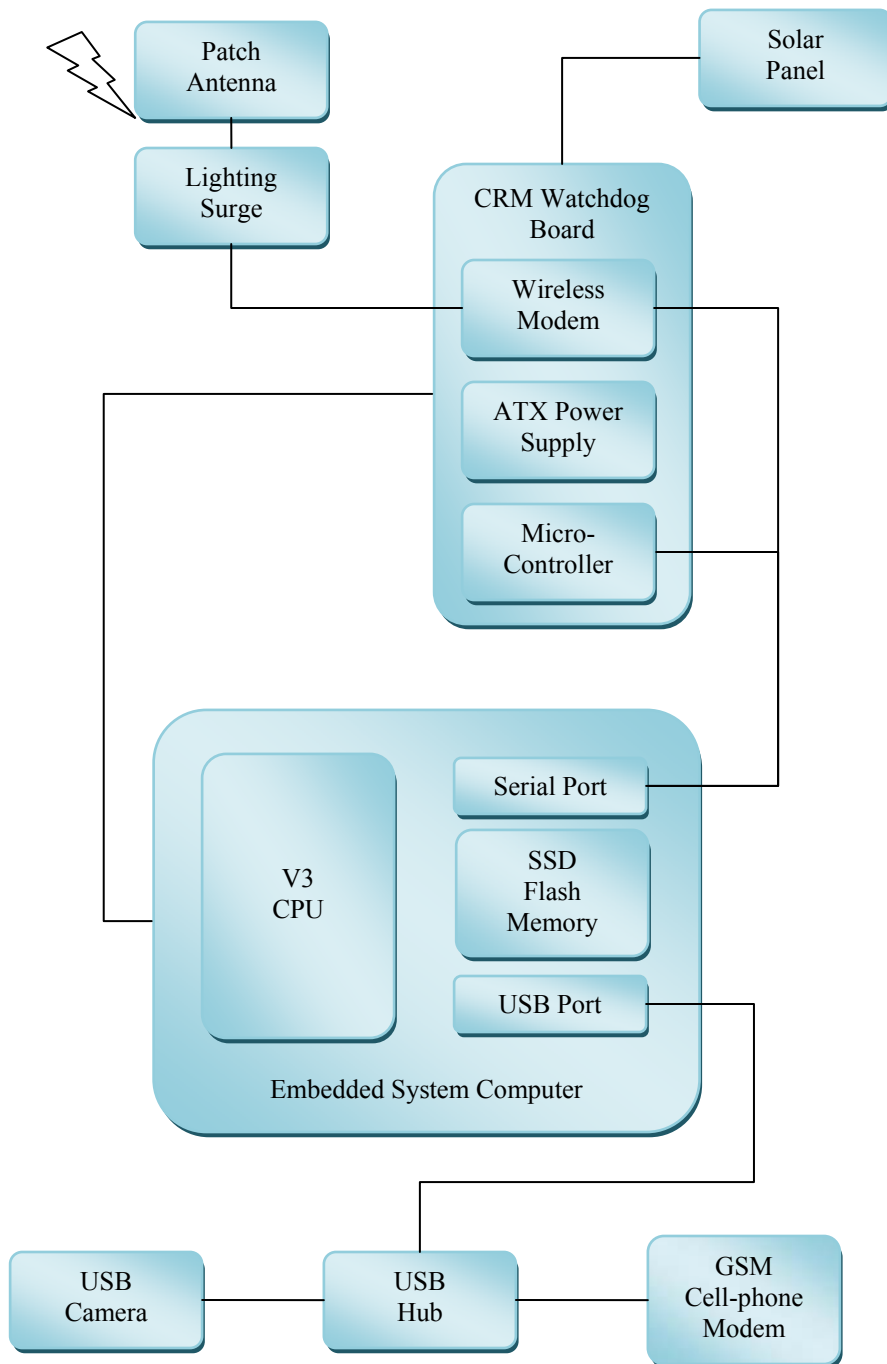


Figure 2.4 CRM-Box Function Diagram.

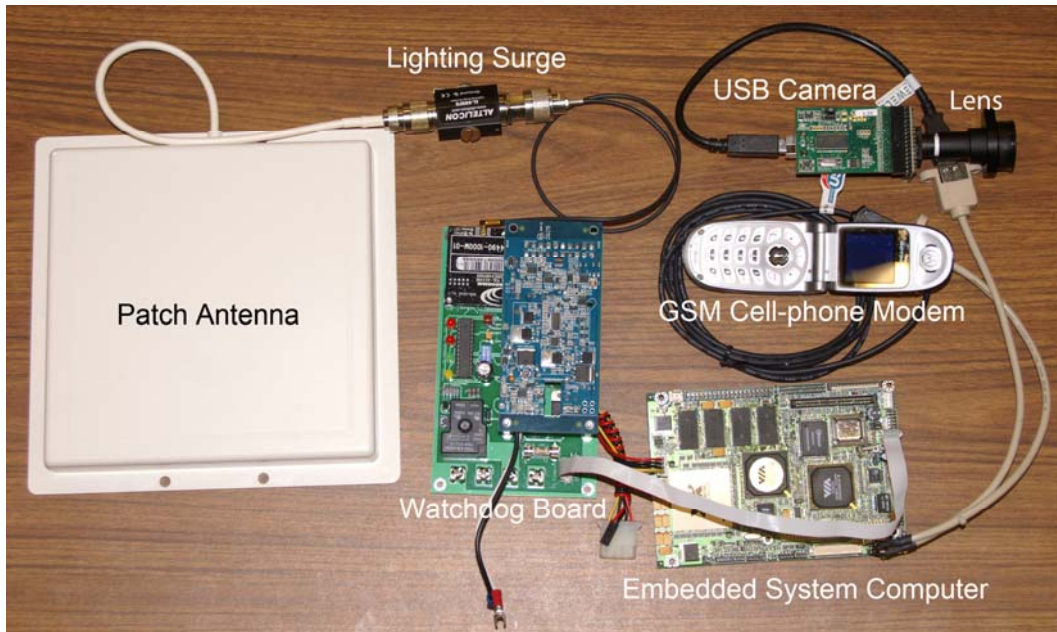


Figure 2.5 CRM-Box System Components.

2.1.1 System Watchdog and Solar Charging Controller

A system watchdog was required to solve continuity and stability issues of the CRM-Box system hardware. The CRM watchdog board includes a CRM watchdog microcontroller system, CRM-Box master to slave communication system, an ATX DC-DC converter power supply unit, and the system solar charging management system. The heart of the CRM-Box system watchdog is an ATmel ATmega8 microcontroller. The ATmega8L is a high-performance AVR 8-bit RISC architecture microcontroller with operating voltages between 2.7 and 5.5 V. When the ATmega8L is active, it only requires 3.6 mA to run and 1 mA at idle mode. The ATmega8L is capable of 1 to 8 MIPS with byte-oriented two-wire serial interface designed for useful debug or information display. The

ATmega8L can be programmed by C code and compiled by the AVR-GCC (GNU C Compiler) with the powerful AVR Library. [4] The microcontroller manages the system power level and solar charging to the batteries inside the CRM-Box system. Figure 2.6 shows the watchdog layout in the CRM-Box system.

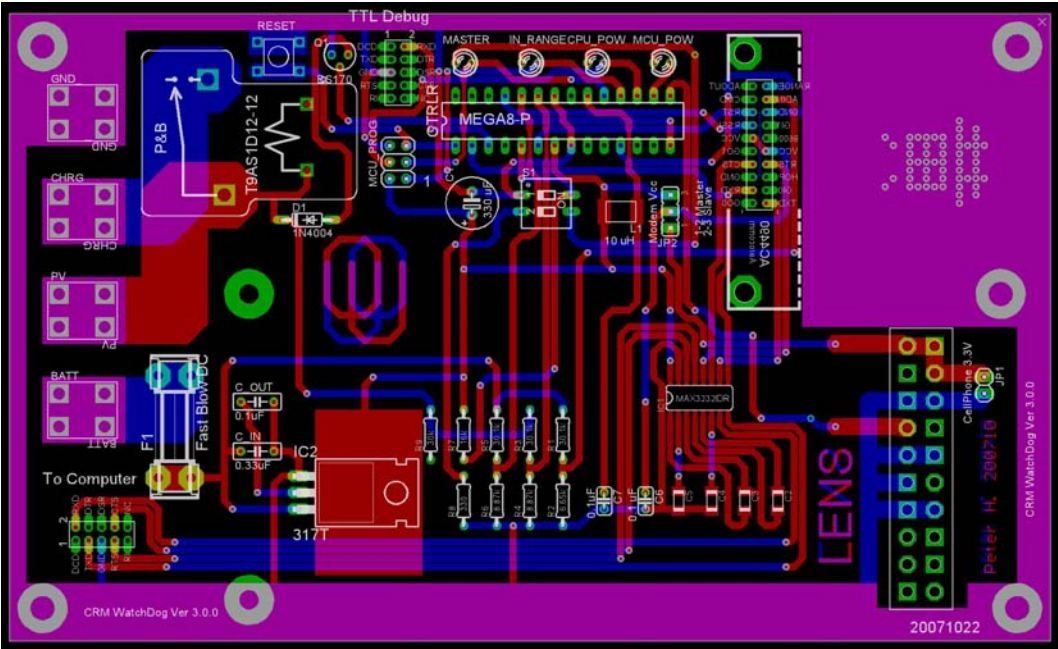


Figure 2.6 Watchdog PCB in the CRM-Box System.

The power management of the CRM-Box computer is done by sleep time control from the CRM remote server and the condition at local CRM-Box system. When the system is powered on for the first time, the microcontroller will determine the local CRM-Box conditions, including power level and master CRM-Box online status. After the status checks, the CRM-Box watchdog microcontroller will power on the CRM-Box computer and wait six minutes allowing the computer to boot up and connect to the CRM remote control server

through the GSM cell phone ISP. After six minutes of no response from the CRM-Box computer, the CRM-Box watchdog will then force the computer to sleep for another three minutes by shutting down the power of the CRM-Box computer and then powering on again. This routine is in order to force the power cycle of the CRM Embedded System by directly cutoff the computer's main power supply. If the CRM-Box computer successfully boots and connects to the CRM remote control server, the server will tell the CRM-Box watchdog to stay powered on or sleep for a precise time and then power the computer off. This routine is set by an alarm clock for the CRM-Box Computer to wake up again and stay in contact with CRM remote control server. The reason for this routine is after the CRM-Box computer is been shutdown by the order of the CRM remote server, the watchdog microcontroller in the CRM-Box system could not directly connect to the CRM remote server without the TCP-IP protocol function. Figure 2.7 shows the system power management flow chart of the CRM-Box system.

The Timeout Refresh Code is sent often to indicate the CRM-Box computer status and determine if the CPU is still alive or CRM-Box system is functioning normally. If the CRM-Box computer is running normally and not halted for some reason, the CRM-Box computer should send out the Timeout Refresh Code (DE AD FA CE EE) to the CRM-Box system watchdog through a one-way RS-232 serial communication port. The Timeout Refresh Code should be sent often and for a period not longer than 360 seconds. Any task software

runs in the CRM-Box embedded system crash or in a wait dead loop will cause the computer not send out this message code. Table 2.1 shows the typical CRM-Box computer to watchdog command codes. The DE AD FA CE is the header for CRM-Box system command codes. The inclusion of the header code in CRM-Box system commands reduces the chance of miscommunication between CRM data and CRM commands. The header code can avoid triggering the sleep or power off code accidentally from the CRM raw data that is sent through the same serial port at a different baud rate.

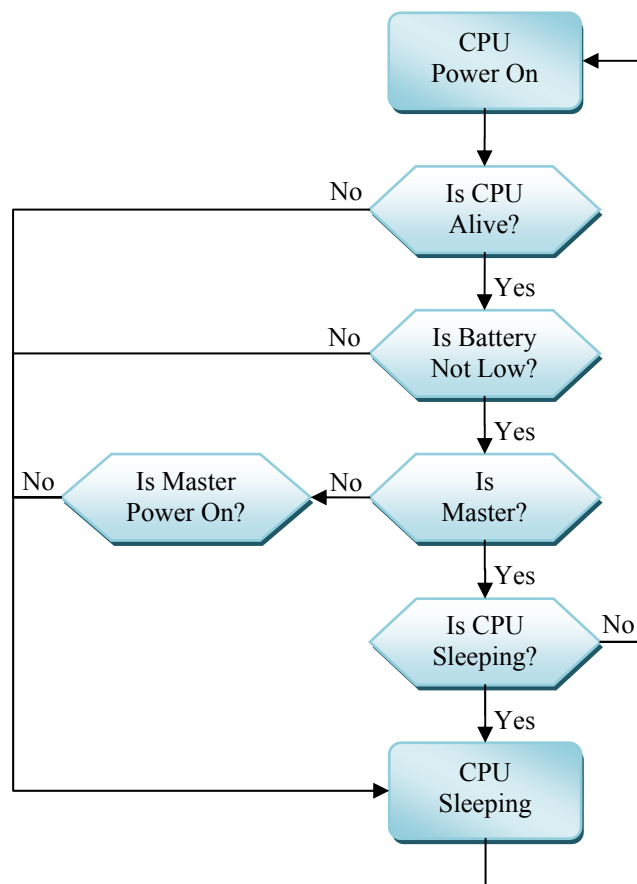


Figure 2.7 CRM-Box Computer Power Management Flow Chart.

DE AD FA CE 55	Power Off Command Code
DE AD FA CE EE	Timeout Refresh Code
DE AD FA CE AA XX	Sleep XX sec Command Code

Table 2.1 CRM Watchdog Command Codes.

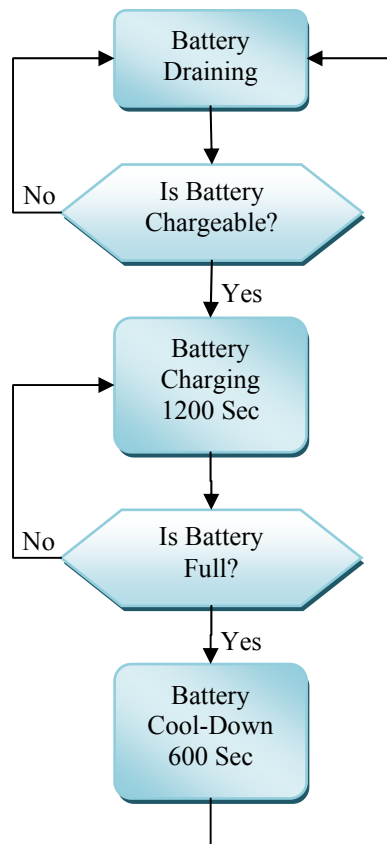


Figure 2.8 CRM Box Solar Charging Flow Chart.

The purpose of the solar charging management of the CRM-Box system is to optimize the power from the solar array as well as preventing the batteries from over charged. Any over charge would raise the battery's temperature, reducing

the battery's life and energy capacity. The watchdog microcontroller will check if the batteries need to be charged and compare the solar voltage level and battery voltage level. When the batteries need to be charged and the battery voltage is less than solar voltage, the watchdog microcontroller will switch the system to charge mode. Next, the watchdog microcontroller will check battery voltage level every twenty minutes until the batteries are completely charged and follow with a ten minute battery cool down cycle. The battery cool down cycle allows the voltage level to be measured correctly by the CRM system watchdog after one complete charge cycle. Figure 2.8 shows the CRM-Box solar charging flow chart.

2.1.2 Master and Slave Wireless Communication

There are two stages of communication between the master and slave CRM-Boxes system and the CRM-Box system to the CRM database server. The communications include the command codes to the CRM-Box system watchdog and the CRM track data with system status information to the CRM database server. These two types of information will transmit data by sharing one serial communication port due to CRM-Box computer hardware limits. A new technique will switch dual baud rates to separate these two types of transmissions. The baud rates are 1200 bps (bits per second) and 38400 bps with consideration for electric signal separation, minimum data transmitting speed needed, and wireless communication bit error rate (BER).

Both CRM master and slave box systems need to be connected to the CRM server through the internet in order to transmit the CRM data back or receive CRM remote control commands. Cost consideration require, the master CRM-Box system to share the internet with the slave CRM-Box system through WLAN (Wireless Local Area Network) with PPP (Point-to-Point Protocol) link encap which construct on the serial ports between both master and slave CRM-Box system computer. Figure 2.9 shows the two types of CRM-Box communications previously mentioned with GSM cell phones, internet, and private wireless link. The wireless internet on the master CRM-Box system is established on a GSM cell phone based Internet Service Provider (ISP) and shares the network IP through private network protocol. The CRM-Box private network uses a 24-bit block (10.0.0.0-10.255.255.255) single class A protocol which has an IP address of 10.0.0.1 to the master CRM-Box system and another IP address of 10.0.0.2 assigned to the slave CRM-Box system.

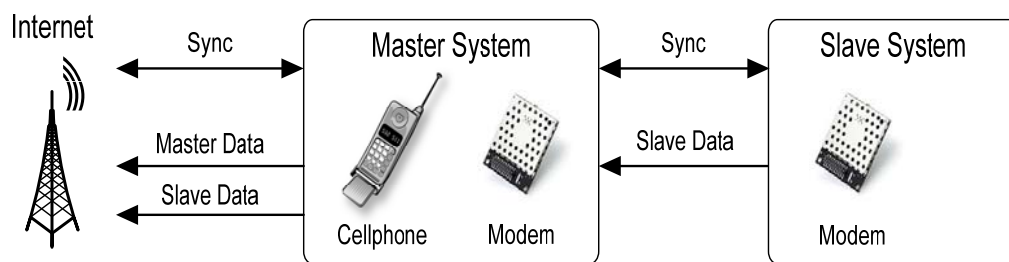


Figure 2.9 CRM-Box System Communications.

The wireless PPP communications between the master CRM-Box system and slave CRM-Box system utilizes a pair of Aerocomm AC4490x 900 MHz transceivers with a pair of HyperLink Technologies HG908P flat patch antennas. The AC4490 is a low cost low power wireless transmitter and the patch antenna will improve the gain of the wireless signal by its directional radio wave character. The Aerocomm AC4490 is capable with full handshaking serial communication wireless modem to ensure transmission quality. The AC4490 is a frequency hopping spread spectrum (FHSS) wireless transceiver with very low power consumption at 200 milliwatt for typical battery powered implementations such as the CRM-Box system. The AC4490 is a very low latency and high throughput Original Equipment Manufacturer (OEM) package with qualified industrial temperatures. The HG908P antenna is an 8.5 inch square flat patch antenna which has a 50 Ohm impedance and 8 dBi gain at a frequency around 902-928 MHz. This patch antenna has a 75 degrees in horizontal beam width and 65 degrees vertical beam width. The HG908P antenna is perfectly suitable for the CRM outdoor airport environment because of its rugged and weatherproof construction features, sealed internal elements, and an aesthetic UV-stable, UL flame rated white plastic radome.

2.1.3 BusyBox Linux Embedded System

The concept of the CRM visual segment research is based on optically recording the approach aircraft's landing lights and calculating the aircraft's 3D

Time and Space Position Information. By using the TSPI database, the data can be used for studying the airplane's behavior during Instrument Meteorological Conditions (IMC). This goal requires a powerful computer to digitize the optical images in order to translate the images into a set of meaningful TSPI data. The COMMELL LE-362E6 VIA-C3 computer takes on this task with a miniature BusyBox Linux embedded system. [5] Figure 2.10 shows a picture of a VIA-C3 computer used inside a CRM-Box system. The computer is a COMMELL VIA C3 Eden architecture CPU board which uses less than 20 watts in the full computing mode. The BusyBox Linux embedded system is running from a compact flash memory card as a SSD (Solid-State Drive) instead of an unreliable mechanical hard drive. The computer also has two Universal Serial Bus (USB) 2.0 ports for image acquisition.

There are two kinds of communication between the master and slave CRM-Boxes system as mentioned in prior paragraph. One is the CRM data acquired from slave CRM-Box system and needs to be transmitted to CRM database server through a master CRM-Box system's GSM internet service. The other communication is the CRM-Box embedded system to watchdog command codes communications, which includes CPU status, powering down the CRM-Box computer, and setting the sleep timer. The two communications share the only serial port on the VIA-C3 computer board. In order to filter out the CRM command code message and CRM image data to different targets, the CRM-Box

embedded system switches between two baud rates. The embedded system uses a baud rate of 38400 bps to communicate between both master and slave CRM-Box systems. When the CRM-Box computer needs to transmit the CRM-Box command codes to the CRM-Box system watchdog to update the power management status, such as keep awake or sleep for certain time, the CRM-Box embedded system switches to a baud rate of 1200 bps. The watchdog serial receiving channel pin or Rx port will ignore the 38400 bps baud rate transmission of the CRM master and slave communications.

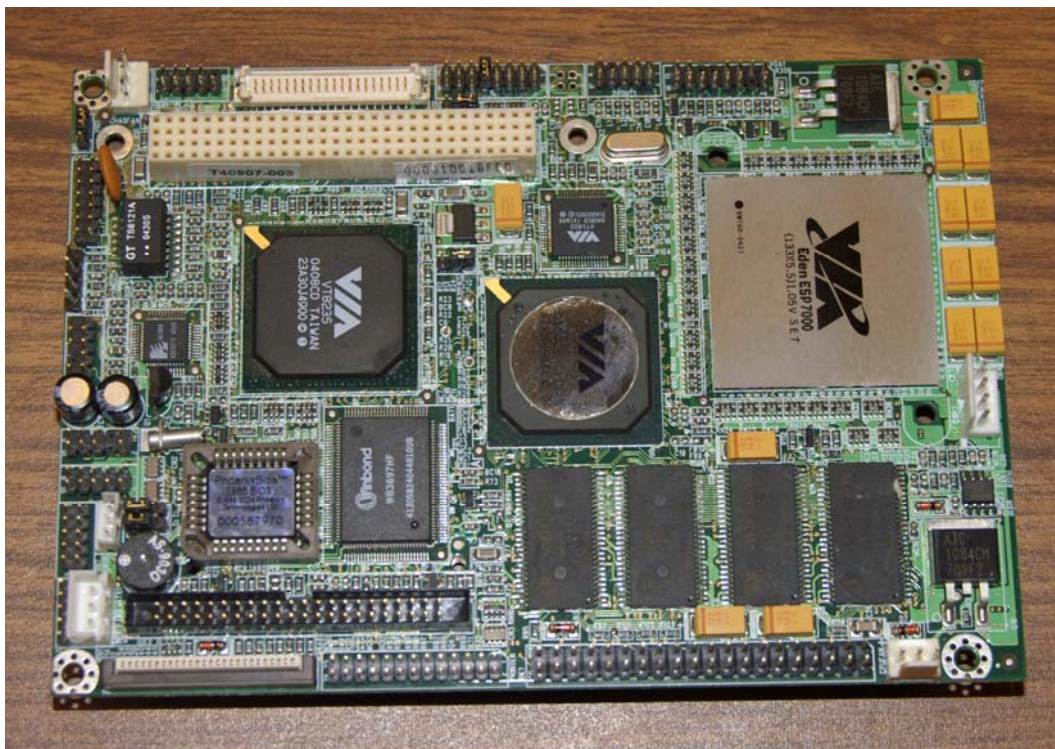


Figure 2.10 VIA C3 Embedded System Computer.

To capture the image from the approaching aircraft's landing lights, the VIA C3 computer needs to be connected to an optoelectronic device. The OmmVision OV9121 USB CCD camera is the optoelectronic device which connects through the CRM computer's USB 2.0 port. The OV9121 USB camera is a one mega-pixel black and white CCD which has an array size up to 1280×1024 (SXGA) pixels. The pixel size is 5.2 μm × 5.2 μm and has an image area of 6.66 mm × 5.32 mm (1/4" CCD). The CCD image is focused by a 50 mm telescope lens (V-4350) made by Marshall Electronics, Inc. The V-4350 50 mm telescope lens is an ideal lens for capturing the approaching aircraft's landing light in the far field airport environment because of its angle of view at 04-03-05 (H-V-D Degree) when applied to a 1/4" CCD. Figure 2.11 shows the OV9121 CCD with V-4350 50 mm lens.



Figure 2.11 OV9121 CCD with 50mm Lens.

The OV9121 CCD has wide range of light response photon sensor and can detect wavelength from 400 nm to 1150 nm. Since the visible light is about 400 nm-700 nm, the CCD can also detect the Infrared (IR) radiation. Figure 2.12

shows the OV9121 Light Response between wavelength and corresponding efficiency. However, the CCD infrared capable part is a down side of this tracking system. Because the most aircraft landing light uses halogen lamp, the landing light is also a strong IR source. The IR causes the image gets too bloom when the aircraft is close to the CRM-Box system. The new designed lens is equipped an IR cut-off filter to reduce the bloomed image from the aircraft's landing light. The IR cut-off filter also reduces the digital image processing load and helps in the location of the aircraft's landing light for the CRM system. Figure 2.13 shows the IR cut-off filter IRC30 transmission value.

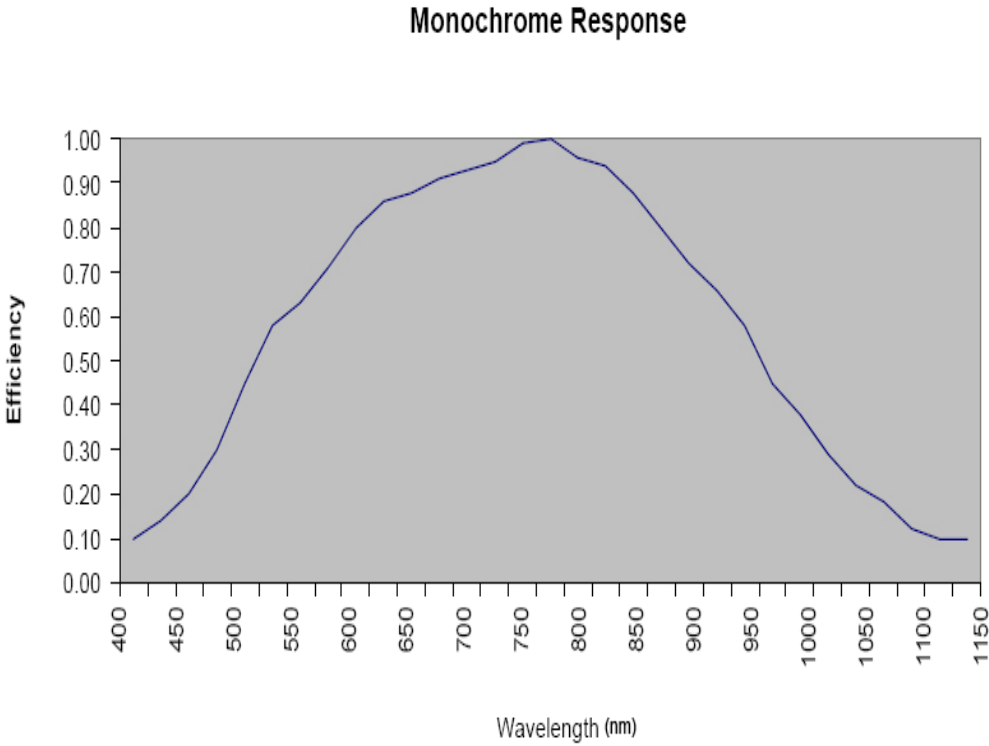


Figure 2.12 OV9121 Light Response.

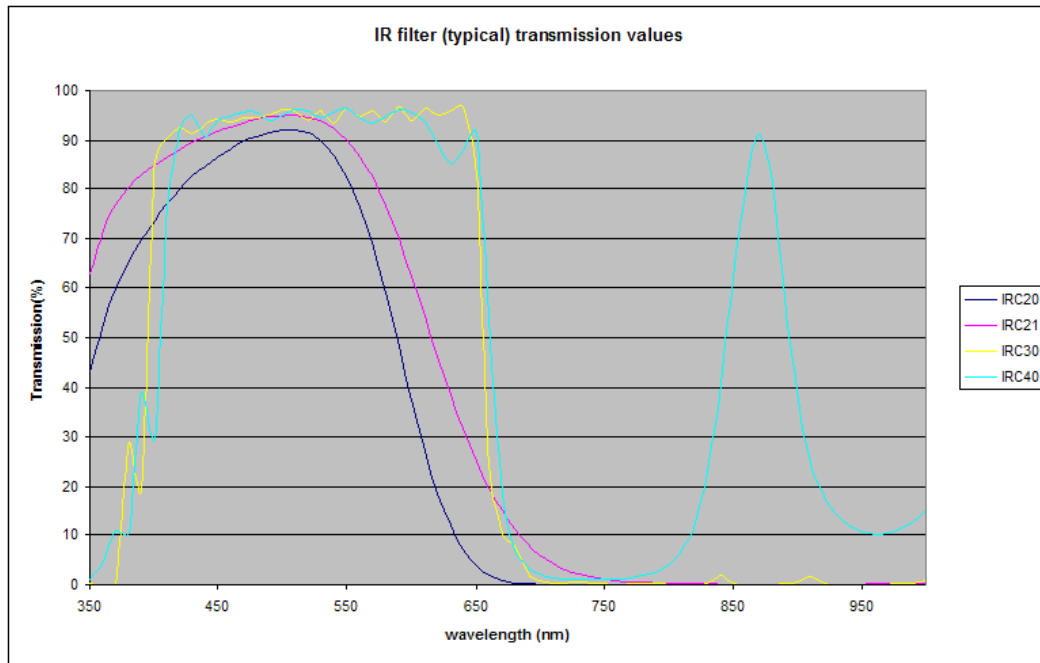


Figure 2.13 IRC30 IR Filter Transmission Value.

2.2 CRM System Remote Control Server

There are several airports in the United States that have been chosen to equip with the CRM-Box systems for the visual segment CRM study. In order to keep these CRM-Box systems working properly during Instrument Meteorological Conditions (IMC), a central CRM-Box system remote control server is required. The CRM system remote control server is a Linux Based A-M-P (Apache, MySQL, PHP) web and database server. The Linux server has a cron job set up for a scheduler of every fifteen minutes to run the weather PERL script, “/home/ben/src/logger/weather.pl”. [6]

The PERL is a scripting programming language that was originally developed in UNIX systems for a high-level, general-purpose, interpreted, dynamic programming language. PERL is a very powerful programming language in manipulating text that borrows many features from C, shell scripting (sh), AWK, sed and Lisp. [7] PERL also wins the nickname of "the Swiss Army knife of programming languages" because of its multi-functions and flexibility. The CRM remote control and database server runs PERL repeatedly in scan, search, insert, assembly, sort, and producing texts. For example, PERL can create a web address by inserting the assembly airport ID, build a database by scan, search and sort from a webpage or text file, and produce a webpage from assembly texts or database. [8]

The weather PERL script automatically collects the weather information from a National Oceanic and Atmospheric Administration (NOAA) website by using a unique NOAA web address format which contains the entire CRM subject airport ID (KXXX), such as KOUN, KOKC or KLAX. The following is an example address.

```
"http://adds.aviationweather.noaa.gov/metars/index.php?station_ids=KATL%20KBOS%20KCVG%20KLAX%20KPDY%20KSFO%20KOKC%20KBED%20KHOU%20KMEM%20KOUN%20KSEA&std_trans=standard&chk_metars=on&chk_tafs=off"
```

The special query of all CRM subject airport weather information will be displayed on this special NOAA website. On the webpage, NOAA provides Aviation Digital Data Service (ADDS) in METeorological Aviation Report (METAR) format and its translated information for each request airport. The weather PERL script then sorts the data into a Linux MySQL database and produces a PHP webpage to display the CRM-Box system status. Figure 2.14 shows a sample of the CRM-Box status webpage that automatic generate by PERL on the CRM-Box system remote control server.

CRM Box Status: Jul-16 23:00:25

id	check (1)	check (2)	status	sleep	weather age	weather
KATL	-	-	-	60.0 min.	90206	FEW 6000 SCT 6000
KBED	-	-	-	0.0 min.	1216249226	
KBOS	-	-	-	60.0 min.	19408	BKN 7000 FEW 7000 SCT 30000
KCVG	-	-	-	30.0 min.	287249	BKN 4000 SCT 2500
KHOU	-	-	online		95070	BKN 25000 FEW 2900 OVC 2000 SCT 4200
KLAX	-	-	online		19951	BKN 1200 OVC 1000 SCT 20000
KLAY	14	32	online		19952	BKN 1200 OVC 1000 SCT 20000
KMEN	-	-	-	110.0 min.	19713	
KOKC	100970	-	-	30.0 min.	6273	BKN 5000 SCT 10000
KOUN	1272	21121	-	60.0 min.	OVERRIDE.	
KPDX	6607906	6613203	online		19235	BKN 2000 OVC 2000
KSEA	187	-	online		22356	FEW 5000 OVC 1000
KSFO	21	242	online		19717	BKN 1100 FEW 1300 OVC 1200 SCT 1400

Override weather conditions:

KATL

Override disable:

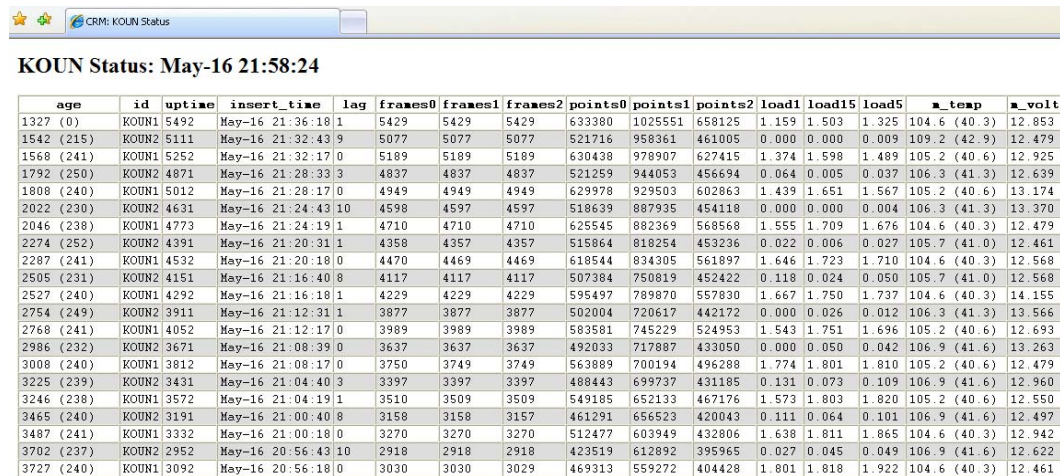
KATL

View Statistics:

KATL

Figure 2.14 CRM-Box System Remote Control Server Web.

The weather PERL scripted Common Gateway Interface (CGI) webpage can automatically turn on or off the CRM-Box systems in the field according to their airport weather condition. If necessary, the CRM-Box systems can also turn on or set the sleeping time manually from the webpage as an override function. The CRM-Box system status web page also provides the CRM-Box systems' check in age, sleeping time setup, weather check in age and weather information. By selecting the airport ID label, that airport CRM-Box system's health condition such as uptime, system load, system temperature, and voltage will display on the web page. Figure 2.15 shows the KOUN CRM-Box system's status webpage that is automatically generated by PERL on the CRM-Box system remote control server.



The screenshot shows a browser window with the title "CRM: KOUN Status". Below the title bar, the page content displays "KOUN Status: May-16 21:58:24" followed by a table of system metrics. The table has 16 columns: age, id, uptime, insert_time, lag, frames0, frames1, frames2, points0, points1, points2, load1, load15, load5, m_temp, and m_volt. The data rows show various system instances with their respective values for each metric.

age	id	uptime	insert_time	lag	frames0	frames1	frames2	points0	points1	points2	load1	load15	load5	m_temp	m_volt
1327 (0)	KOUN1	5492	May-16 21:36:18	1	5429	5429	5429	633380	1025551	658125	1.159	1.503	1.325	104.6 (40.3)	12.853
1542 (215)	KOUN2	5111	May-16 21:32:43	9	5077	5077	5077	521716	958361	461005	0.000	0.000	0.009	109.2 (42.9)	12.479
1568 (241)	KOUN1	5252	May-16 21:32:17	0	5189	5189	5189	630438	978907	627415	1.374	1.598	1.489	105.2 (40.6)	12.925
1792 (250)	KOUN2	4871	May-16 21:28:33	3	4837	4837	4837	521259	944053	456694	0.064	0.005	0.037	106.3 (41.3)	12.639
1808 (240)	KOUN1	5012	May-16 21:28:17	0	4949	4949	4949	629978	929503	602863	1.439	1.651	1.567	105.2 (40.6)	13.174
2022 (230)	KOUN2	4631	May-16 21:24:43	10	4598	4597	4597	518639	887935	454118	0.000	0.000	0.004	106.3 (41.3)	13.370
2046 (238)	KOUN1	4773	May-16 21:24:19	1	4710	4710	4710	625545	882369	568568	1.555	1.709	1.676	104.6 (40.3)	12.479
2274 (252)	KOUN2	4391	May-16 21:20:31	1	4358	4357	4357	515864	818254	453236	0.022	0.006	0.027	105.7 (41.0)	12.461
2287 (241)	KOUN1	4532	May-16 21:20:18	0	4470	4469	4469	618544	834305	561897	1.646	1.723	1.710	104.6 (40.3)	12.568
2505 (231)	KOUN2	4151	May-16 21:16:40	8	4117	4117	4117	507384	750819	452422	0.118	0.024	0.050	105.7 (41.0)	12.568
2527 (240)	KOUN1	4292	May-16 21:16:18	1	4229	4229	4229	595497	789870	557830	1.667	1.750	1.737	104.6 (40.3)	14.155
2754 (249)	KOUN2	3911	May-16 21:12:31	1	3877	3877	3877	502004	720617	442172	0.000	0.026	0.012	106.3 (41.3)	13.566
2768 (241)	KOUN1	4052	May-16 21:12:17	0	3989	3989	3989	583581	745229	524953	1.543	1.751	1.696	105.2 (40.6)	12.693
2986 (232)	KOUN2	3671	May-16 21:08:39	0	3637	3637	3637	492033	717887	433050	0.000	0.050	0.042	106.9 (41.6)	13.263
3008 (240)	KOUN1	3812	May-16 21:08:17	0	3750	3749	3749	563889	700194	496288	1.774	1.801	1.810	105.2 (40.6)	12.479
3225 (239)	KOUN2	3431	May-16 21:04:40	3	3397	3397	3397	488443	699737	431185	0.131	0.073	0.109	106.9 (41.6)	12.960
3246 (238)	KOUN1	3572	May-16 21:04:19	1	3510	3509	3509	549185	652133	467176	1.573	1.803	1.820	105.2 (40.6)	12.550
3465 (240)	KOUN2	3191	May-16 21:00:40	8	3158	3158	3157	461291	656523	420043	0.111	0.064	0.101	106.9 (41.6)	12.497
3487 (241)	KOUN1	3332	May-16 21:00:18	0	3270	3270	3270	512477	603949	432806	1.638	1.811	1.865	104.6 (40.3)	12.942
3702 (237)	KOUN2	2952	May-16 20:56:43	10	2918	2918	2918	423519	612892	395965	0.027	0.045	0.049	106.9 (41.6)	12.622
3727 (240)	KOUN1	3092	May-16 20:56:18	0	3030	3030	3029	469313	559272	404428	1.801	1.818	1.922	104.6 (40.3)	12.461

Figure 2.15 CRM-Box Remote Control Server Web.

2.3 CRM System Database Server

Every day, the CRM-Box systems will transmit huge amounts of data back to the CRM database server piece by piece in a small data packages. These intermittent data packages will be packed and sorted one day after being received by the Linux corn job which will run a sorting PERL script algorithm to organize and form the CRM-Box system’s raw data. The sorting PERL file path on the CRM database server is located at “/home/ben/src/repack/sortp.pl”. The sorting PERL script algorithm will create a folder named in order of year-month-day, such as “YYYYMMDD”, and then create an airport ID named folder under the date-folder. Each CRM intermittent data will be assembled as raw data and put in order of their airport ID folder under the date folder. Figure 2.16 illustrates the CRM raw data structure from compressed and characterized images.

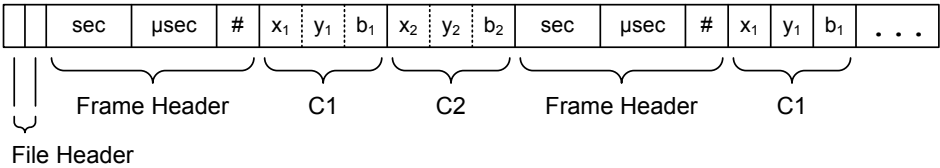


Figure 2.16 CRM Raw Data Structure.

The CRM raw data file stream starts with the file header which includes airport and camera ID using two characters format. Following the file header is the frame header which includes Coordinated Universal Time (UTC) second, microsecond and the number of landing light points in each frame image in order of two 4-byte and one 2-byte unsigned integers. The approaching aircraft’s

landing lights and some other lights pixel locations and brightness are written in three 2-byte unsigned integers.

Chapter 3 Methodology and Algorithm

Determining the three-dimensional positions of the approaching aircraft in the final visual segment of the landing path is similar to calculating an astronomical unit such as a distance from a star to Earth. The science behind the CRM visual segment research is solid geometry; the proper placing of the observation location and establishing a reference for calibration is the critical point for success to this CRM project. In other words, acquiring data through a proper method means less mathematical manipulation. The unique digital image processing and trigonometric function algorithm provides an effective solution for reconstructing the visual segment of the approaching airplane's Time and Space Position Information (TSPI).

3.1 CRM-Box System Runway Sighting

For the optical solution, the 50mm telescope lens for 1/4" CCD is ideal for this research and covers the entire approach field by using only one CCD camera instead of three used in an earlier attempt. This design will make an approach observation channel approximately 200 meters wide and at least 3 Nautical Miles (NM) from the touchdown point which will cover the typical visual segment of a aircraft on approach to landing. Figure 3.1 shows the sketch for the CRM-Box systems on each side of the instrumented runway as well as the ideal field of coverage. Figure 3.1 also displays the physical CCD minimum resolution (meter

per pixel) by this new telescope lens. The blue (CRM Master) and red (CRM Slave) lines intersection is the three dimension solution detection area is a typical approach to landing should fly through the channel between blue and red dotted lines. The red dot is represents the runway touchdown point. The reason the touchdown point is not inside the CRM the detection area is the limited of runway at KOUN. The runway length at KOUN is about 5000 feet and is much shorter than a normal commercial airport which usually has about 10,000 foot runways.

[9]

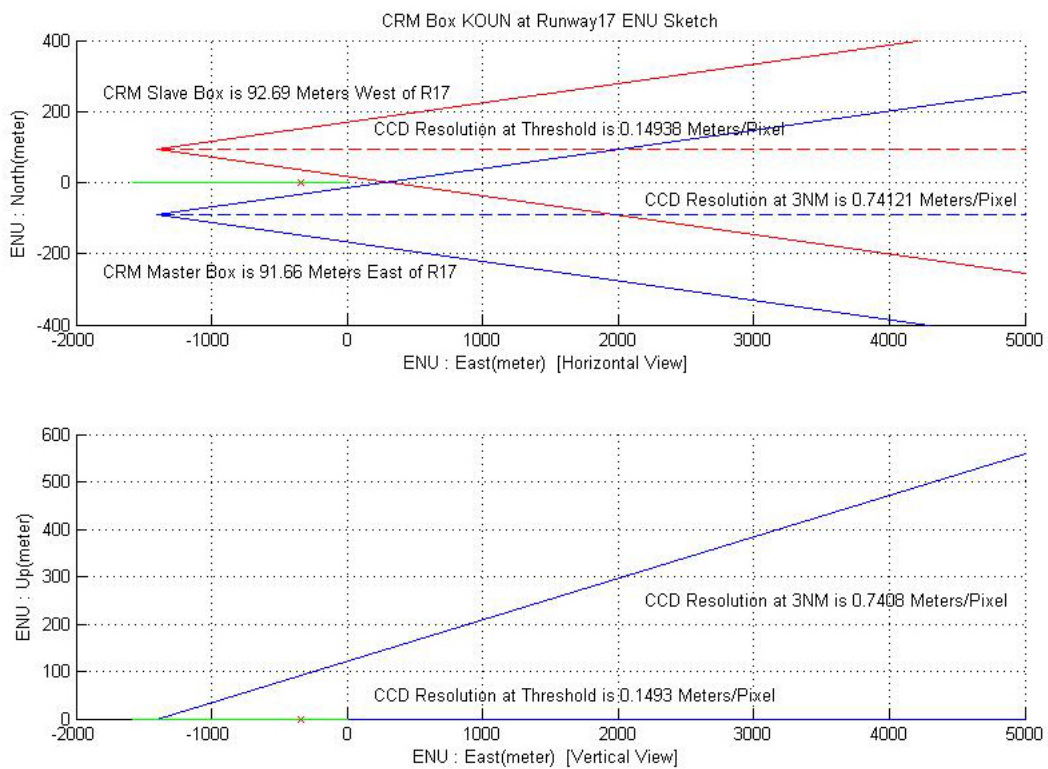


Figure 3.1 CRM Box CCD View Coverage.

Placing a calibration point in front of the CRM-Box systems through precision survey aids in the calibration of the CCD orientation and will also allow for checking any minor changes in CCD orientation. Figure 3.2 illustrates the KOUN runway 17 master and slave CRM-Box systems and reference rods arranged on the satellite photo on the top of the figure. The bottom two pictures are taken at top of the KOUN master CRM-Box system to show the view of heading at bottom left of the figure. The view of calibration rod in front of the master CRM-Box system is at bottom right of the figure.



Figure 3.2 KOUN CRM Boxes Runway Sighting.

The method requires a land survey transit instrument placed at the center of the subject runway to point out the perpendicular line relative to the runway for placing the master and slave CRM-Box systems at a distance of about 100 meters from the runway. The TOPCON Auto-Level AT-F1A is the land survey leveling instrument used in the CRM project. The airport will require at least 250 feet clear zone along the runway or taxiway to avoid the potential collisions. The leveling instrument is used again at both master and slave CRM-Box system locations to apply a previous point on the center of the runway that indicates two parallel lines relative to the instrumented runway. The two lines have same heading as runway 17-35 and at right side of runway 17 for master CRM-Box system and left side for slave CRM-Box system. Figure 3.3 shows the CRM-Box system side survey and the three parallel lines indicated in red arrow.

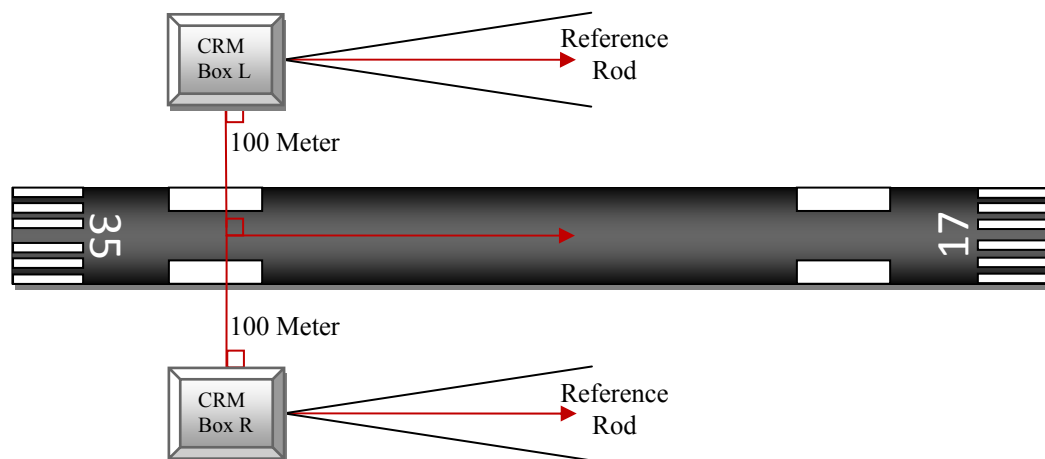


Figure 3.3 CRM System Runway Survey.

Both KOUN runway 17 CRM-Box systems observe the KOUN runway 17 aircraft. Two parallel lines also create the CRM-box system CCD orientation heading and can be placed the reference rods at about 100 meters in front of each CRM-Box systems for CCD aiming, alignment, and calibration. The red and blue dot lines on the top of figure 3.1 should match these two survey parallel lines. This special CRM-Box system sighting design and the angle of view from the 50mm telescope lens provide an ideal detection zone for observation the approaching aircraft. The proper CRM-Box system runway sighting will significantly reduce the CRM system digital image processing work load and limited the noise signals from ambient city lights.

3.2 Data Acquisition

The CRM data acquisition is a series of optoelectronic processing that transfer images from approaching aircraft's landing light into the electrical impulse. The optoelectronic signal is digitalized frame by frame and becomes the continuous two dimension raw TSPI data of each approaching aircraft. Because of energy consumption at the field CRM-Box system, the digital image processing of the CRM system is separated into two stages: one basic digital image processing at the CRM-Box system computer and the other, a more complicated digital image processing, at the CRM database server. The first basic digital image processing method calculates the landing light image pixels from a CCD into a single location by applying a gradient descending centroids algorithm.

Figure 3.4 top shows aircraft's blurry landing light and figure 3.4 bottom shows two different landing light image centroid algorithms. [10]

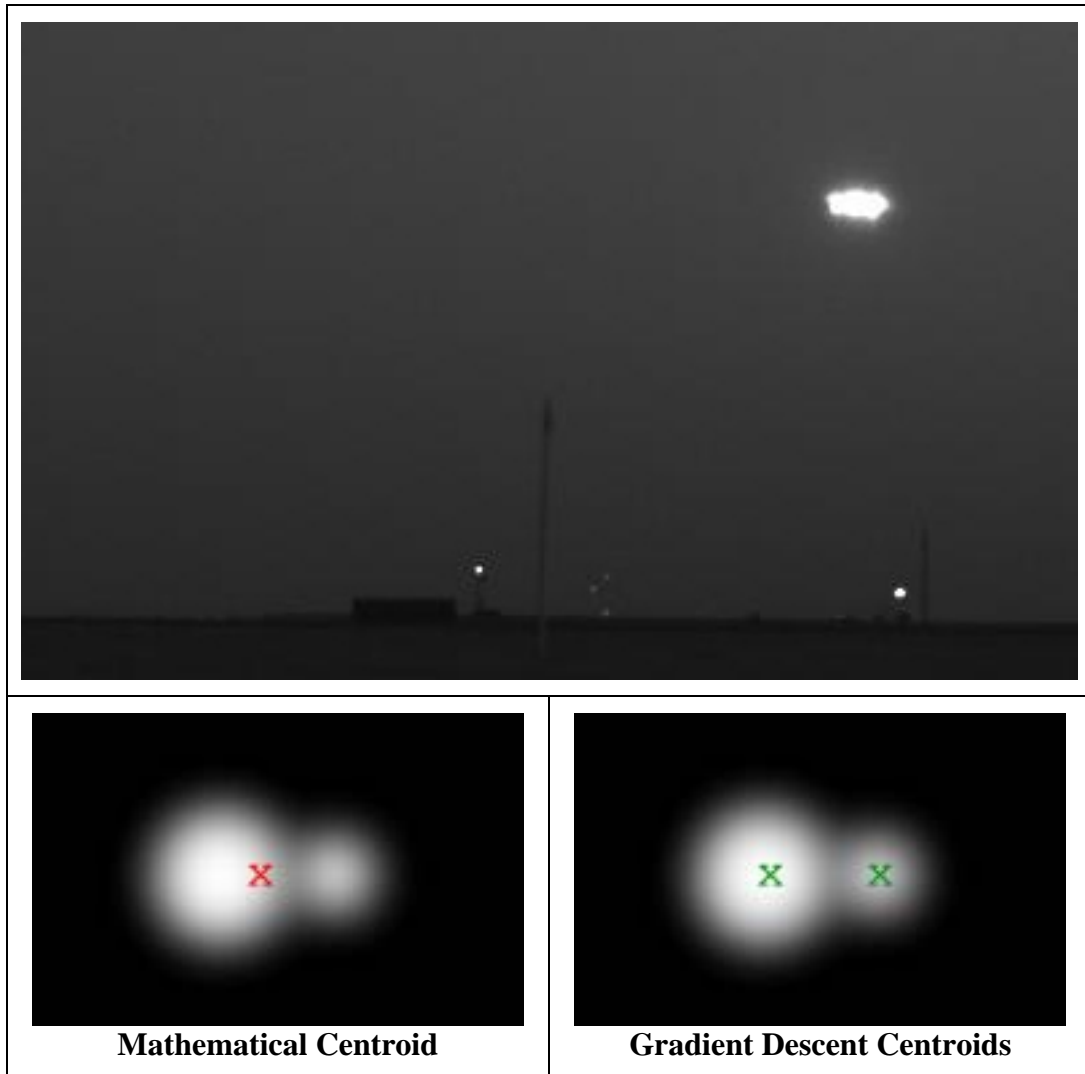


Figure 3.4 Landing Light Image Centroid.

The gradient descending centroids algorithm starts with locating possible bright spots from the CCD image. [11] The algorithm also applies the hill-climbing algorithm to locate the local maxima brightest spot for the possible aircraft landing lights. This method allows separation of the landing light's cloud.

[12] Then the mathematical centroid or mean centroid algorithm applies the physical center of mass to find the landing light's cloud location and remove the duplicate points. The gradient descending centroids algorithm is produces a better aircraft's TSPI resolution for the CRM system. [13]

Usually an aircraft is equipped with more than one landing light at the front landing gear. The gradient descending centroids algorithm is capable of pointing out the locations of two or more landing lights near each other. The 2D raw TSPI data package or CRM raw data structure includes UTC time when acquiring, 2D pixel locations of aircraft's landing lights, and the brightness of the lights for one frame of CRM raw data stream. The CRM raw data then transmits via GSM wireless network to the CRM database server for second stage digital image processing.

3.3 System Calibration

A system without calibration is useless and will yield results in poor data. After the CRM-Box systems are deployed in the field, a system calibration is necessary to guarantee the CRM system functions correctly. The CRM system calibration includes static land surveys, CCD lens orientation aiming, and a dynamic calibration flight. Each calibration step will fine tune the CRM system, resulting in an exceptional measuring tool for tracking an approaching aircraft.



Figure 3.5 Ashtech Z-Xtreme DGPS Receivers.

The purpose of static land survey is to locate the ENU (East, North, Up) of both master and slave CRM-Box systems relative to the instrumented runway threshold and check the CRM-Box systems runway sighting. The static land survey consists of four locations that include both master and slave CRM-Box systems and their calibration rods. This survey data combines the subject runway threshold and end points which can be accessed from the airport database to calculate both master and slave CRM-Box systems heading. The Ashtech Z-Xtreme differential GPS (DGPS) receivers are used to calibrate the CRM system in both the static land survey within centimeter accuracy and dynamic flight

calibration within half meter accuracy. Figure 3.5 shows a pair of the Ashtech Z-Xtreme DGPS receivers.



Figure 3.6 CRM Camera Aiming Software.

A Windows graphical user interface (GUI) application software was developed in C# to properly aim the CRM CCD Cameras. The main function of the CRM camera aiming software is to capture the CCD image from the USB port and plot additional horizontal and vertical half degree angle of view ticks for reference. The CCD aiming scope software also has a center crosshair plot on the capture image to indicate the center of the CCD pixels. Figure 3.6 shows the CRM camera aiming GUI.

With this CCD aiming software, the CRM CCD camera can be precisely aligned within one tenth of a degree horizontally using reference rod in front of the CRM-Box systems. The dynamic calibration flight uses the Ashtech Z-Xtreme in the kinematic mode as a truth system to record one flight of TSPI data and then compare the data with master and slave CRM camera data to compute the lens' true angel of view and the center offset within one hundredth of a degree.

3.4 Digital Image Processing

The second stage digital image processing starts with unpacking the binary CRM raw data from the CRM database server and forms a comma-separated values (CSV) file with UTC time, x pixel location, y pixel location, and brightness index value columns. Both master and slave CRM-Box system CSV files are loaded to a MatLab script for the digital image processing program to reconstruct the subject runway TSPI data of the day.

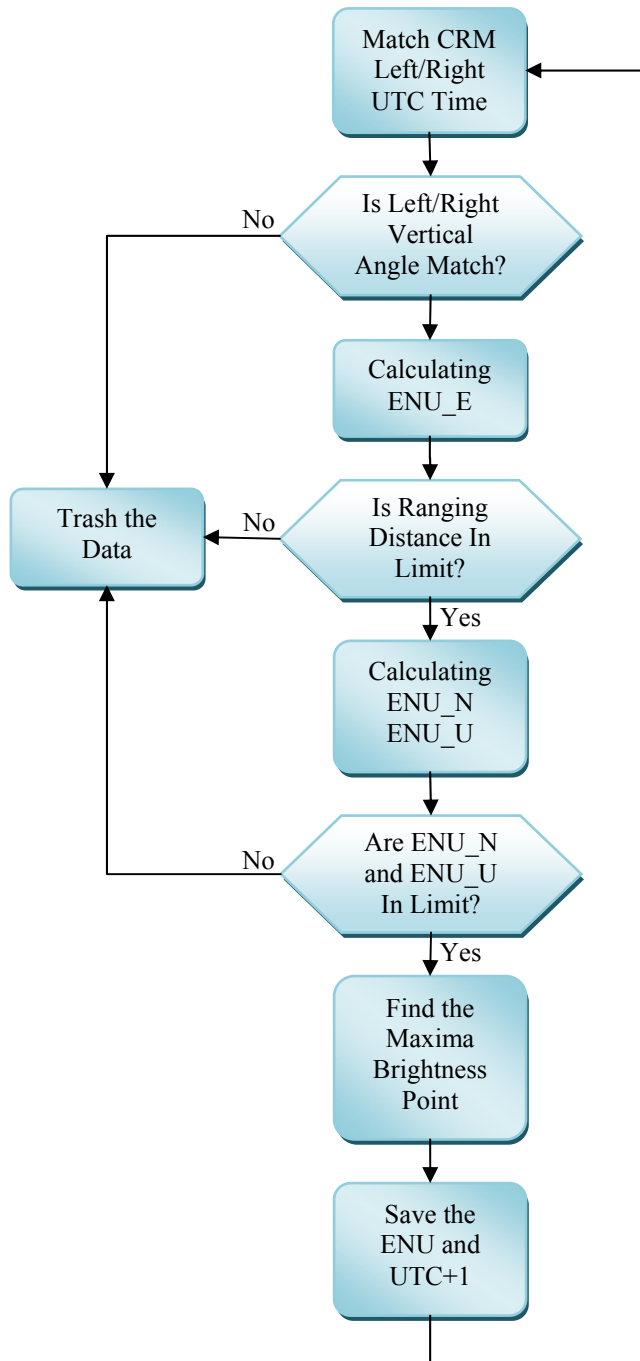


Figure 3.7 CRM DSP Flow Chart.

First in the MatLab script, the dynamic calibration data of left and right CCDs center offset and lens' angle of view are added to align the master and slave CRM-Box system's data with same UTC Time. Then transform the pixel locations to the angles of view data with matched vertical angles to less than 0.05 degree from left and right CRM-Box systems image. The Computed ENU data must be within the setup limits and pick up the brightness point for this UTC group. Figure 3.7 shows the CRM second stage DSP flow chart. [14]

The vertical angle matching is a simple and efficient algorithm of digital image processing that determines which two points belong to one single landing light from the left and right CRM-Box system images. The vertical angle matching algorithm also automatically filters out the scattered ground lights. This is because the typical aircraft approach to landing is located at the designed intersection of detecting zone. The probability of both master and slave CRM-Box system observations to an approaching aircraft's landing lights is extremely higher than the other ambient city lights when vertical angle comparison is applied. In most cases, the scattered city lights are not even in the intersection detecting zone. Figure 3.8 shows the CRM-Box system left and right images and demonstrates the vertical angle matching algorithm. N1 is second aircraft's landing light captures by left CRM-Box system and N2 is the ground tower light captures by the right CRM-Box system. N1 and N2 are noise lights relative to the approaching aircraft. When $\Delta\theta_L$ is equal to $\Delta\theta_R$, the left object and right object

are captured as the same object. For example, the $\Delta\theta_{LN1}$ is not equal to $\Delta\theta_R$ or $\Delta\theta_{RN2}$, there is no matched N1 image from the right side of the CRM-Box system image and N1 is designated noise. N2 also has no matched image at the left side of the CRM-Box system image. N1 and N2 then are filtered out by a vertical angle matching algorithm.

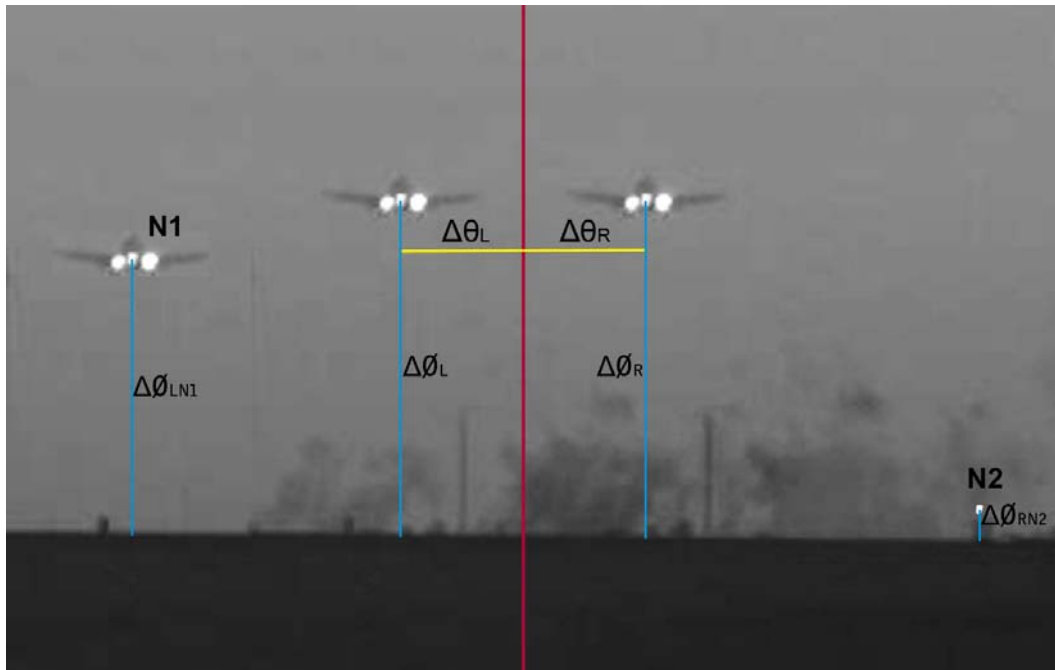


Figure 3.8 CRM-Box Camera Left and Right Images.

3.5 TSPI Construction

With a pair of left and right points identified from a single landing light of an approaching aircraft, the stereoscopic information can be calculated with Equations 1 through 5 to determine the ENU of the landing light relative to the instrumented runway threshold. Equations 1 and 2 are the trigonometric functions for determining the horizontal angle of the aircraft relative to the CRM-Boxes for

the left and right CRM-Box systems respectively. D_R and D_L are the distances measured from the CRM-Box systems to the center of the runway, and D_{BT} is the distance between the CRM-Box systems and the runway threshold. The distance is calculated from the DGPS land survey data by using Ashtech Truth System. $\Delta\theta_R$ and $\Delta\theta_L$ are the horizontal angles translated by measuring the CCD pixels, and $\Delta\phi$ is the vertical angle translated by measuring the CCD pixels. Both $\Delta\theta$ and $\Delta\phi$ are the view of angles from the CRM camera with calibration offset correction. Therefore, the standard East, North, Up (ENU) will be calculated with Equations 3, 4, and 5. Using this ENU data and comparing with the GPS Truth System calculation data the total system error can be determined. Figure 3.8 illustrates the CRM system TSPI Algorithm.

$$\frac{D_R - N}{D_{BT} + E} = \tan(\Delta\theta_R) \quad (1)$$

$$\frac{D_L - N}{D_{BT} + E} = \tan(\Delta\theta_L) \quad (2)$$

$$E = \frac{D_R + D_L}{\tan(\Delta\theta_R) + \tan(\Delta\theta_L)} - D_{BT} \quad (3)$$

$$N = \frac{D_L \tan(\Delta\theta_R) - D_R \tan(\Delta\theta_L)}{\tan(\Delta\theta_R) + \tan(\Delta\theta_L)} \quad (4)$$

$$U = \tan(\Delta\phi) \left[\frac{D_R + D_L}{\tan(\Delta\theta_R) + \tan(\Delta\theta_L)} - D_{BT} \right] = \tan(\Delta\phi) \times E \quad (5)$$

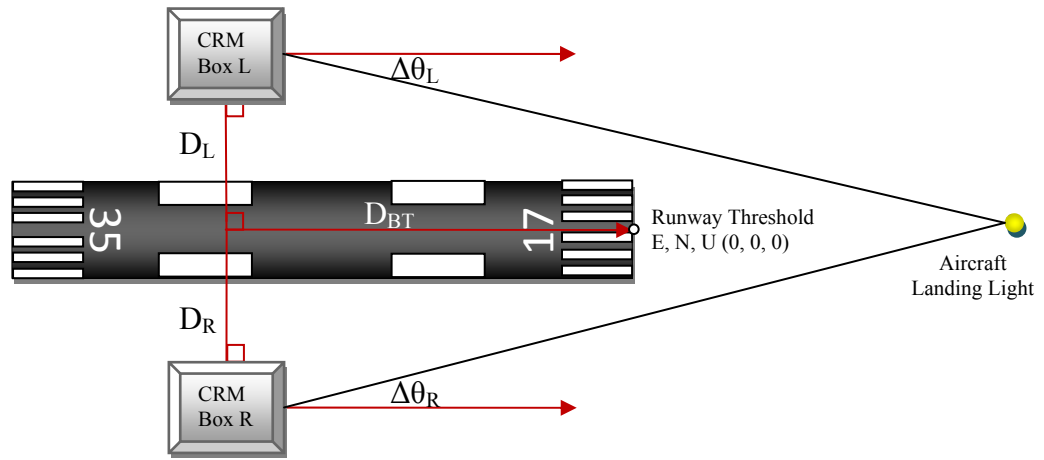


Figure 3.9 CRM TSPI Algorithm Illustration.

Chapter 4 Results

The results of the developed CRM systems performance are strong evidence of the functional realization of the CRM concept with allowable error. The CRM system flight test results also provide enough data to evaluate the CRM's system performance. Comparing the CRM system results to a Differential Global Positioning System (D-GPS) Truth System flight data has been useful to validate the CRM Total System Error (TSE) and developed algorithm performance. The CRM system TSE provides a guideline for the CRM system error components. When the CRM system error components are well defined, the CRM system is easier to review. An analysis of the CRM system error data has been used to improve the CRM's system performance. It was necessary to compare the CRM system flight test results with the Truth data in order to evaluate the performance of the CRM system presented in this dissertation.

4.1 CRM System Evaluation and Validation

It was necessary to set up a control system simultaneously with CRM system evaluation flight test for compares the result of the experiment. The control system in the CRM system flight test is a near ideal TSPI define measurement tool or Truth System. By comparing the measurement between CRM system and Truth System, the CRM system's performance can be easily observed.

4.1.1 Truth System

A Truth System is an on board device used for tracking the test aircraft and recording the Time and Space Position Information (TSPI) and Position Velocity Time (PVT) data. This tracking system is an Ashtech Z-Xtreme GPS Receiver installed in the test aircraft. [15] A similar GPS receiver is operated and records data at a surveyed location. The surveyed ground location must be near the CRM-Box system for post flight data analysis to produce a very accurate D-GPS post processing data. A pair of Ashtech Z-Xtreme Receivers acts as the Truth System and has demonstrated dynamic accuracy of less than 0.2 meter error at 5 Hz rate. [16]

4.1.2 Normalized ENU Method

The truth data is based on Earth Centered, Earth Fixed (ECEF-XYZ) and the WGS-84 Coordinate System. Although the ECEF-XYZ is very useful to determine the global position, it does not provide good local relative data at a local position on the globe. Thus, a local coordinate transformation from X, Y, Z coordinates to a local E, N, U (East, North, Up) coordinates is applied. Figure 4.1 shows ECEF-XYZ and ENU local tangent plane on the globe. Equation (6) calculates the conversion from ECEF coordinates to the local coordinates ENU. X, Y, and Z are reference points which are the runway threshold and x, y, and z are TSPI of the aircraft. ϕ and λ are the local latitude and longitude. To create

horizontal and vertical profiles, the TSPI and PVT data that is collected during the approaches done on the different subject runways is normalized to a single west bound runway. This is accomplished by rotating each runway heading to 270° as standard normalized ENU runway heading. Figure 4.2 shows the KOKC-35R calibration flight track on the left of figure and an normalized ENU plot on the right of the figure. As shown in Equation (7), the E_N , N_N , and U_N (Normalized) are created by rotating ENU at θ degrees which is the difference of 270° and the test runway heading. [17]

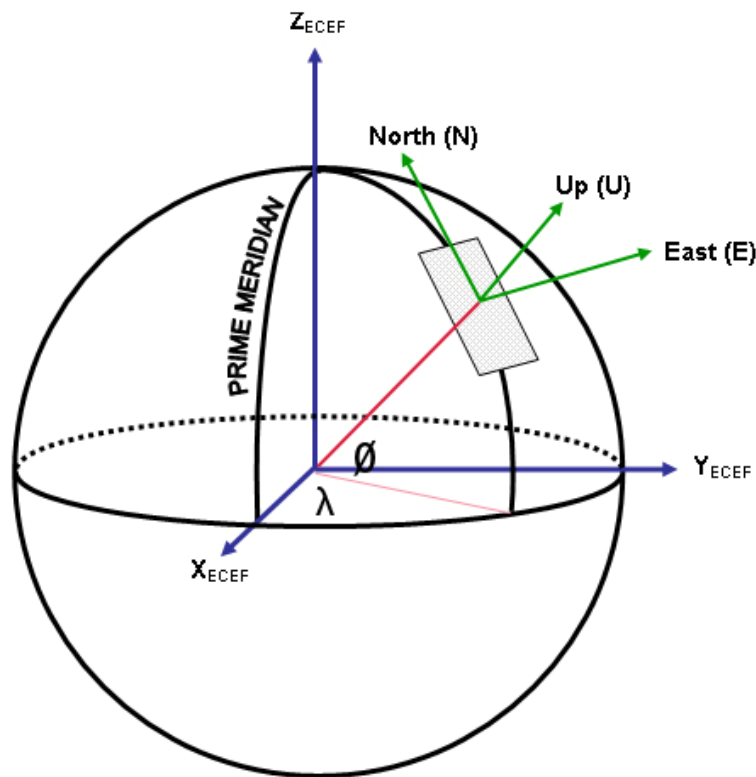


Figure 4.1 ECEF-XYZ and ENU Local Tangent Plane.

$$\begin{bmatrix} E \\ N \\ U \end{bmatrix} = \begin{bmatrix} -\sin \lambda & \cos \lambda & 0 \\ -\sin \phi \cos \lambda & -\sin \phi \sin \lambda & \cos \phi \\ \cos \phi \cos \lambda & \cos \phi \sin \lambda & \sin \phi \end{bmatrix} \begin{bmatrix} x - X \\ y - Y \\ z - Z \end{bmatrix} \quad (6)$$

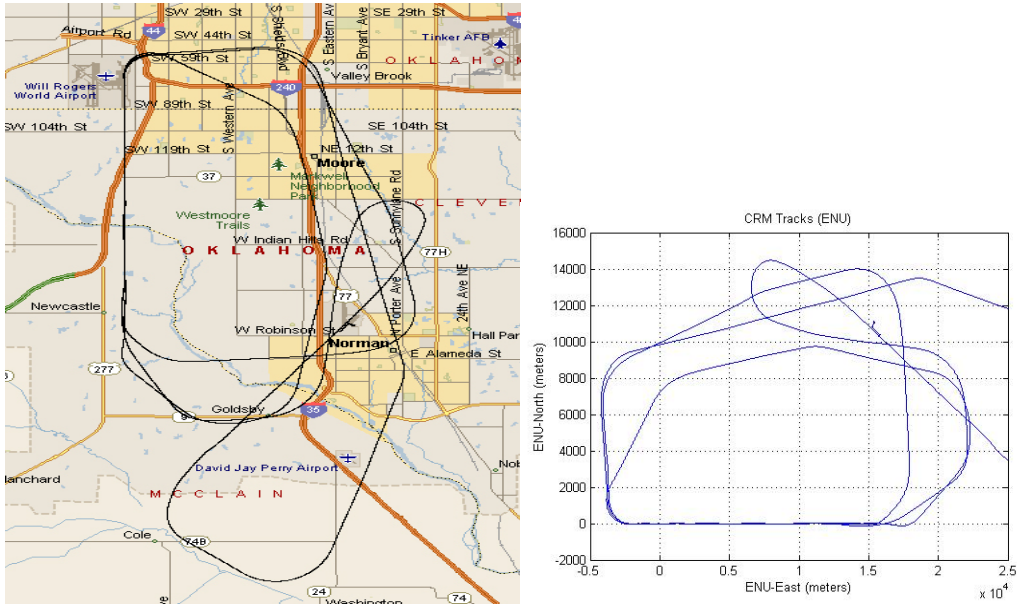


Figure 4.2 Flight Track ENU (Left) to Normalized ENU (Right).

$$\begin{bmatrix} E_N \\ N_N \\ U_N \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} E \\ N \\ U \end{bmatrix} \quad (7)$$

4.1.3 Time Domain to Space Domain

Because aircraft do not fly at a constant velocity and the Truth System measures the aircraft's position at a period of time, the distribution of the aircraft's position is not separated by a uniform distance. In order to calculate the

statistical data for the CRM metric, it is necessary to transfer the data from the time domain to the space domain. Equation (8) is the conversion from $P_{E,N,U}(t)$ to $P_{E,N,U}(d)$.

$$P_{E,N,U}(d) = P(t) + \frac{P(t) - P(t-1)}{\|P(t) - P(t-1)\|} \times (C \times n - k) \quad (8)$$

$$k = \|P(t)\| \% C \quad (9)$$

$$0 \leq (C \times n - k) \leq \|P(t) - P(t-1)\|, \quad n = 1, 2, 3, \dots \quad (10)$$

Here, $P_{E,N,U}(t)$ is TSPI data which has sets of data E, N, U with 0.2 sec increments or 5 Hz sample rate. Constant C is a space interval of the linear interpolation method and the programming operator % in Equation (9) is the remainder of division operation or module. Thus, the 5 Hz data linearly shapes to a uniform 10 meters interval sets of $P_{E,N,U}(d)$ corresponding to the range distance (ENU-East) from the runway threshold and ready for any CRM approach flight statistical processing. [18]

4.2 CRM System Errors Analysis

The CRM system error analysis is separated into Total System Error (TSE), CRM system time error, and CRM system error components definition error. Figure 4.3 shows one calibration flight digitalized into both left and right 2D image in the CRM-Box system. The image includes several stationary and

dynamic noise signals such as other airplane's landing lights and other stationary lights.

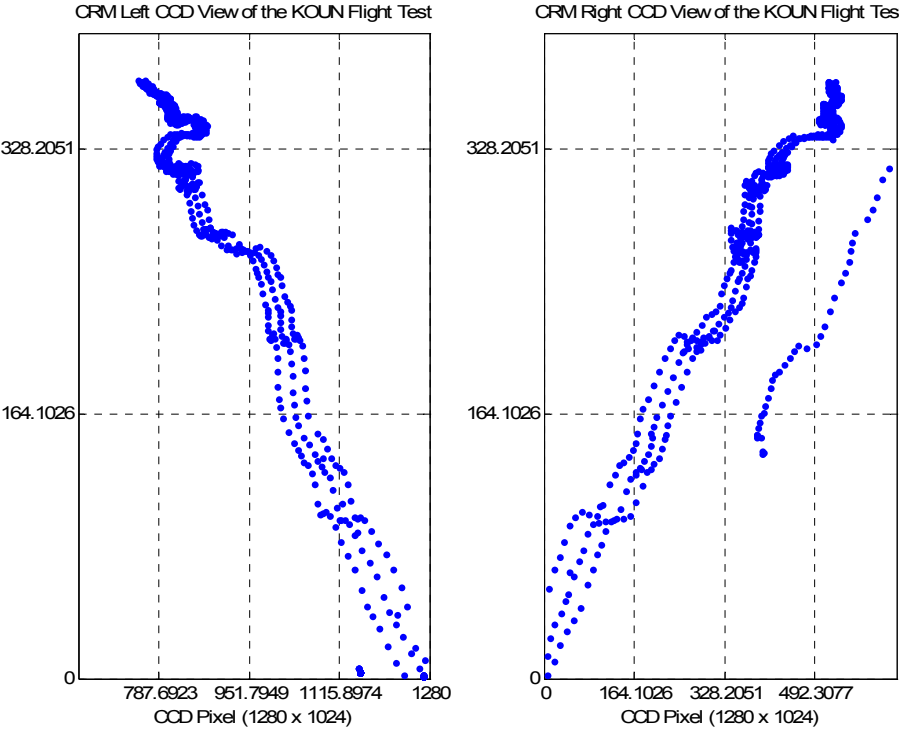


Figure 4.3 CRM System Left and Right View of CCD.

The CRM system left and right images are then processed using a unique new transform into horizontal and vertical angle representative information. Hence, the noise signal can be easily identified by angle measure plots. Figure 4.4 shows CRM system CCD angle measurement plots. From the top to the bottom of the plots included in the figure are the right horizontal angle, left horizontal angle, right vertical angle, and left vertical angle of the CRM system CCD angle measurements. The blue-dots indicate the aircraft's landing light and

noise lights. The figure also includes angle measure plots generated from Truth System data as the red line represented. Because the horizontal and vertical data are grouped as one pair, the noise signal can be easy filtered out by using the vertical angle matching algorithm. The red line of the Truth System information track in the plot can also indicate the quality of the CRM system calibration statue.

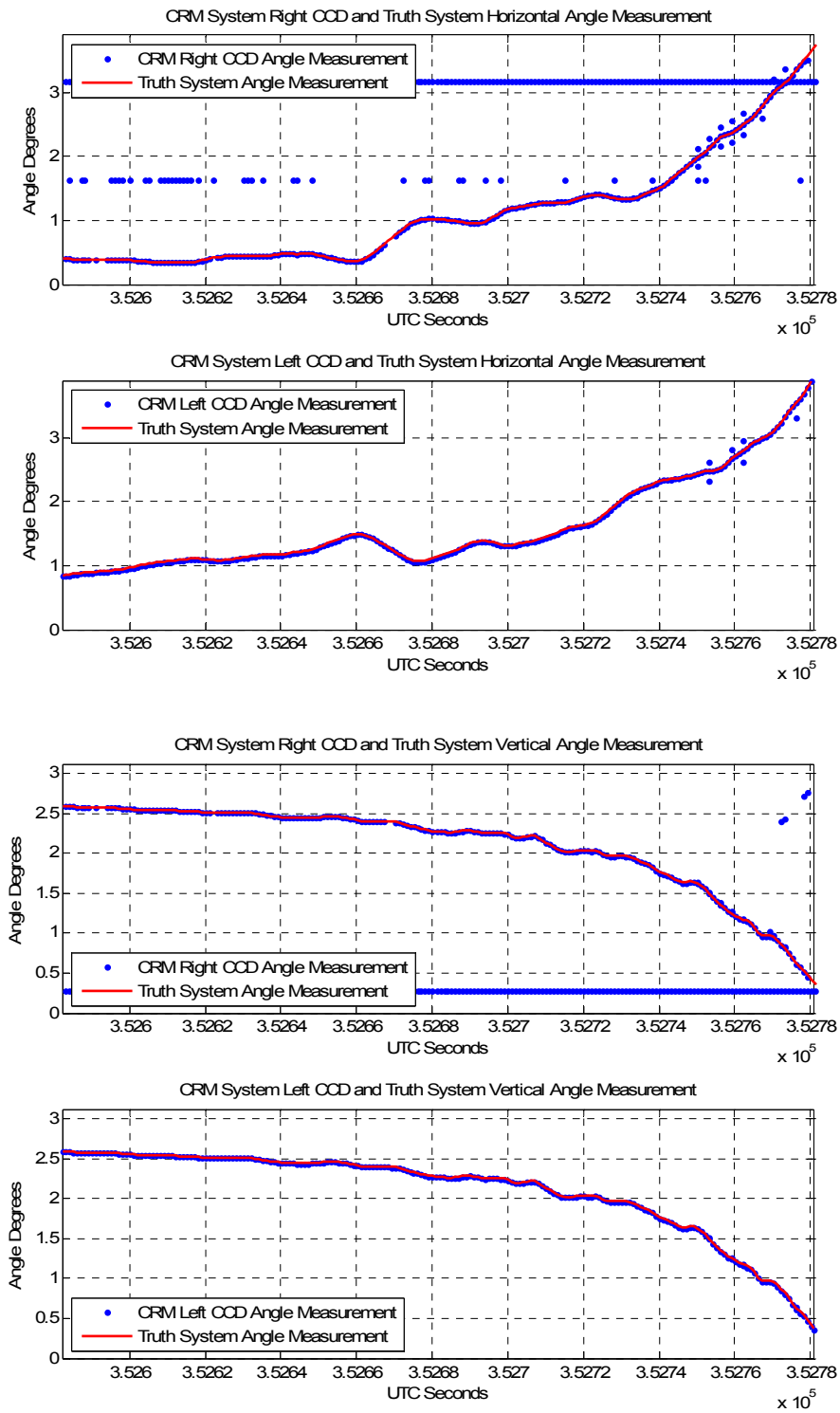


Figure 4.4 CRM System CCD and Truth System Angle Measurement.

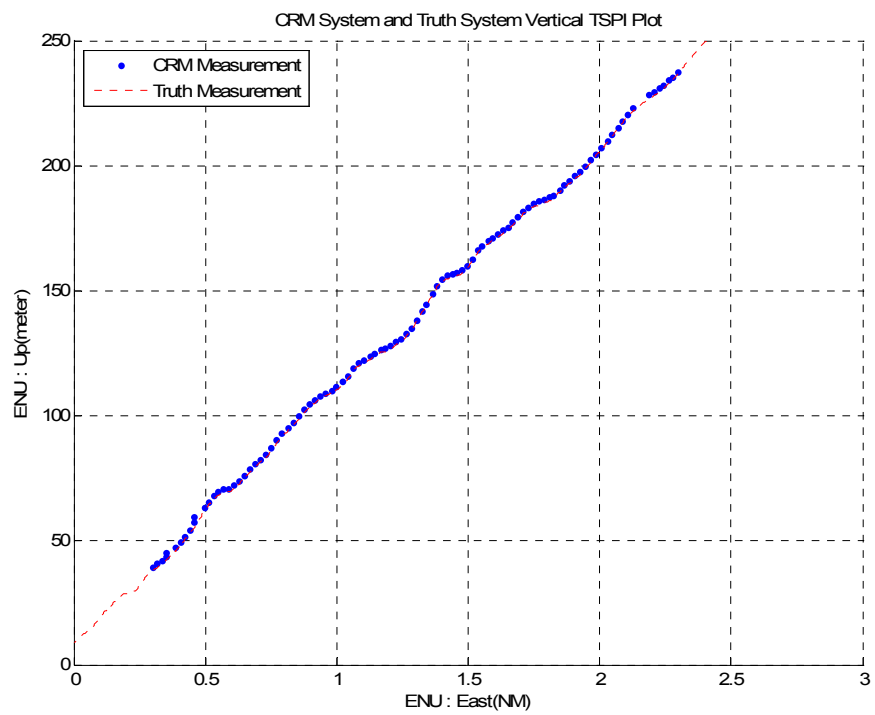
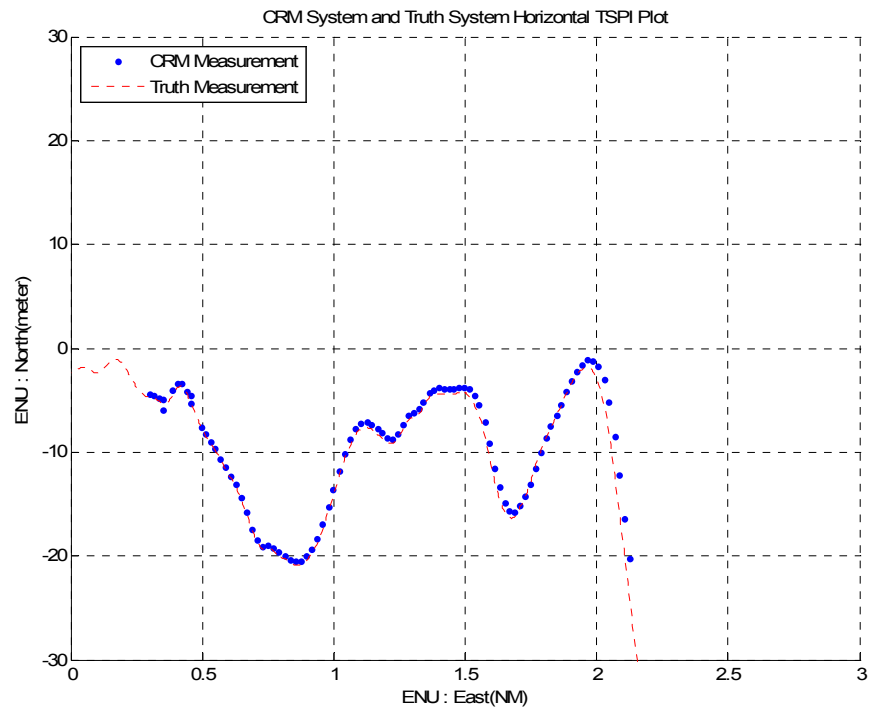


Figure 4.5 CRM and Truth System Horizontal and Vertical TSPI Plot.

After the calibration is verified, the CRM system's unique DSP algorithm computes the angle measurement data in order to construct the CRM ENU data for the test flight. The algorithm applies Equations 1 through 5 with figure 3.9 from Chapter 3-3.5 with the horizontal and vertical angle data present in Figure 4.4. Figure 4.5 shows CRM Horizontal and Vertical TSPI Plot along with Truth System data for comparison. Thus, the CRM system Total System Error (TSE) can be determined with the corresponding time of the Truth System data.

4.2.1 Total System Error

The CRM system TSE can be determined by the difference between the CRM system measurement and Truth System measurement. The method is applying the root mean squared error (RMSE). [19] Figure 4.6 shows a calibration flight test result of CRM system and Truth System measurement in three-dimensions which is built from the horizontal and vertical TSPI data shown in Figure 4.5. The blue dot line is the measurement from the CRM system and thin red line is the Truth System measurement. The black dot line is the shadow of the track on the ground for the horizontal and vertical of the blue dot line as well as the black thin shadow line from the thin red line for Truth System data represented. The CRM system Total System Error (TSE) is discussed in the ENU three components.

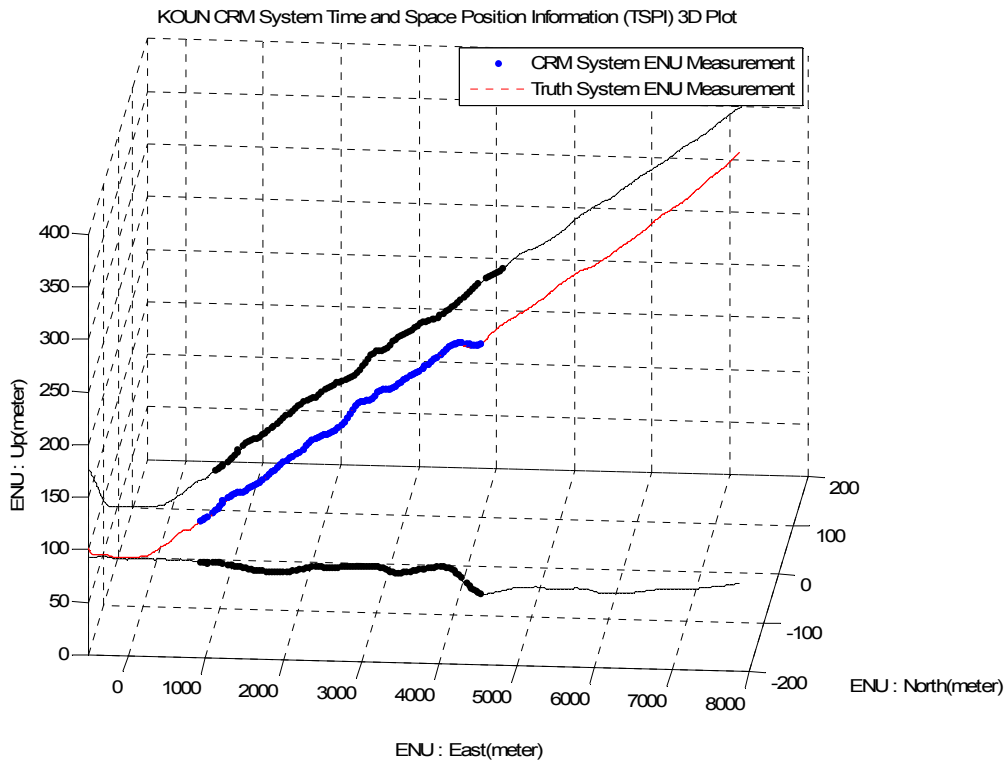


Figure 4.6 KOUN CRM 3D TSPI Plot.

The CRM system ENU coordinates is a rectangular coordinate system that the origin (0, 0, 0) is at the runway threshold. The ENU-East is the range distance from the target aircraft to the runway threshold. The ENU-North is the cross track distance of the desired approach line corresponding to the Terminal En-Route Procedures (TERPS) of the runway. The ENU-Up is the altitude of the target aircraft from the runway threshold altitude. Figure 4.7 shows the CRM system Total System Error (TSE) in ENU and its three component errors separately along with the range distance ENU-East.

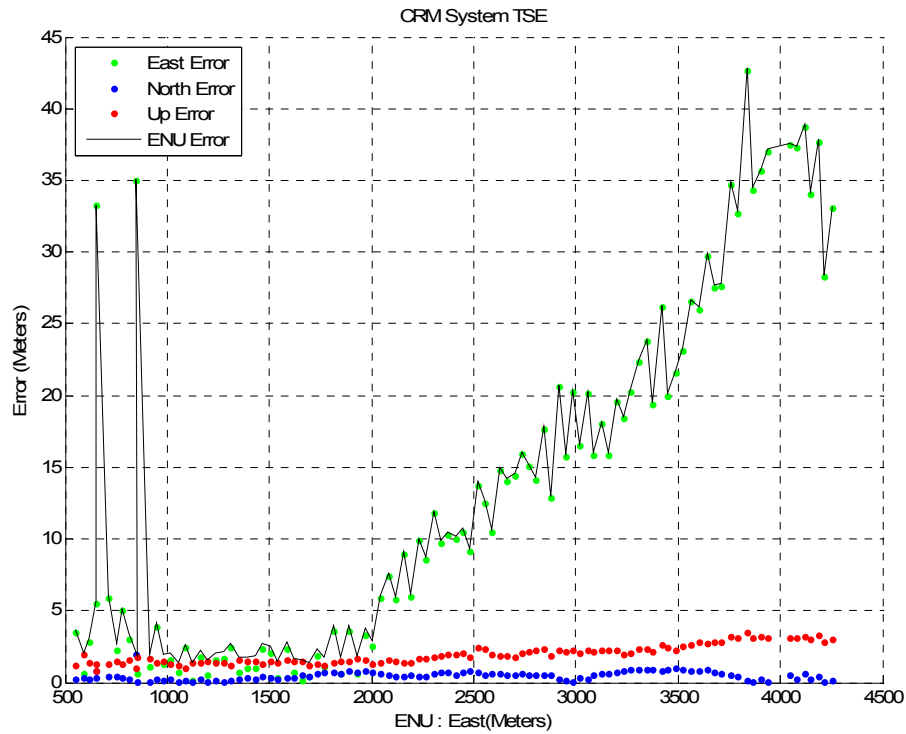


Figure 4.7 CRM Systems TSE With Distance East.

The consequence from Figure 4.7 indicates that as the range distance goes up, the CRM system produces more position error. The CRM system TSE results also implies that the cross track distance error (TSE ENU-North) and the altitude error (TSE ENU-Up) maintain in the same amount of error as the range distance is varied. Therefore, the CRM system TSE, as expected, is a function of distance from the runway threshold. Table 4.1 shows the mean of the CRM system TSE from Figure 4.7 in order of the error at different ranges for a calibration flight. Table 4.2 shows the standard deviations to accompany the data in Table 4.1. The CRM system TSE from the range 1,000 meter to 4,000 meter has results of 2.2,

11.92, and 25.37 meters and an average error is 13.94 meters. The CRM system TSE is reliable at 1,000 meter to 2,000 meters which is the region of interest for an approach.

Mean (Meters)					
Range	Samples	TSE(ENU)	TSE (E)	TSE (N)	TSE (U)
0-600	2	2.8806	2.0727	0.2929	1.5804
600-1000	12	8.7020	8.2905	0.4451	1.3688
1000-2000	26	2.2054	1.4880	0.3664	1.3993
2000-3000	28	11.9199	11.7394	0.5316	1.8918
3000-4000	26	25.3655	25.2200	0.6267	2.5888
>0	101	13.9442	13.5977	0.4848	1.9583

Table 4.1 TSE Mean of the Calibration Flight.

Standard Deviation (Meters)					
Range	Samples	TSE(ENU)	TSE (E)	TSE (N)	TSE (U)
0-600	2	1.1728	1.9962	0.0261	0.4756
600-1000	12	11.9841	12.1920	0.5196	0.2695
1000-2000	26	0.7510	1.0303	0.2410	0.1547
2000-3000	28	4.3161	4.3496	0.1715	0.3218
3000-4000	26	7.3134	7.3133	0.2924	0.4385
>0	101	11.9977	12.2055	0.2959	0.6417

Table 4.2 Standard Deviation of the Above Calibration Flight.

There are six CRM system calibration flights for the KOUN station. The CRM-Box system has collected as little as 39 to as many as 101 samples for each calibration flight. Table 4.3 shows the CRM system TSE mean errors of each calibration flight that also includes TSE, TSE-East, TSE-North, and TSE-Up error components and has a matched standard deviation shown in Table 4.4. The

calibration flight experiments have indicated that the mean of the CRM TSE is 28.90 meters of total 439 samples. The varied CRM system TSE reveals a possible reason of more than one error source constructing the CRM system Total System Error.

Mean Error (Meters)					
Flight #	Samples	TSE(ENU)	TSE (E)	TSE (N)	TSE (U)
1	101	13.9442	13.5977	0.4848	1.9583
2	101	28.1106	27.7645	0.9024	2.3994
3	42	40.8580	39.8956	3.6895	2.6973
4	39	36.9655	36.1645	3.7429	2.5539
5	75	38.7171	38.4221	1.4165	2.5145
6	81	29.3351	28.9096	1.5373	2.2062

Table 4.3 Mean Error for 6 Calibration Flights.

Standard Deviation (Meters)					
Flight #	Samples	TSE(ENU)	TSE (E)	TSE (N)	TSE (U)
1	101	11.9977	12.2055	0.2959	0.6417
2	101	37.3925	37.5390	0.8287	1.1102
3	42	39.8702	40.5517	1.5794	0.9539
4	39	42.2907	42.7022	1.5574	1.2833
5	75	28.9233	29.1349	0.8800	1.2503
6	81	27.2007	27.4703	1.1319	1.3243

Table 4.4 Standard Deviation for the Above 6 Calibration Flights.

4.2.2 CRM System Time Error

Since the CRM system TSE is calculated by a comparison to a Truth System, the amount of time difference between CRM system and Truth System will introduce some amount of error in the CRM system TSE computation.

Figure 4.8 is the time difference between the Truth System to the left and right CRM-Box system. The red plot indicates the time difference between left and right CRM-Box system which is about sixteen milliseconds. The green plot is the difference between Truth System and the left CRM-Box system and blue plot is the difference between Truth System and the right CRM-Box system at KOUN. The CRM system time error is about 100 – 200 milliseconds.

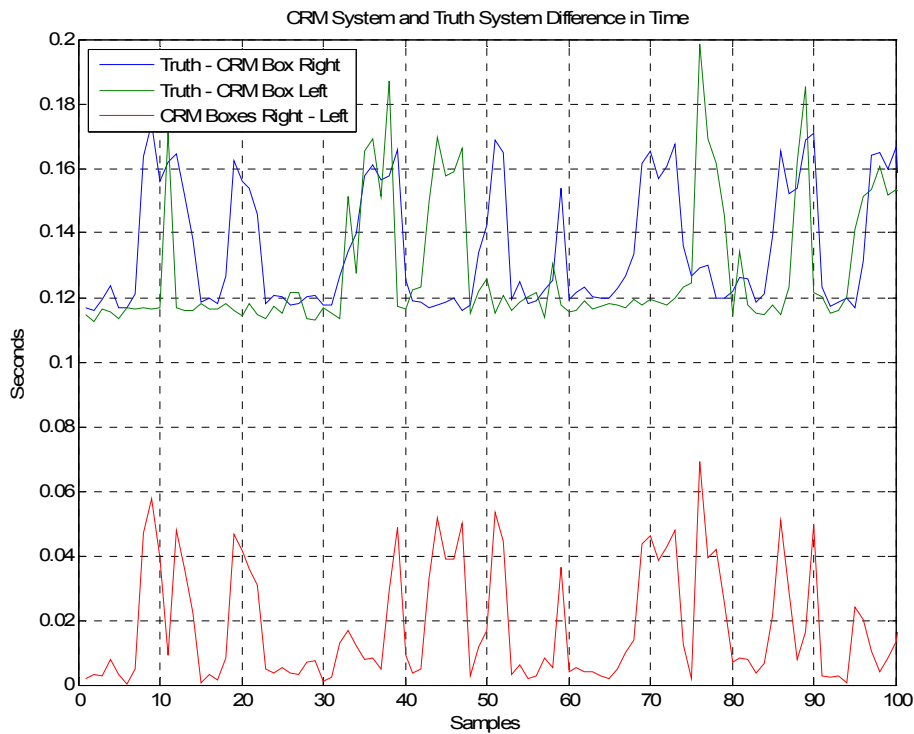


Figure 4.8 CRM-Box Systems and Truth System Difference in Time.

During the approach, the aircraft is moving much faster in the ENU-East axis than ENU-North and ENU-Up axes. This is because the ENU-East axis is aligned to the range distance from the runway threshold to the approaching

aircraft and the movement on this axis is the approaching speed. Typical Category-A aircraft final approaching speed is about 70-100 knots and 200 millisecond can produce 7.20-10.29 meters in error. Table 4.5 indicates ICAO aircraft approach category of the specified range of handling speeds in knots. V_{AT} is the speed at threshold based on 1.3 times stall speed in the landing configuration at maximum certified landing mass. [1] The CRM System Time Error (STE) produces more error in CRM system TSE ENU-East component. This is the main reason for the amount of CRM system Total System Error ENU-East part compared with Truth System.

Aircraft Category	V_{AT}	Range of Speed for Initial Approach	Range of Final Approach Speeds	Max Speed for Visual Maneuvering (Circling)
A	<91	90/150	70/100	100
B	91/120	120/180	85/130	135
C	121/140	160/240	115/160	180
D	141/165	185/250	130/185	205
E	166/210	185/250	155/230	240

Table 4.5 ICAO Aircraft Approach Category (knots).

4.2.4 CRM System Error Components Definition

The CRM Total System Error (TSE) includes System Time Error (STE), Ideal Physics Limit (IPL), Thermal and Visual effect Optical Error (TVOE), and CCD Computing Error (CCE). The CRM system STE has an upper bound of

10.29 meters with maximums of 100 knot and 200 millisecond latency. The estimate Ideal Physics Limit is the error boundary that has been measured by the manufacturer of the CRM components and can be found by examining the specifications. The main IPL is the CCD resolution which translates to meter per pixel. Figure 4.9 shows the CRM-Box systems CCD resolution. The upper bound of the IPL in the CRM system is horizontal of 74.12 centimeters and vertical of 74.08 centimeters at 3NM range distance. The other CRM system IPL are Bits Error Rate (BER) from the CCD pixel error, GSM wireless transmission error, and local wire and wireless transmission error. Those BER are usually less than 10^{-6} to 10^{-8} and negligible.

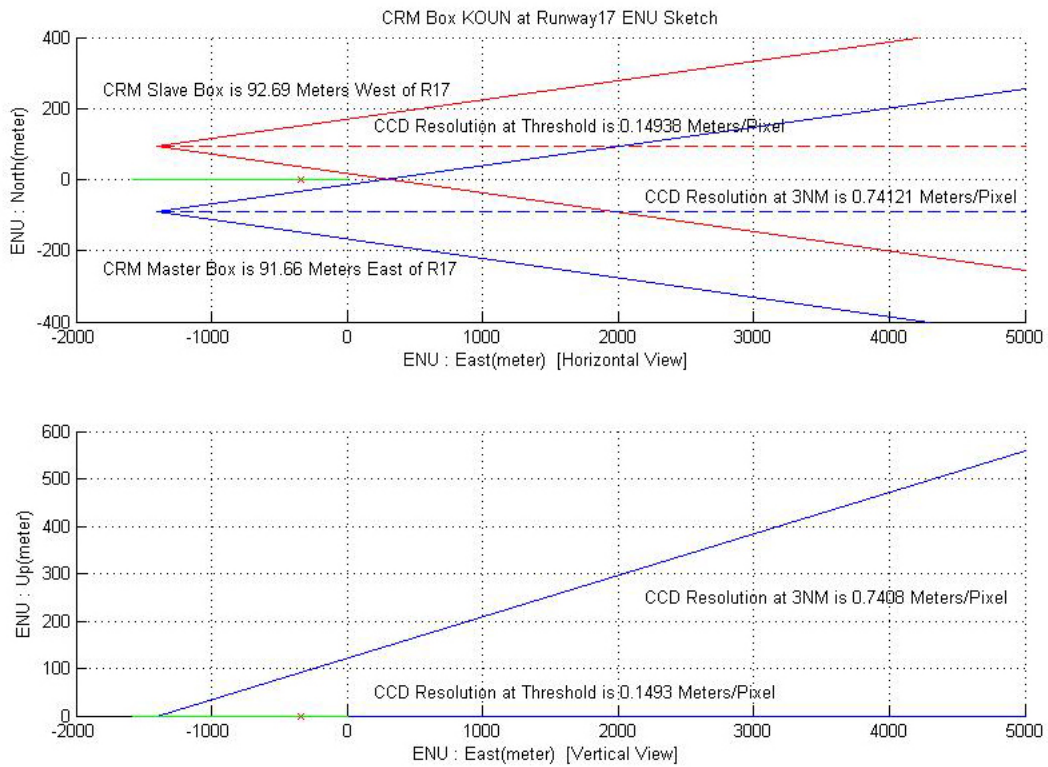


Figure 4.9 CRM-Box Systems CCD Resolution.

The Thermal and Visual effect Optical Error (TVOE) is the error created by weather such as cloud, dust, fog, rain and heat waves from the earth ground. Clouds and dust are the main reasons for the reduction of the landing light brightness and the continuity of the landing light for the track. Fog and rain cause blurry and distorted images as well as the ground thermal waves. The Ideal Physics Limit (IPL) and Thermal and Visual effect Optical Error (TVOE) are errors cannot be controlled and removed, but must be accounted for.

4.3 Results

The result of the CRM project shows the CRM system TSE can be decomposed into ENU three components representation. The CRM total system error can be expanded as in Equation (11). CRM STE and IPL are fixed by the boundary of 11.03 meters. The only error that can be improved in the CRM system is the CCD Computing Error which has two stages in the CRM-Box embedded system and CRM server computer.

$$TSE = CCE + TVOE + [STE + IPL] \quad (11)$$

$$[STE + IPL] \leq 10.29 + 0.74 = 11.03(meters) \quad (13)$$

The results of the CRM TSE performance are an average of 28.90 meter and have standard deviation of 31.94 meters. The CRM TSE is within 28.44 meters in East component, 1.53 meters in North component, and 2.32 meters in

Up component based on the 6 calibration flights with a total of 439 samples. Compared to modern air traffic radar, the CRM system has much better tracking resolution in the observation distance less than 5000 meter. The modern air traffic radar has the Radar Cross-Session (RCS) volume about $0.033 \times 69.81 \times 69.81$ (ENU) cubic-meter at 4000 meter with 9 GHz 1 degree beam width and can only scan every six seconds. The radar observation resolution for every six second is about 161 cubic-meters and the CRM system has observation tolerance at average of 99 cubic-meters every second.

The normal parameter estimates (normfit) provides 95% confidence intervals for the parameter estimates on the mean and standard deviation of the CRM TSE. The confidence lower and upper bounds intervals of mean and standard deviation are 25.90-31.90 and 29.96-34.21 meters. [20] [21] The correlation coefficient of the CRM TSE and a normal distribution is 0.96. Figure 4.10 shows the CRM TSE distributions histogram with a Gaussian distribution fit and confirms the result of CRM TSE is a normally distributed error. The CRM TSE six-sigma boundary is 220.55 meter and its East, North, and Up six-sigma boundary are 221.47 meter, 10.46 meter, and 9.02 meter. [22] These six-sigma values are the precise performance boundaries for the CRM system.

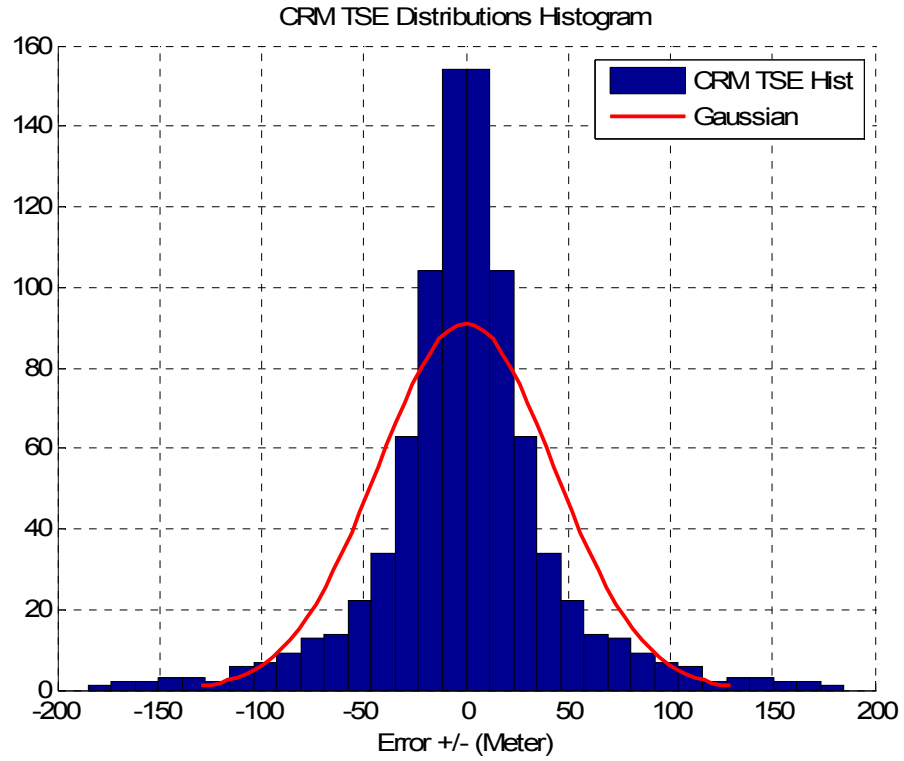


Figure 4.10 CRM TSE Distributions Histogram.

Chapter 5 Conclusions

The CRM system is the first low cost attempt to visually track a high speed approaching aircraft. The system satisfies the requirement to provide a large volume of track data on an area of the approach that had not been examined for the risk for collision of each type of aircraft on final approach after leaving IMC conditions. The system is new and novel and is capable of delivering a wide variety of approach and departure data in order to help develop the FAA's Collision Risk Model for the final approach to landing in IMC conditions. Much has been learned about commercial pilots and aircraft on approach to landing. More has been learned about the use of the auto pilot as compared to the manual approach to landing. Controller and pilot error in the critical phase of flight can be determined in order to implement new FAA procedure for the final approach to landing.

The CRM system developed as part of this research is a unique combination of hardware and software that has no duplicate in today's technology. This CRM optoelectronic three-dimensional tracking system has proved its functional integrity and has successfully produced the TSPI data at the University of Oklahoma Westheimer Airpark (KOUN) and Oklahoma City Will Rogers World Airport (KOKC). The CRM system dissertation meets all requirements of CRM visual segment flight tracking data acquirement functions for the FAA. The CRM system now is ready to deploy in the field of any subject

airport of CRM visual segment study and will provide the CRM database during Instrument Meteorological Conditions (IMC) at visual segment from approach to landing for the FAA.

The dissertation successfully introduces a new and novel engineering solution for the CRM research. The accomplishment of this dissertation includes a complete CRM system architecture, a unique CRM system runway field sighting method, and an efficient digital image processing algorithm.

Future work in this field may include adding a new mathematical filtering algorithm to reduce the CRM system error and an executable file integrated into the CRM database server instead of a MatLab script for second stage digital image processing. The CCD camera can also be upgraded to a much higher resolution in order to improve the precision of the measurement for the CRM-Box system.

REFERENCES

1. International Civil Aviation Organizations, "Manual on the Use of the Collision Risk Model (CRM) for ILS Operations", Montreal, Quebec, Canada: ICAO Doc 9274. First Ed. 1980.
2. Herman, L.K.; "The history, definition and peculiarities of the Earth centered inertial (ECI) coordinate frame and the scales that measure time"; IEEE-AAC, 2(0):233-263, 1995.
3. B. Sridhar and R. Suorsa, "Comparison of motion and stereo methods in passive ranging systems", IEEE Trans-AES., 27(4):741-746. 1991.
4. Atmel Corporation ATmega8L microcontroller, 2486Q-AVR-10/06.
5. Karim Yaghmour, "Building Embedded Linux Systems", O'Reilly & Associates Inc, 2003.
6. Nicholas D. Wells, "Linux! I Didn't Know You Could Do That...", Second Edition, Sybex Inc, 2001.
7. Wall, Larry, Tom Christiansen and Jon Orwant, "Programming Perl", Third Edition, O'Reilly, July 2000.
8. Sheppard, Doug, "Beginner's Introduction to Perl". O'Reilly Media, 2000-10-16. <http://www.perl.com/pub/a/2000/10/begperl1.html>.
9. Peter S. Jergensen, John L. Jergensen, Troelz Denver, Maurizio Betto, Louis Toscon, "Autonomous Target Ranging Techniques", IEEE-0-7803-8 142-4/03.
10. Mordecai Avriel, "Nonlinear Programming: Analysis and Methods", Dover, 2003

11. M. Irani and P., "Anandan. A unified approach to moving object detection in 2d and 3d scenes", IEEE-PAMI, 20(6):577–589, 1998.
12. Jan A. Snyman, "Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms", Springer, 2005.
13. Russell, Stuart J.; Norvig, Peter, "Artificial Intelligence: A Modern Approach", 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2003.
14. Jorgensen, P.S.; Jorgensen, J.L.; Denver, T.; Betto, M.; Toscon, L.; "Autonomous target ranging techniques", RAST, 275-280, Nov 2003.
15. J.Farrell & M.Barth "The Global Positioning System & Inertial Navigation", McGraw Hill, 1999.
16. Y. P. Huang, H. Wen, J. Dyer, J. Fagan, "Flight Test Results of a MOPS Compliant LAAS System to Provide Guided Straight and Curved Path Departures and Missed Approaches", ION GNSS 2005.
17. J. Zhu, "Conversion of Earth-centered Earth-fixed coordinates to geodetic coordinates," Aerospace and Electronic Systems, IEEE Trans, vol. 30, pp. 957-961, 1994.
18. Y. P. Huang, J. Fagan, Yu-Zhen Xue, "Validation of the Offset Precision Approach with Vertical Guidance to Landing using LAAS as Sole Means of Navigation", ION GNSS 2008.
19. George Casella & E.L. Lehmann, "Theory of Point Estimation", Springer, 1999.
20. Edwards, A. L, "The Correlation Coefficient", W. H. Freeman, 1976.

21. Rice, John, "Mathematical Statistics and Data Analysis", Second ed., Duxbury Press, 1995.
22. Mood, A., F. Graybill, D. Boes, "Introduction to the Theory of Statistics", 3 ed., McGraw-Hill, 1974.

APPENDICES

Appendix A: CRM-Box CCD Camera Aiming Tool C# Code

```
////////////////////////////////////
// CRM-Box System CCD Camera Aiming Tool
//
// MS-Visual Studio 2005 C#
// CRM_CamRecView v 1.0
// --> MainForm.cs
// Author
// Huang, Yih-Ru Peter 20080323
//
////////////////////////////////////

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using System.Drawing.Imaging;
using System.Collections;
using System.Runtime.InteropServices;
using System.IO;
using System.Diagnostics;

using DShowNET;
using DShowNET.Device;

namespace CRM_CamRec
{
    public partial class MainForm : Form, ISampleGrabberCB
    {
        public MainForm()
        {
            InitializeComponent();
            timer1.Interval = (int)numericUpDownMsec.Value;
        }

        private void MainForm_Activated(object sender, EventArgs e)
        {
            if (!firstActive)
                return;
            firstActive = false;

            if (!DsUtils.IsCorrectDirectXVersion())

```

```

    {
        MessageBox.Show(this, "DirectX 8.1 NOT installed!", "DirectShow.NET",
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        this.Close();
        return;
    }

    if (!DsDev.GetDevicesOfCat(FilterCategory.VideoInputDevice, out capDevices))
    {
        MessageBox.Show(this, "No video capture devices found!",
            "DirectShow.NET",
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        this.Close();
        return;
    }

    DsDevice dev = null;
    if (capDevices.Count == 1)
        dev = capDevices[0] as DsDevice;
    else
    {
        DeviceSelector ds = new DeviceSelector(capDevices);
        ds.ShowDialog(this);
        dev = ds.SelectedDevice;
    }

    if (dev == null)
    {
        this.Close();
        return;
    }

    if (!StartupVideo(dev.Mon))
        this.Close();
}

private void MainForm_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
{
    this.Hide();
    CloseInterfaces();
}

/// <summary> do cleanup and release DirectShow. </summary>
void CloseInterfaces()
{
    int hr;
    try
    {
        {
            if (mediaCtrl != null)
            {

```

```

        hr = mediaCtrl.Stop();
        mediaCtrl = null;
    }

    if (mediaEvt != null)
    {
        hr = mediaEvt.SetNotifyWindow(IntPtr.Zero, WM_GRAPHNOTIFY,
IntPtr.Zero);
        mediaEvt = null;
    }

    if (videoWin != null)
    {
        hr = videoWin.put_Visible(DsHlp.OAFALSE);
        hr = videoWin.put_Owner(IntPtr.Zero);
        videoWin = null;
    }

    baseGrabFlt = null;
    if (sampGrabber != null)
        Marshal.ReleaseComObject(sampGrabber); sampGrabber = null;

    if (capGraph != null)
        Marshal.ReleaseComObject(capGraph); capGraph = null;

    if (graphBuilder != null)
        Marshal.ReleaseComObject(graphBuilder); graphBuilder = null;

    if (capFilter != null)
        Marshal.ReleaseComObject(capFilter); capFilter = null;

    if (capDevices != null)
    {
        foreach (DsDevice d in capDevices)
            d.Dispose();
        capDevices = null;
    }
}
catch
{ }
}

/// <summary> override window fn to handle graph events. </summary>
protected override void WndProc(ref Message m)
{
    if (m.Msg == WM_GRAPHNOTIFY)
    {
        if (mediaEvt != null)
            OnGraphNotify();
        return;
    }
}

```

```

        base.WndProc(ref m);
    }

    /// <summary> graph event (WM_GRAPHNOTIFY) handler. </summary>
    void OnGraphNotify()
    {
        DsEvCode code;
        int p1, p2, hr = 0;
        do
        {
            hr = mediaEvt.GetEvent(out code, out p1, out p2, 0);
            if (hr < 0)
                break;
            hr = mediaEvt.FreeEventParams(code, p1, p2);
        }
        while (hr == 0);
    }

    #region StartupVideo()
    /// <summary> start all the interfaces, graphs and preview window. </summary>
    bool StartupVideo(UCOMIMoniker mon)
    {
        int hr;
        try
        {
            if (!CreateCaptureDevice(mon))
                return false;

            if (!GetInterfaces())
                return false;

            if (!SetupGraph())
                return false;

            if (!SetupVideoWindow())
                return false;

            hr = mediaCtrl.Run();
            if (hr < 0)
                Marshal.ThrowExceptionForHR(hr);

            return true;
        }
        catch
        {
            return false;
        }
    }

    /// <summary> create the user selected capture device. </summary>
    bool CreateCaptureDevice(UCOMIMoniker mon)

```

```

    {
        object capObj = null;
        try
        {
            Guid gbf = typeof(IBaseFilter).GUID;
            mon.BindToObject(null, null, ref gbf, out capObj);
            capFilter = (IBaseFilter)capObj; capObj = null;
            return true;
        }
        catch
        {
            return false;
        }
        finally
        {
            if (capObj != null)
                Marshal.ReleaseComObject(capObj); capObj = null;
        }
    }

    /// <summary> create the used COM components and get the interfaces. </summary>
    bool GetInterfaces()
    {
        Type comType = null;
        object comObj = null;
        try
        {
            comType = Type.GetTypeFromCLSID(Clsid.FilterGraph);
            if (comType == null)
                throw new NotImplementedException(@"DirectShow FilterGraph not
installed/registered!");
            comObj = Activator.CreateInstance(comType);
            graphBuilder = (IGraphBuilder)comObj; comObj = null;

            Guid clsid = Clsid.CaptureGraphBuilder2;
            Guid riid = typeof(ICaptureGraphBuilder2).GUID;
            comObj = DsBugWO.CreateDsInstance(ref clsid, ref riid);
            capGraph = (ICaptureGraphBuilder2)comObj; comObj = null;

            comType = Type.GetTypeFromCLSID(Clsid.SampleGrabber);
            if (comType == null)
                throw new NotImplementedException(@"DirectShow SampleGrabber not
installed/registered!");
            comObj = Activator.CreateInstance(comType);
            sampGrabber = (ISampleGrabber)comObj; comObj = null;

            mediaCtrl = (IMediaControl)graphBuilder;
            videoWin = (IVideoWindow)graphBuilder;
            mediaEvt = (IMediaEventEx)graphBuilder;
            baseGrabFlt = (IBaseFilter)sampGrabber;
        }
    }
}

```

```

        return true;
    }
    catch
    {
        return false;
    }
    finally
    {
        if (comObj != null)
            Marshal.ReleaseComObject(comObj); comObj = null;
    }
}

/// <summary> build the capture graph for grabber. </summary>
bool SetupGraph()
{
    int hr;
    try
    {
        hr = capGraph.SetFiltergraph(graphBuilder);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        hr = graphBuilder.AddFilter(capFilter, "Ds.NET Video Capture Device");
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        DsUtils.ShowCapPinDialog(capGraph, capFilter, this.Handle);
        AMMediaType media = new AMMediaType();
        media.majorType = MediaType.Video;
        media.subType = MediaSubType.RGB24;
        media.formatType = FormatType.VideoInfo; // ???
        hr = sampGrabber.SetMediaType(media);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        hr = graphBuilder.AddFilter(baseGrabFlt, "Ds.NET Grabber");
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        Guid cat = PinCategory.Preview;
        Guid med = MediaType.Video;
        hr = capGraph.RenderStream(ref cat, ref med, capFilter, null, null); //
baseGrabFlt
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        cat = PinCategory.Capture;
        med = MediaType.Video;
        hr = capGraph.RenderStream(ref cat, ref med, capFilter, null,
baseGrabFlt); // baseGrabFlt
    }
    catch
    {
        return false;
    }
}

```

```

        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        media = new AMMediaType();
        hr = sampGrabber.GetConnectedMediaType(media);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);
        if ((media.formatType != FormatType.VideoInfo) || (media.formatPtr ==
IntPtr.Zero))
            throw new NotSupportedException("Unknown Grabber Media Format");

        videoInfoHeader =
(VideoInfoHeader)Marshal.PtrToStructure(media.formatPtr, typeof(VideoInfoHeader));
        Marshal.FreeCoTaskMem(media.formatPtr); media.formatPtr = IntPtr.Zero;

        hr = sampGrabber.SetBufferSamples(false);
        if (hr == 0)
            hr = sampGrabber.SetOneShot(false);
        if (hr == 0)
            hr = sampGrabber.SetCallback(null, 0);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        return true;
    }
    catch
    {
        return false;
    }
}

/// <summary> make the video preview window to show in videoPanel. </summary>
bool SetupVideoWindow()
{
    int hr;
    try
    {
        // Set the video window to be a child of the main window
        hr = videoWin.put_Owner(panelVideo.Handle);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        // Set video window style
        hr = videoWin.put_WindowStyle(WS_CHILD | WS_CLIPCHILDREN);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        // Use helper function to position video window in client rect of owner
        window
        ResizeVideoWindow();
    }
}

```

```

        // Make the video window visible, now that it is properly positioned
        hr = videoWin.put_Visible(DsHlp.OATRUE);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);

        hr = mediaEvt.SetNotifyWindow(this.Handle, WM_GRAPHNOTIFY, IntPtr.Zero);
        if (hr < 0)
            Marshal.ThrowExceptionForHR(hr);
        return true;
    }
    catch
    {
        return false;
    }
}

//private void paneVideo_Resize(object sender, System.EventArgs e)
//{
//    ResizeVideoWindow();
//}

void ResizeVideoWindow()
{
    if (videoWin != null)
    {
        Rectangle cr = panelVideo.ClientRectangle;
        videoWin.SetWindowPosition(0, 0, cr.Right, cr.Bottom);
    }
}
#endregion

private void timer1_Tick(object sender, EventArgs e)
{
    labelDT.Text = getUTCtime();

    if (recording)
    {
        if (filePath == null)
            filePath = @"c:\CRM" + getUTCtime().Remove(8);

        if (!Directory.Exists(filePath))
            Directory.CreateDirectory(filePath);
        int rt = recTime; // (int)numericUpDownMsec.Value * 1000;
        labelRecTime.Text = rt.ToString();
        recTime++;
        //get image;

        int hr;
        if (sampGrabber == null)
            return;
    }
}

```



```

        if (savedArray == null)
        {
            int size = videoInfoHeader.BmiHeader.ImageSize;
            if ((size < 1000) || (size > 16000000))
                return;
            savedArray = new byte[size + 640000];
        }

        //Image old = pictureBox1.Image;
        //pictureBox1.Image = null;
        //if (old != null)
        //    old.Dispose();

        hr = sampGrabber.SetCallback(this, 1);
    }
    else
        recTime = 0;
}

/// <summary> get UTC time (string YYYYMMDDHHMMSS) </summary>
string getUTCtime()
{
    string dt;
    // YYYYMMDD
    dt = DateTime.UtcNow.Year.ToString();
    if (DateTime.UtcNow.Month.ToString().Length == 1)
        dt += "0";
    dt += DateTime.UtcNow.Month.ToString();
    if (DateTime.UtcNow.Day.ToString().Length == 1)
        dt += "0";
    dt += DateTime.UtcNow.Day.ToString();
    //HHMMSS
    if (DateTime.UtcNow.Hour.ToString().Length == 1)
        dt += "0";
    dt += DateTime.UtcNow.Hour.ToString();
    if (DateTime.UtcNow.Minute.ToString().Length == 1)
        dt += "0";
    dt += DateTime.UtcNow.Minute.ToString();
    if (DateTime.UtcNow.Second.ToString().Length == 1)
        dt += "0";
    dt += DateTime.UtcNow.Second.ToString();

    return dt;
}

void OnCaptureDone()
{
    if (sampGrabber == null)
        return;
}

```

```

int hr;
hr = sampGrabber.SetCallback(null, 0);

int w = videoInfoHeader.BmiHeader.Width;
int h = videoInfoHeader.BmiHeader.Height;
if (((w & 0x03) != 0) || (w < 32) || (w > 4096) || (h < 32) || (h > 4096))
    return;

//get Image
int stride = w * 3;
GCHandle handle = GCHandle.Alloc(savedArray, GCHandleType.Pinned);
int scan0 = (int)handle.AddrOfPinnedObject();
scan0 += (h - 1) * stride;

Bitmap H = new Bitmap(w, h, -stride, PixelFormat.Format24bppRgb,
(IntPtr)scan0);

handle.Free();

//pictureBox1.Image = H;

textBoxFileDir.Text = filePath + @"\\" + getUTCtime().Remove(0, 8) +
    DateTime.Now.Millisecond.ToString().Remove(1) + "." +
    comboBoxImageFormat.Text;

switch (comboBoxImageFormat.Text)
{
    case "Raw":
        H.Save(textBoxFileDir.Text, System.Drawing.Imaging.ImageFormat.Bmp);
        break;
    case "Jpg":
        H.Save(textBoxFileDir.Text, System.Drawing.Imaging.ImageFormat.Jpeg);
        break;
    case "Bmp":
        H.Save(textBoxFileDir.Text,
System.Drawing.Imaging.ImageFormat.MemoryBmp);
        break;
    case "Tiff":
        H.Save(textBoxFileDir.Text, System.Drawing.Imaging.ImageFormat.Tiff);
        break;
}

savedArray = null;
}

#region declare members
#region Interface DShowNET Functions
/// <summary> video window interface. </summary>
private IVideoWindow videoWin;
/// <summary> control interface. </summary>
private IMediaControl mediaCtrl;

```

```

    /// <summary> base filter of the actually used video devices. </summary>
    private IBaseFilter          capFilter;
    /// <summary> graph builder interface. </summary>
    private IGraphBuilder        graphBuilder;
    /// <summary> capture graph builder interface. </summary>
    private ICaptureGraphBuilder2 capGraph;
    private ISampleGrabber        sampGrabber;
    /// <summary> event interface. </summary>
    private IMediaEventEx        mediaEvt;
    /// <summary> grabber filter interface. </summary>
    private IBaseFilter          baseGrabFlt;
    /// <summary> structure describing the bitmap to grab. </summary>
    private VideoInfoHeader      videoInfoHeader;
#endregion

    /// <summary> event when callback has finished (ISampleGrabberCB.BufferCB).
</summary>
    private delegate void CaptureDone();

    private const int WM_GRAPHNOTIFY = 0x00008001; // message from graph
    private const int WS_CHILD = 0x40000000; // attributes for video window
    private const int WS_CLIPCHILDREN = 0x02000000;
    private const int WS_CLIPSIBLINGS = 0x04000000;

    /// <summary> list of installed video devices. </summary>
    private ArrayList          capDevices;
    /// <summary> flag to detect first Form appearance </summary>
    private bool                firstActive = true;
    /// <summary> buffer for bitmap data. </summary>
    private byte[]             savedArray;
    /// <summary> file path for saving </summary>
    private string              filePath;
    /// <summary> flag to detect buttonStop/Rec_Click </summary>
    private bool                recording = false;
    /// <summary> recording time mark </summary>
    int                         recTime;

#endregion

#region Functions_Click
    private void buttonAbout_Click(object sender, EventArgs e)
    {
        AboutBox about = new AboutBox();
        about.Show();
    }

    private void buttonExit_Click(object sender, EventArgs e)
    {
        Close();
    }

```

```

private void buttonBrowse_Click(object sender, EventArgs e)
{
    //SaveFileDialog sd = new SaveFileDialog();
    //sd.Title = "Save Images as.....";
    //sd.FileName = "crm";
    //sd.Filter = "Raw Data(*.raw)|*.raw";
    //sd.InitialDirectory = textBoxFileDir.Text;
    //sd.ShowDialog();
    //textBoxFileDir.Text = sd.FileName + getUTCtime() + ".raw";

    FolderBrowserDialog fbd = new FolderBrowserDialog();
    fbd.SelectedPath = textBoxFileDir.Text;
    fbd.Description = "Choose or Create This Task Root Folder";
    if (fbd.ShowDialog() == DialogResult.OK)
    {
        filePath = fbd.SelectedPath + @"\CRM" + getUTCtime().Remove(8);
        textBoxFileDir.Text = filePath;
    }
}

private void buttonRec_Click(object sender, EventArgs e)
{
    buttonAbout.Enabled = true;
    buttonBrowse.Enabled = false;
    buttonExit.Enabled = false;
    buttonRec.Enabled = false;
    buttonStop.Enabled = true;
    buttonPlay.Enabled = false;
    textBoxFileDir.Enabled = false;
    comboBoxImageFormat.Enabled = false;
    numericUpDownMsec.Enabled = false;
    pictureBox1.Visible = true;
    recording = true;

    labelRecTime.Text = "0";
    recTime = 1;
    timer1.Interval = (int)numericUpDownMsec.Value;
}

private void buttonStop_Click(object sender, EventArgs e)
{
    buttonAbout.Enabled = true;
    buttonBrowse.Enabled = true;
    buttonExit.Enabled = true;
    buttonRec.Enabled = true;
    buttonStop.Enabled = false;
    buttonPlay.Enabled = false;
    textBoxFileDir.Enabled = true;
    comboBoxImageFormat.Enabled = true;
    numericUpDownMsec.Enabled = true;
    pictureBox1.Visible = false;
}

```

```

        recording = false;
    }

    private void buttonPlay_Click(object sender, EventArgs e)
    {

    }
    #endregion

    #region ISampleGrabberCB Members
    /// <summary> buffer callback, COULD BE FROM FOREIGN THREAD. </summary>
    int ISampleGrabberCB.BufferCB(double SampleTime, IntPtr pBuffer, int BufferLen)
    {
        if (savedArray == null)
        {
            Trace.WriteLine("ISampleGrabberCB.BufferCB (savedArray == null)");
            return 0;
        }

        if ((pBuffer != IntPtr.Zero) && (BufferLen > 1000) && (BufferLen <=
savedArray.Length))
            Marshal.Copy(pBuffer, savedArray, 0, BufferLen);
        else
            Trace.WriteLine("Grab Failed!!");
        this.BeginInvoke(new CaptureDone(this.OnCaptureDone));
        return 0;
    }

    public int SampleCB(double SampleTime, IMediaSample pSample)
    {
        throw new Exception("The method or operation is not implemented.");
    }

    #endregion

    private void panelVideo_Paint(object sender, PaintEventArgs e)
    {

    }

    }
}

```

Appendix B: CRM System Raw Data to CSV Tool C# Code

```
////////////////////////////////////
// CRM System Raw Data to CSV format
//
// MS-Visual Studio 2005 C#
// CRM Data to CSV v 1.0
// --> Form1.cs
// Author
// Huang, Yih-Ru Peter 20060705
//
////////////////////////////////////

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace CRM_Data_to_CSV
{
    public partial class CRM : Form
    {
        public CRM()
        {
            InitializeComponent();
        }

        private void button_OK_Click(object sender, EventArgs e)
        {
            Unpack();
        }

        private void button_EXIT_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void button_FileName_Click(object sender, EventArgs e)
        {
            OpenFileDialog OpenFile = new OpenFileDialog();
            OpenFile.ShowDialog();
            textBox_FileDir.Text = OpenFile.FileName;
        }

        private void Unpack ()
        {

```

```

FileStream CRMdata = new FileStream(textBox_FileDir.Text, FileMode.Open);
StreamWriter CRMCSV = File.CreateText(textBox_FileDir.Text + ".csv");

char side, cam;
uint utc, utc_usec, numPoints,
    x_b, y_b, b_b;
double GPSsec; // GPS week sec Sunday 00:00:00 ++
double utcu;
double x, y, b;

BinaryReader br = new BinaryReader(CRMdata);

side = br.ReadChar();
cam = br.ReadChar();

while(br.BaseStream.Length > br.BaseStream.Position)
{
    utc = br.ReadUInt32();
    utc_usec = br.ReadUInt32();
    utcu = utc + utc_usec * 1e-6;

    DateTime dt = new DateTime(1970, 1, 1, 0, 0, 0).AddSeconds(utc);
    GPSsec = dt.Hour * 3600 + dt.Minute * 60 + dt.Second + utc_usec * 1e-6;
    switch (dt.DayOfWeek.ToString())
    {
        case "Monday":
            GPSsec += 86400*1;
            break;
        case "Tuesday":
            GPSsec += 86400*2;
            break;
        case "Wednesday":
            GPSsec += 86400*3;
            break;
        case "Thursday":
            GPSsec += 86400*4;
            break;
        case "Friday":
            GPSsec += 86400*5;
            break;
        case "Saturday":
            GPSsec += 86400*6;
            break;
    }

    numPoints = br.ReadUInt16();

    for (int j = 0; j < numPoints; j++)
    {
        x_b = br.ReadUInt16();
        y_b = br.ReadUInt16();
    }
}

```

```

        b_b = br.ReadUInt16();

        x = 1280 - (x_b / 51.0); /* rescale back */
        //x = x_b / 51.0; /* rescale back */
        y = y_b / 63.0; // 1024 or 1030???
        //y = 1030 - (y_b / 63.0);
        b = b_b; /* 8.0;
        CRMCSV.WriteLine(GPSsec + "," + x + "," + y + "," + b );
    }
}

CRMCSV.Close();
CRMdata.Close();
br.Close();
}
}
}

```


Appendix C: CRM_BoxRL_ProcessAutoCal.m MATLAB Code

CRM_BoxRL_ProcessAutoCal.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%   Huang, Yih-Ru  20080410
%%
%%   INPUT:  *Truth.csv      truth(Ashtech)  ENU raw data
%%           Kxxx1.csv
%%           Kxxx2.csv
%%
%%   OUTPUT: KxxxYYYYMMDD.csv  (E,N,U  meters)
%%   KOKC
%%           17R_T           Dab = d1 + d2
%%                           d1 => th1,      d2 => th2
%%                           d1/D = tan(th1),  d2/D = tan(th2)
%%
%%           KOUN2r          KOUN1r          d1 = ----- Dbox
%%                           tan(th1)+tan(th2)
%%
%%           KOUN2  CRL  KOUN1          d1      d2
%%                           D = ----- = -----
%%                           tan(th1)   tha(th2)
%%
%%           17R_E
%%
%%   OmmVision- OV09121
%%   Array Size = 1280*1024 (SXGA)
%%   PixelSize = 5.2 um * 5.2 um
%%   ImageArea = 6.66 mm * 5.32 mm (1/4" CCD)
%%   http://www.ovt.com/data/parts/pdf/OV9620_PB%20(2.7).pdf
%%
%%   Marshall Electronics Len
%%   Focal   Angular Field
%%   Length of View
%%   -----
%%   V-4308  8mm      27-19-37 (H-V-D Degree)
%%   V-4350  50mm     04_03_05 (H-V-D Degree) @ 1/4" CCD
%%   http://www.mars-cam.com/lenses/ccd_cmos/43fix.html
%%
%%   1 nautical mile(NM) = 6076.11549 feet
%%   1 feet = 0.3048 meters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;
clear all;
format long g;

[inFile, path] = uigetfile('*.csv', 'Looking for the Ashtech file,
"yyymmddx_truth.csv" !!!!');
AshtechNENU = csvread([path inFile]);

[inFile, path] = uigetfile('*.csv', 'Looking for the CRM Box R file,
"yyymmddxKxxx1.csv" !!!!');
KxxR = csvread([path inFile]);
%KxxR = KxxR(find(KxxR(:,2)<640 & KxxR(:,3)<512),:);
KxxxR = KxxR(find(KxxR(:,4)>1000),:);

[inFile, path] = uigetfile('*.csv', 'Looking for the CRM Box L file,
"yyymmddxKxxx2.csv" !!!!');
KxxL = csvread([path inFile]);
%KxxL = KxxL(find(KxxL(:,2)>640 & KxxL(:,3)<512),:);
```

```

KxxxL = KxxL(find(KxxL(:,4)>1000),:);

DRL = 222.6725;           % 1R to 1L
DR  = 111.1621;         % 1R to CRL
DL  = 111.5103;         % 1L to CRL
Dbt = 2381.8167;        % 17T to CRL

DbtRV = 1.509;          % 35R_T to BoxR High
DbtLV = 0.761;          % 35R_T to BoxL High

CHR = 0;    CVR = 0;      %cal Horizontal Vertical
CHL = 0;    CVL = 0;
K = 14;     % GPS UTC Times Offset 13sec NOW

GridHR = 0:1280/7.80:1280;
GridHL = fliplr(abs(-1280:1280/7.80:0));
GridV = 0:1024/(6.24*5):1024;

figure(1)
subplot(1,2,2); plot(KxxR(:,2),KxxR(:,3), '.'); %-90
set(gca,'XTick',GridHR, 'YTick',GridV);
axis([0 640 0 800]); grid on; %axis([0 1280 0 1024]);
title('CRM KOUN Flight Test (CRM Right View)')
xlabel('CCD Pixel (1280 x 1024)')
subplot(1,2,1); plot(KxxL(:,2),KxxL(:,3), '.'); %-135
set(gca,'XTick',GridHL, 'YTick',GridV);
axis([640 1280 0 800]); grid on; %axis([0 1280 0 1024]);
title('CRM KOUN Flight Test (CRM Left View)')
xlabel('CCD Pixel (1280 x 1024)')

figure(2); hold on
plot3(AshtechNENU(:,2), AshtechNENU(:,3), zeros(length(AshtechNENU), 1), 'k:');
plot3(AshtechNENU(:,2), 100*ones(length(AshtechNENU), 1), AshtechNENU(:,4), 'k--');
plot3(AshtechNENU(:,2), AshtechNENU(:,3), AshtechNENU(:,4), 'b. ');
view(5, 25)
title('CRM KOUN Flight Test (Truth Data)')
xlabel('ENU : East(meter)')
ylabel('ENU : North(meter)')
zlabel('ENU : Up(meter)')
grid on; hold off

AKA = [AshtechNENU... %AKA[1:4] Ashtech KxxxR/L Angle
atan( (DR +AshtechNENU(:,3))./(Dbt+AshtechNENU(:,2)) ) *180/pi...
atan( (DbtRV+AshtechNENU(:,4))./(Dbt+AshtechNENU(:,2)) ) *180/pi...
atan( (DL -AshtechNENU(:,3))./(Dbt+AshtechNENU(:,2)) ) *180/pi...
atan( (DbtLV+AshtechNENU(:,4))./(Dbt+AshtechNENU(:,2)) ) *180/pi];
AKA = [AKA AKA(:,5)./AKA(:,7) AKA(:,6)./AKA(:,8) AKA(:,2)/0.3048]; %AKA[9:11]

KRA = [KxxxR... %KRA[1:4] Kxxx RightBox Angel
(1280/2-KxxxR(:,2))/1280 * 7.720 + CHR... %KRA(:,5)
KxxxR(:,3)/1024 * 6.176 + CVR]; %KRA(:,6)
KLA = [KxxxL... %KLA[1:4] Kxxx LeftBox Angel
(KxxxL(:,2)-1280/2)/1280 * 7.720 + CHL... %KLA(:,5)
KxxxL(:,3)/1024 * 6.176 + CVL]; %KLA(:,6)

for i=1:length(AKA)-1
I = find(KRA(:,1)> AKA(i,1)-K & KRA(:,1)< AKA(i+1,1)-K);
KRA(I,7) = AKA(i,2); KRA(I,8) = AKA(i,5); KRA(I,9) = AKA(i,6);
I = find(KLA(:,1)> AKA(i,1)-K & KLA(:,1)< AKA(i+1,1)-K);
KLA(I,7) = AKA(i,2); KLA(I,8) = AKA(i,7); KLA(I,9) = AKA(i,8);
end

%cal Horizontal
figure(3)
subplot(2,1,1); plot(KRA(:,1), KRA(:,5), 'b.', KRA(:,1), KRA(:,8), 'r')
axis([KLA(1,1) KLA(end,1) 0 3.90]);

```

```

subplot(2,1,2);      plot(KLA(:,1), KLA(:,5), 'b.', KLA(:,1), KLA(:,8), 'r')
axis([KLA(1,1) KLA(end,1) 0 3.90]);

%cal Vertical
figure(4)
subplot(2,1,1);      plot(KRA(:,1), KRA(:,6), 'b.', KRA(:,1), KRA(:,9), 'r')
axis([KLA(1,1) KLA(end,1) 0 3]);

subplot(2,1,2);      plot(KLA(:,1), KLA(:,6), 'b.', KLA(:,1), KLA(:,9), 'r')
axis([KLA(1,1) KLA(end,1) 0 3]);      %

KRAC=KRA(find(abs(KRA(1:end/2,5)-KRA(1:end/2,8))<0.3),:);
KRAC=KRA(find(abs(KRA(1:end/2,6)-KRA(1:end/2,9))<0.3),:);
KLAC=KLA(find(abs(KLA(1:end/2,5)-KLA(1:end/2,8))<0.3),:);
KLAC=KLA(find(abs(KLA(1:end/2,6)-KLA(1:end/2,9))<0.3),:);

CHR = mean(KRAC(:,8)-KRAC(:,5))      %cal Horizontal
CVR = mean(KRAC(:,9)-KRAC(:,6))      %cal Vertical
CHL = mean(KLAC(:,8)-KLAC(:,5))
CVL = mean(KLAC(:,9)-KLAC(:,6))

KRA = [KxxxR...
        (1280/2-KxxxR(:,2))/1280 * 7.720 + CHR...      %KRA(:,5)
        KxxxR(:,3)/1024 * 6.176 + CVR];              %KRA(:,6)
KLA = [KxxxL...
        (KxxxL(:,2)-1280/2)/1280 * 7.720 + CHL...      %KLA(:,5)
        KxxxL(:,3)/1024 * 6.176 + CVL];              %KLA(:,6)

for i=1:length(AKA)-1
    I = find(KRA(:,1)> AKA(i,1)-K & KRA(:,1)< AKA(i+1,1)-K);
    KRA(I,7) = AKA(i,2);    KRA(I,8) = AKA(i,5);    KRA(I,9) = AKA(i,6);
    I = find(KLA(:,1)> AKA(i,1)-K & KLA(:,1)< AKA(i+1,1)-K);
    KLA(I,7) = AKA(i,2);    KLA(I,8) = AKA(i,7);    KLA(I,9) = AKA(i,8);
end

%cal Horizontal
figure(5)
subplot(2,1,1);      plot(KRA(:,1), KRA(:,5), 'b.', KRA(:,1), KRA(:,8), 'r')
axis([KLA(1,1) KLA(end,1) 0 3.90]);      %axis([351660 351870 0 3.90]);
legend('CRM Right CCD Angle Measurement','Truth System Angle Measurement', 2)
title('CRM System Right CCD and Truth System Horizontal Angle Measurement')
xlabel('UTC Seconds')
ylabel('Angle Degrees')
grid on
subplot(2,1,2);      plot(KLA(:,1), KLA(:,5), 'b.', KLA(:,1), KLA(:,8), 'r')
axis([KLA(1,1) KLA(end,1) 0 3.90]);      %axis([351660 351870 0 3.90]);
legend('CRM Left CCD Angle Measurement','Truth System Angle Measurement', 2)
title('CRM System Left CCD and Truth System Horizontal Angle Measurement')
xlabel('UTC Seconds')
ylabel('Angle Degrees')
grid on

%cal Vertical
figure(6)
subplot(2,1,1);      plot(KRA(:,1), KRA(:,6), 'b.', KRA(:,1), KRA(:,9), 'r')
axis([KLA(1,1) KLA(end,1) 0 3]);      %axis([351660 351870 0 3.12]);
legend('CRM Right CCD Angle Measurement','Truth System Angle Measurement', 3)
title('CRM System Right CCD and Truth System Vertical Angle Measurement')
xlabel('UTC Seconds')
ylabel('Angle Degrees')
grid on
subplot(2,1,2);      plot(KLA(:,1), KLA(:,6), 'b.', KLA(:,1), KLA(:,9), 'r')
axis([KLA(1,1) KLA(end,1) 0 3]);      %axis([351660 351870 0 3.12]);

```

```

legend('CRM Left CCD Angle Measurement','Truth System Angle Measurement', 3)
title('CRM System Left CCD and Truth System Vertical Angle Measurement')
xlabel('UTC Seconds')
ylabel('Angle Degrees')
grid on %axis([352600 352810 0 3.12]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CRM_E = E = D - Dbt;
% CRM_E = (DR+DL) / (tan(thHR)+tan(thHL)) - Dbt
% CRM_N = (DL*tan(thHR) - DR*tan(thHL)) ./ (tan(thHR)+tan(thHL))
% CRM_U = tan(thV1) * Crm_E = tan(thV2) * Crm_E
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
l = 1;
for i = 1:length(AshtechNENU)-1
    IR = find(KRA(:,1) > AshtechNENU(i,1)-K & KRA(:,1) < AshtechNENU(i+1,1)-K );
    IL = find(KLA(:,1) > AshtechNENU(i,1)-K & KLA(:,1) < AshtechNENU(i+1,1)-K );
    for j = 1:length(IR)
        for k = 1:length(IL)
            if ( abs(KRA(IR(j),6)-KLA(IL(k),6)) < 0.2 )
                thHR = KRA(IR(j), 5) * pi/180;
                thHL = KLA(IL(k), 5) * pi/180;
                thV1 = KRA(IR(j), 6) * pi/180;
                CRM_E = (DR+DL) / (tan(thHR)+tan(thHL)) - Dbt;
                if CRM_E < 1852*3 && CRM_E > 0
                    CRM_N = (DL*tan(thHR) - DR*tan(thHL)) ./ (tan(thHR)+tan(thHL));
                    CRM_U = tan(thV1) * (CRM_E + Dbt) ;%+/- DbtRV;
                    if CRM_U > 10
                        CRM_ENU(l,:) = [AshtechNENU(i,1) CRM_E CRM_N CRM_U
KRA(IR(j), 1) KLA(IL(k), 1)];
                        l = l+1;
                    end
                end
            end
        end
    end
end

i=1; j=1;
while i<length(CRM_ENU)
    I = find(CRM_ENU(:,1) == CRM_ENU(i,1));
    [B II] = max(CRM_ENU(I,5).*CRM_ENU(I,6));
    CRM_ENUr(j,:) = [CRM_ENU(I(II),:) ...
                    CRM_ENU(I(II),5)-CRM_ENU(I(II),1)+K ...
                    CRM_ENU(I(II),6)-CRM_ENU(I(II),1)+K ...
                    CRM_ENU(I(II),5)-CRM_ENU(I(II),6) ];

    i = i + length(I);
    j = j+1;
end

figure(7); hold on
plot3(CRM_ENUr(:,2), CRM_ENUr(:,3), CRM_ENUr(:,4), 'r.')
plot3(AshtechNENU(:,2), AshtechNENU(:,3), AshtechNENU(:,4), 'b-')
plot3(CRM_ENUr(:,2), CRM_ENUr(:,3), zeros(length(CRM_ENUr), 1), 'k.')
plot3(CRM_ENUr(:,2), 100*ones(length(CRM_ENUr), 1), CRM_ENUr(:,4), 'k.')
plot3(CRM_ENUr(:,2), CRM_ENUr(:,3), CRM_ENUr(:,4), 'r.')
plot3(AshtechNENU(:,2), AshtechNENU(:,3), zeros(length(AshtechNENU), 1), 'k:')
plot3(AshtechNENU(:,2), 100*ones(length(AshtechNENU), 1), AshtechNENU(:,4), 'k--')
plot3(AshtechNENU(:,2), AshtechNENU(:,3), AshtechNENU(:,4), 'b-')
legend('CRM','Truth')
axis([-500 8000 -200 +200 0 400]);
view(5, 25)
title('KOUN CRM Time and Space Position Information (TSPI)')
xlabel('ENU : East(meter)')
ylabel('ENU : North(meter)')
zlabel('ENU : Up(meter)')

```

```

grid on; hold off

for i=1:length(CRM_ENUr)
    I = find(AshtechNENU(:,1) == CRM_ENUr(i,1));
    SE_E = (CRM_ENUr(i,2)-AshtechNENU(I,2))^2;
    SE_N = (CRM_ENUr(i,3)-AshtechNENU(I,3))^2;
    SE_U = (CRM_ENUr(i,4)-AshtechNENU(I,4))^2;
    SE = (SE_E + SE_N + SE_U) ^ 0.5;
    CRM_SE(i, :) = [SE SE_E^0.5 SE_N^0.5 SE_U^0.5];
end

CRM_SEm = mean(CRM_SE)
CRM_SEs = std(CRM_SE)

figure(8); hold on
plot(CRM_ENUr(:,2)/1852, CRM_ENUr(:,3), 'r.')
plot(AshtechNENU(:,2)/1852, AshtechNENU(:,3), 'k:')
axis([0 3 -50 50]);
hold off

figure(9); hold on
plot(CRM_ENUr(:,2)/1852, CRM_ENUr(:,4), 'r.')
plot(AshtechNENU(:,2)/1852, AshtechNENU(:,4), 'k:')
axis([0 3 0 600]);
hold off

figure(10)
plot(abs(CRM_ENUr(:,7:9)))
axis([0 100 0 0.2]);
legend('Truth - CRM Box Right','Truth - CRM Box Left','CRM Boxes Right - Left', 2)
title('CRM Boxes and Truth Time Difference')
xlabel('')
ylabel('Seconds')
grid on

##### Write to the file
#####
outFile1 = ['CRM_' inFile(1:4) inFile(10:17) '.csv'];
outFid1=fopen(outFile1, 'w+');
if ~outFid1
    error('Unable to open file.');
```

```

end

fprintf(outFid1, '%12.14f, %12.14f,%12.14f,%12.14f \n', CRM_ENUr(:,1:4));
fclose(outFid1);
```

Appendix D: CRM-Box System Watchdog C code

Watchdog-revB.c

```
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
//
// Watchdog-revB.c
//
// Owner: Patrick Macklin
// Modified: 2/5/2007
//
// Owner: Yih-Ru Huang
// Modified: 20070707
//
// Notes: This was coded to coincide with the first hardware upgrade of the
// Collision Risk Monitor Watchdog unit to the ATmega8 microprocessor. It is
// not to be confused with the old hardware which also used revision A for
// its first version.
//
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
//
// Pinout for the ATmega8 microcontroller on the CRM board
//
//          PC0 (ADC0) ---- Battery voltage
//          PC1 (ADC1) ---- PV voltage
//          PC2 (ADC2) ---- Analog signal strength from the modem
//
//          PD0 (RDX) ---- Connected through Max 2323 to COMP's TDX (Debug)
//          PD1 (TDX) ---- Connected through Max 2323 to Comp's RDX (CUP)
//          PD3 ---- Master or slave select switch
//          PD4 ---- Modem reset
//          PD5 ---- Modem In Range
//
//          PB0 ---- Micro Controller on and working LED
//          PB1 ---- CPU power (negative logic)
//          PB2 ---- Relay ON or OFF this is an output
//          PB3 ---- CPU power indicator LED
//          PB4 ---- In range ON LED
//          PB5 ---- Master ON LED (Debugr)
//
// CRM code for the AtMega8 microcontroller follows
//          www.atmel.com/dyn/resources/prod_documents/doc2486.pdf
//
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////

// Set the internal clock to 1 MHz
```

```

#define F_CPU 1000000

// These are all here to get some of the functions to work such as
// the printf calls
// avr/*.h files are located in the winAVR directory, not the AVR studio dir
#include <avr/io.h>
#include <avr/interrupt.h> //New version of
signal.h
#include <inttypes.h>
#include <util/delay.h>
#include <stdio.h>

////////////////////////////////////
////////////////////////////////////
//A header file used with (or in) the makefile?
//Also contains some util functions
////////////////////////////////////
////////////////////////////////////
#include "CRMWD_lib.h"

#ifndef __CRMWD_lib_h
# warning "CRMWD_lib.h failed to include"
#endif

#define __u8 unsigned char
#define __u16 unsigned short

//Take an A/D reading
unsigned int ADC_conversion(unsigned char chan);
void collectData(void);
void printStatus(void);
void chargeBattery(void);
void manageCPU(void);
void power_up(void);
void masterOrSlave(void);
//inline void sleep_sanity(void);

// These test the internal data memory
inline unsigned char is_battery_chargeable(void);
inline unsigned char is_battery_full(void);
inline unsigned char is_battery_low(void);
inline unsigned char enough_power(void);

volatile __u16 last_cpu_refresh;
volatile __u16 offline_ticks;
static __u16 charge_ticks;
static __u16 cooldown_ticks;
static __u16 panelVolts;
static __u16 battVolts;

```

```

static __u8 data_mask;
static __u8 battery_chargeable;
static __u8 battery_full;
static __u8 battery_low;
static __u8 battery_turn_on;
static __u8 modem_in_range;
static __u8 battery_state;
static __u8 CPU_state;
static __u8 Print_Delay = 10;

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
int main(void)
{
    DDRB = 0xFF; //Set
Port B to be all outputs
    DDRD = 0x00; //Set
Port D to be all inputs
    PORTB = 0b00000011;
    //Initialized PB outputs cpuOff(),close_relay()

    ioinit();
    //Initialize PD0/PD1 as a serial device, p.136
    init_ADC();
    //Initialize the ADC
    init_TIOVF();
    //Initialize the Timer Overflow
    sei();
    //Enable global interrupts

    _delay_ms(100);
    printf("\r\n\r\n\r\n CRM Watchdog-revB Starting Up . . . \r\n\r\n");
    flashLEDs(250); flashLEDs(250); flashLEDs(250);
    PORTB = 0b00000010;
    //Initialized PB outputs cpuOff(),open_relay()

    PORTB |= MCU_POW_LED; //Show the power
light for the MCU

    battery_state = BATT_DRAINING;
    CPU_state = CPU_SLEEPING;
    charge_ticks = 0;
    cooldown_ticks = 0;

    battery_chargeable = 0x00; //Reset all internal
data tracking
    battery_full = 0x00;
    battery_low = 0x00;
    modem_in_range = 0x00;

```



```

data_mask =                0x01;                //Reset *

//Watchdog unit will wait 10 seconds minimum before attempting to power CPU
offline_ticks = 10;

// CPU left disabled and the CPU manager will decide to turn it on or not

while(1)
{
}

ISR(TIMER1_OVF_vect)
{
    TIFR &= (1<<TOV1);                //static
    int T1OVF_count = 0;
    TIM16_addTCNT1( 32000 );

    /*if(++Tick_Count == 30)
    {printf("Timer 1 Overflow 30 times.\r\n");    Tick_Count = 0;} */

    masterOrSlave();                //Set
    the debug light on if master, off if slave
    collectData();
    chargeBattery();
    manageCPU();

    if(Print_Delay == 10)                // Print the
    status every 10 sec
    {
        printStatus();
        Print_Delay = 0;
    }
    ++Print_Delay;
    printf(". ");
}

void masterOrSlave(void)
{
    if (is_master())
        PORTB |= DEBUG_LED;
    else
        PORTB &= ~DEBUG_LED;
}

void collectData(void)
{
    panelVolts = ADC_conversion(PANEL_CHAN); // Volts ==> [ 48 = 1.0 V ]
    battVolts = ADC_conversion(BATT_CHAN);
}

```

```

        //Is battery chargeable? ==> solar volts > batt volts + 0.6V
        if (battVolts < BATTERY_FULL_VOLTS && (battVolts + ENGAGE_RELAY_DIFF<
panelVolts ) )
            battery_chargeable = 0xFF; //|= data_mask;
        else
            battery_chargeable = 0x00; //&= ~data_mask;

        //Is battery full? ==> batt volts > 13.8 V
        if(battVolts > BATTERY_FULL_VOLTS)
            battery_full = 0xFF; //|= data_mask;
        else
            battery_full = 0x00; //&= ~data_mask;

        //Is battery low? ==> batt volts < 10.75 V
        if(battVolts < MINIMUM_VOLTS)
            battery_low = 0xFF; //|= data_mask;
        else
            battery_low = 0x00; //&= ~data_mask;

        //Enough power to power up? ==> batt volts < 11.75 V
        if(battVolts > POWER_UP_VOLTS)
            battery_turn_on = 0xFF; //|= data_mask;
        else
            battery_turn_on = 0x00; //&= ~data_mask;

        //Is modem communicating with partner box?
        if(is_in_range())
        {
            modem_in_range = 0xFF; //&= ~data_mask;
            PORTB |= IN_RANGE_LED;
        }
        else
        {
            modem_in_range = 0x00; //|= data_mask;
            PORTB &= ~IN_RANGE_LED;
        }

        //Cycle the data mask
        if (data_mask == 0b10000000)
            data_mask = 0x01; // cycle back to zero
        else
            data_mask <<= 1; // left shift by one
    }

void printStatus(void)
{
    int BV, PV;
    BV = battVolts/0.70794;//0.4655;//47.038
    PV = panelVolts/0.70794;//0.4655;
    printf("\r\n\r\n\r\n");
    // printf("*****");
}

```

```

//      *   CRM WatchDog Master Unit Status.   *
//      printf("\r\n*   CRM WatchDog ");
//      if(is_master())
//          printf("Master Unit Status.   *\r\n");
//      else
//          printf("Slave Unit Status.   *\r\n");
//      printf("*****\r\n");
//      printf("Battery: %d0(mV) Solar Panel: %d0(mV) \r\n", BV, PV);
//      printf("Battery: %i Solar Panel: %i \r\n", battVolts, panelVolts);
//      printf("Battery Chargeable: %x \r\n", battery_chargeable);
//      printf("Battery Full: 0x%x Battery Low: 0x%x\r\n", battery_full,
battery_low);
//      printf("Enough Power: 0x%x \r\n", battery_turn_on);
//      printf("Modem Range: 0x%x \r\n", modem_in_range);
//      printf("Data Mask: 0x%x \r\n", data_mask);

//      printf("Battery State: ");
//      switch(battery_state)
//      {
//          case BATT_DRAINING:
//              printf("Battery Draining");
//              break;
//          case BATT_CHARGING:
//              printf("Charging %d sec", charge_ticks);
//              break;
//          case BATT_COOLDOWN:
//              printf("Cooling Down %d sec", cooldown_ticks);
//              break;
//          default:
//              printf("error");
//      }
//      printf(".\r\n");

printf("CPU: ");
switch(CPU_state)
{
    case CPU_SLEEPING:
        printf("Sleeping %d sec", offline_ticks);
        break;
    case CPU_TIMED_OUT:
        printf("Timed Out. Retry %d sec", offline_ticks);
        break;
    case CPU_LOW_POWER:
        printf("Low Power. Restart %d sec", offline_ticks);
        break;
    case CPU_ON:
        printf("On. Last Refresh %d sec Ago", last_cpu_refresh);
        break;
    case CPU_POWERING_UP:
        printf("Powering Up %d sec", offline_ticks);
        break;
}

```

```

        case CPU_OFF:
            printf("Turned Itself Off");
            break;
        default:
            printf("error");
            break;
    }
    printf(".\r\n");
}

void chargeBattery(void)
{
    switch(battery_state)
    {
        case BATT_DRAINING:
            if( battery_chargeable )
            {
                battery_state = BATT_CHARGING;
                charge_ticks = CHARGE_INTERVAL;
                close_relay();
            }
            break;
        case BATT_CHARGING:
            if(is_battery_full())
            {
                battery_state = BATT_COOLDOWN;
                cooldown_ticks = BATT_FULL_WAIT;
                open_relay();
            }
            else if(--charge_ticks <= 0)
            {
                battery_state = BATT_COOLDOWN;
                cooldown_ticks = BATT_CHARGE_CD;
                open_relay();
            }
            break;
        case BATT_COOLDOWN:
            if( --cooldown_ticks <= 0)
            {
                battery_state = BATT_DRAINING;
            }
            break;
        default:
            battery_state = BATT_DRAINING;
            break;
    }
}

void manageCPU(void)
{
    //printf("manageCPU() entered.\r\n");
}

```

```

switch(CPU_state)
{
    case CPU_ON:
        if (++last_cpu_refresh > WATCHDOG_TIMEOUT)
        {
            // CPU has stopped communicating
            // sleep it for a small time then reboot it
            cpuOff();
            last_cpu_refresh = 0;
            offline_ticks = WATCHDOG_SLEEP;
            CPU_state = CPU_TIMED_OUT;
        }
        else if (offline_ticks > 0)
        {
            // This can occur from a recieved sleep command from the
            // or an erroneous sleep state.
            cpuOff();
            last_cpu_refresh = 0;
            CPU_state = CPU_SLEEPING;
        }
        else if (is_battery_low())
        {
            cpuOff();
            last_cpu_refresh = 0;
            if (offline_ticks < WATCHDOG_SLEEP)
                offline_ticks = WATCHDOG_SLEEP;
            CPU_state = CPU_LOW_POWER;
        }

        // Slave sleeps when master isn't communicating
        // if ( !(is_master()) && (modem_in_range == 0xFF) )
        // {
        //     cpuOff();
        //     CPU_state = CPU_SLEEPING;
        // }

        break;
    case CPU_LOW_POWER:
        if(offline_ticks > OFFLINE_MAX)
            offline_ticks = OFFLINE_MAX;

        if (offline_ticks > 0)
            --offline_ticks;
        else //428
            power_up();
        break;
    case CPU_TIMED_OUT:
    case CPU_SLEEPING:
        if(offline_ticks > OFFLINE_MAX)

```

```

        offline_ticks = OFFLINE_MAX;

        if (offline_ticks > 0)
            --offline_ticks;
        else
            power_up();

        if(is_battery_low())
            CPU_state = CPU_LOW_POWER;
        break;
case CPU_POWERING_UP:
    // CPU needs time to power up before sending timeout refreshers
    if (offline_ticks > 0)
        --offline_ticks;
    else
    {
        CPU_state = CPU_ON;
        last_cpu_refresh = 0;
    }
    break;
case CPU_OFF:
    // CPU has decided to turn itself off
    // do nothing, only a hard reset can recover
default:
    break;
    }
}

////////////////////////////////////
////////////////////////////////////
// power_up()
// makes the decision to power up the computer
// returns 0x01 if cpu powers up and 0x00 if it does not
////////////////////////////////////
////////////////////////////////////
void power_up(void)
{
    if (enough_power()) // Power
up if there's power to be had
    {
        if( !(is_master() || is_in_range()) ) // This device is the slave

            return; //
Slave will off unless in communication with master

        cpuOn();
        // Master wakes up regardless so slave will find it

        offline_ticks = WATCHDOG_TO_PWR;
        last_cpu_refresh = 0;

```

```

        CPU_state = CPU_POWERING_UP;
    }
    else
    // Continue sleeping till the battery is charged
    {
        offline_ticks = WATCHDOG_SLEEP;
    }
}

//inline void sleep_sanity(void)
//{
//    if(offline_ticks > OFFLINE_MAX)
//        offline_ticks = OFFLINE_MAX;
//}

inline unsigned char is_battery_chargeable(void)
{
    return battery_chargeable == 0xFF;
}

inline unsigned char is_battery_full(void)
{
    return battery_full == 0xFF;
}

inline unsigned char is_battery_low(void)
{
    return battery_low == 0xFF;
}

inline unsigned char enough_power(void)
{
    return battery_turn_on == 0xFF;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
// ISR(USART_RXC_vect)
// Interrupt handler for RS232 communications. The CPU can sleep and wake
// the watchdog unit based on the weather conditions obtained from the
// server. The watchdog accepts one of three commands following a 4-byte
// header.
//
// 0xdeadface  Rx beginning signature
// 0x55        kill power
// 0xEE        watchdog refresh
// 0xAA        Sleep. Next two bytes are wake delay
//
// The USART has to be initialized in the ioinit() function in oulib.c

```

```

////////////////////////////////////
////////////////////////////////////
ISR(USART_RXC_vect)
{
    static __u8 serial_state;
    static __u8 rx;

    //clear interrupt bit by reading the USART data register
    rx = UDR;
    // Do not need a wait loop because the interrupt has been triggered
    //printf("%x ", rx);
    switch (serial_state)
    {
        case 0:
            if (rx == 0xDE)
                serial_state++;
            else
                serial_state = 0;
            break;
        case 1:
            if (rx == 0xAD)
                serial_state++;
            else
                serial_state = 0;
            break;
        case 2:
            if (rx == 0xFA)
                serial_state++;
            else
                serial_state = 0;
            break;
        case 3:
            if (rx == 0xCE)
                serial_state++;
            else
                serial_state = 0;
            break;
        case 4:
            // header recieved, check for command
            if (rx == 0x55)
            {
                printf("Kill Power command recieved.\r\n");
                //          cpuOff();
                //          serial_state = 0;
                //          CPU_state = CPU_OFF;
                break;
            }
            else if (rx == 0xEE)
            {
                printf("Timeout refresh recieved.\r\n");
            }
    }
}

```



```

        last_cpu_refresh = 0;           // Reset timeout
count
        offline_ticks = 0;
        serial_state = 0;
//UCSRB &= ~(1<<RXCIE);
    }
    else if (rx == 0xAA)
    {
        printf("Sleep command recieved.\r\n");    //next
two bytes=wakeup time.
        serial_state++;
        break;
    }
    else
    {
        printf("Erroneous CPU message(invalid command, do
nothing).\r\n");
        serial_state = 0;
        break;
    }
    break;
case 5:
    offline_ticks = rx << 8;
    serial_state++;
    break;
case 6:
    offline_ticks |= rx;
    //printf("Sleep %d sec.\r\n", offline_ticks);
    serial_state = 0;
    break;
default:
    serial_state = 0;           //
invalid serial_state, do nothing
    break;
}
}

```

CRMWD_lib.h

```
////////////////////////////////////
//////////
//
// CRMWD_lib.h
//
// Owner: Yih-Ru Huang
// Modified: 20070707
//
////////////////////////////////////
//////////

#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#ifndef __CRMWD_lib_h
#define __CRMWD_lib_h
#endif

#define F_CPU 8000000

#ifndef F_CPU
#error "F_CPU must be defined"
#endif

#if (F_CPU == 16000000)
#define MS_SHIFT 4
#define MS_GATE 0x0f
#elif (F_CPU == 8000000)
#define MS_SHIFT 8
#define MS_GATE 0xff
#else
#error "Bad F_CPU specification"
#endif

////////////////////////////////////
//////////
// Constant definitions from CRMWD_lib.h
////////////////////////////////////
//////////

// A/D converter values. Measured from voltages and A/D readouts [ 70.794 == 1.0V ]
#define ENGAGE_RELAY_DIFF      35                // charger >
battery by 0.1V - (Diode -0.4V) = 0.5v
#define MINIMUM_VOLTS          760              // turnoff
voltage 10.75 V
```

```

#define POWER_UP_VOLTS      830                // power on
voltage 11.75 V
#define BATTERY_FULL_VOLTS  948                // floating
voltage 13.4 V = 13.8-0.4
#define MAXIMUM_VOLTS      999                // charge cutoff
13.8 V (+0.4 Dode) = 14.2 V

// Timing delay intervals
#define CHARGE_INTERVAL     20*60             // charge time before
recheck
#define BATT_FULL_WAIT      10*60             // wait interval after
charge complete
#define WATCHDOG_TIMEOUT   6*60              // CRM sleeps the box if
imager cpu does not refresh in 5 minutes
#define WATCHDOG_TO_PWR    2*60              // Power timeout is
shorter
#define WATCHDOG_SLEEP     3*60              // Sleep interval after
timeout shutdown
#define CPU_SLEEP           30*60             // CPU Sleep
time ???
#define OFFLINE_MAX         6*60*60          // Don't sleep
for more than 6 hours
#define BATT_CHARGE_CD      10                // wait interval
after charge cycle
#define IN_RANGE_WAIT_ON    100              // slave turn on
delay
#define IN_RANGE_WAIT_OFF   10                // slave turn
off delay

// A/D channel selection bits
#define PANEL_CHAN          0                 // A/D
channel for solar panel connection
#define BATT_CHAN           1                 // A/D
channel for batter connection

// battery_state definitions
#define BATT_DRAINING       0                 // Waiting for
recharge conditions to be met
#define BATT_CHARGING       1                 // Relay is
closed
#define BATT_COOLDOWN       2                 // Battery has
reached full charge

// CPU_state definitions
#define CPU_SLEEPING        0                 // CPU is off
with plans to wake
#define CPU_TIMED_OUT       1                 // CPU waiting
to reboot after comm. timeout
#define CPU_LOW_POWER       2                 // Battery has
run out of power

```

```

#define CPU_ON          3          // CPU
up and running
#define CPU_POWERING_UP 4          // CPU on but
waiting to boot
#define CPU_OFF        5          // CPU
has turned itself off

// Port pinouts
// Port B
#define RELAY_SW        0x01          // pin 0
00000001 MCU pin 14
#define CPU_SW          0x02          // pin 1
00000010 MCU pin 15
#define MCU_POW_LED    0x04          // pin 2 00000100 MCU
pin 16
#define CPU_POW_LED    0x08          // pin 3 00001000 MCU
pin 17
#define IN_RANGE_LED   0x10          // pin 4 00010000 MCU
pin 18
#define DEBUG_LED      0x20          // pin 5
00100000 MCU pin 19

//Port D
#define RXD             0x01          // pin 0
00000001 MCU pin 2
#define TXD             0x02          // pin 1
00000010 MCU pin 3
#define M_S_SWITCH     0x08          // pin 3
00001000 MCU pin 5 Master/Slave Switch
#define MODEM_RESET    0x10          // pin 4
00010000 MCU pin 6
#define MODEM_RANGE     0x20          // pin 5
00100000 MCU pin 11

#define ADC_MUX_MASK    0x07

////////////////////////////////////
////////////////////////////////////
// Delay
////////////////////////////////////
////////////////////////////////////
//void delay_ms(unsigned int);
void delay_s(unsigned int);

inline unsigned char is_in_range(void);
inline unsigned char is_master(void);
inline void close_relay(void);
inline void open_relay(void);

```

```

inline void cpuOn(void);
    //Turns on the CPU imager
inline void cpuOff(void);
    //Turns off the CPU imager
inline void flashLEDs(unsigned char length);

inline void init_T1OVF(void); //Timer
overflow initializer
inline void init_ADC(void);
    //Initializes the A/D converter
inline void TIM16_addTCNT1(unsigned int i);

// Possible configuration parameters for timer0_config()
#define TIMERO_NOCLK 0
#define TIMERO_NOPRE 1
#define TIMERO_PRE_8 2
#define TIMERO_PRE_64 3
#define TIMERO_PRE_256 4
#define TIMERO_PRE_1024 5
#define TIMERO_EXT_FALLING 6
#define TIMERO_EXT_RISING 7
inline void timer0_enable(void);
inline void timer0_disable(void);
inline void timer0_config(unsigned char config);
inline void timer0_set(unsigned char);
inline unsigned char timer0_read(void);

#define TIMER1_NOCLK 0
#define TIMER1_NOPRE 1
#define TIMER1_PRE_8 2
#define TIMER1_PRE_64 3
#define TIMER1_PRE_256 4
#define TIMER1_PRE_1024 5
#define TIMER1_EXT_FALLING 6
#define TIMER1_EXT_RISING 7
inline void timer1_enable(void);
inline void timer1_disable(void);
inline void timer1_config(unsigned char config);
inline void timer1_set(unsigned int);
inline unsigned int timer1_read(void);

#define TIMER2_NOCLK 0
#define TIMER2_NOPRE 1
#define TIMER2_PRE_8 2
#define TIMER2_PRE_32 3
#define TIMER2_PRE_64 4
#define TIMER2_PRE_128 5
#define TIMER2_PRE_256 6
#define TIMER2_PRE_1024 7
inline void timer2_enable(void);
inline void timer2_disable(void);

```

```

inline void timer2_config(unsigned char config);
inline void timer2_set(unsigned int);
inline unsigned int timer2_read(void);

// Serial I/O
extern void uart_recv_flush(void);
extern int uart_send(char c, FILE *fp);
extern int uart_recv(FILE *fp);
extern char kbhit(void);
extern void ioinit(void);

// A/D Conversion
extern inline void adc_set_reference(uint8_t ref);
#define ADC_REF_AREF          0x0
#define ADC_REF_AREF_CAP     0x1
#define ADC_REF_2p56V        0x3

extern inline void adc_set_adlar(uint8_t adlar);
extern inline void adc_set_channel(uint8_t chan);
#ifdef atmega8
#define ADC_CHANNEL_0         0x0
#define ADC_CHANNEL_1         0x1
#define ADC_CHANNEL_2         0x2
#define ADC_CHANNEL_3         0x3
#define ADC_CHANNEL_4         0x4
#define ADC_CHANNEL_5         0x5
#define ADC_CHANNEL_6         0x6
#define ADC_CHANNEL_7         0x7
#define ADC_CHANNEL_1p23V     0xE
#define ADC_CHANNEL_0V        0xF
#else
#define ADC_CHANNEL_0         0x0
#define ADC_CHANNEL_1         0x1
#define ADC_CHANNEL_2         0x2
#define ADC_CHANNEL_3         0x3
#define ADC_CHANNEL_4         0x4
#define ADC_CHANNEL_5         0x5
#define ADC_CHANNEL_6         0x6
#define ADC_CHANNEL_7         0x7
#define ADC_CHANNEL_1p1V      0xE
#define ADC_CHANNEL_0V        0xF

```

```

#endif

extern inline void adc_set_enable(uint8_t cmd);
#define ADC_ENABLE 1
#define ADC_DISABLE 0

extern inline void adc_start_conversion(void);

extern inline void adc_set_auto_trigger(uint8_t cmd);
#define ADC_AUTO_TRIGGER_ENABLE 1
#define ADC_AUTO_TRIGGER_DISABLE 0

extern inline uint8_t adc_interrupt_flag(void);

extern inline void adc_interrupt_enable(uint8_t cmd);
#define ADC_INTERRUPT_ENABLE 1
#define ADC_INTERRUPT_DISABLE 0

extern inline void adc_set_prescalar(uint8_t cmd);
    // #define ADC_PRESCALAR_1 0 // Correct? docs say this is factor 2
#define ADC_PRESCALAR_2 1
#define ADC_PRESCALAR_4 2
#define ADC_PRESCALAR_8 3
#define ADC_PRESCALAR_16 4
#define ADC_PRESCALAR_32 5
#define ADC_PRESCALAR_64 6
#define ADC_PRESCALAR_128 7

extern inline uint16_t adc_read(void);

unsigned int ADC_conversion(unsigned char chan);

#ifdef atmega88
extern inline void adc_set_trigger_source(uint8_t cmd);
#define ADC_TRIGGER_MODE_FREE 0x0
#define ADC_TRIGGER_MODE_ACOMP 0x1
#define ADC_TRIGGER_MODE_EXTINT 0x2
#define ADC_TRIGGER_MODE_TCOA 0x3
#define ADC_TRIGGER_MODE_TCOO 0x4
#define ADC_TRIGGER_MODE_TC1B 0x5
#define ADC_TRIGGER_MODE_TC10 0x6
#define ADC_TRIGGER_MODE_TCCE 0x7
extern inline void adc_set_digital_disable(uint8_t cmd);
#endif

```

CRMWD_lib.c

```
////////////////////////////////////
//////////
//
// CRMWD_lib.c
//
// Owner: Yih-Ru Huang
// Modified: 20070707
//
////////////////////////////////////
//////////

#include "CRMWD_lib.h"

////////////////////////////////////
//////////
// Delay
// convert from loop cycles to msec
//
// void delay_ms(unsigned int t)
//
// _delay_loop_2() can only take a 16-bit argument. This routine
// repeatedly calls _delay_loop_2() in order to get the desired busy
// wait length. Note that there is some overhead in the loop that is
// not counted in the delay (so the delay is actually just a little longer)
//
// This code is also specific to the 1 MHz clock
//
////////////////////////////////////
//////////
#define _delay_ms(x) _delay_loop_2((x)*(F_CPU/4000))

inline void delay_s(unsigned int t)
{
    _delay_loop_2((t*(F_CPU/1UL)) >> 2);
};

////////////////////////////////////
//////////
// is_in_range
// Checks Port D pin 5 to see if the master switch is set
////////////////////////////////////
//////////
inline unsigned char is_in_range(void)
{
    return !(PIND & MODEM_RANGE);
}
```



```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
// Checks Port D pin 3 to see if the master switch is set
// masterOrSlave sets the indicator light for the
// master/slave setting of the input switch
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
inline unsigned char is_master(void)
{
    return PIND & M_S_SWITCH;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
// close_relay and open_relay
// Set the output control for the battery charging relay
// and the LED indicator to either on or off
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
inline void close_relay(void)
{
    PORTB |= RELAY_SW;
    printf("\r\n");
    printf("*****\r\n");
    printf("**   Relay Closed   **\r\n");
    printf("*****\r\n");
}

inline void open_relay(void)
{
    PORTB &= ~RELAY_SW;
    printf("\r\n");
    printf("*****\r\n");
    printf("**   Relay Opened   **\r\n");
    printf("*****\r\n");
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
// Turn on power to the CPU and other light on
// PB1 is the POWER MOSFET that switches 5 V to the
// Computer If it is LOW then the computer is ON
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
inline void cpuOn(void)
{
    //PORTB |= CPU_SW;
    //PB3 is the Computer ON light IF it
    PORTB &= ~CPU_SW;
    // no Q1 BS170
}

```

```

        PORTB |= CPU_POW_LED;                                //is Lit
then the computer is on
    printf("\r\n");
    printf("*****\r\n");
    printf("** CPU Power ON **\r\n");
    printf("*****\r\n");
}

inline void cpuOff(void)
{
    //PORTB &= ~CPU_SW;                                    // Turn
off power to the computer.
    PORTB |= CPU_SW;
    // no Q1 BS170
    PORTB &= ~CPU_POW_LED;                                // Turn
off led
    printf("\r\n");
    printf("*****\r\n");
    printf("** CPU Power OFF **\r\n");
    printf("*****\r\n");
}

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// flashLEDs
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
inline void flashLEDs(unsigned char length)
{
    PORTB |= DEBUG_LED + IN_RANGE_LED + MCU_POW_LED + CPU_POW_LED;
    _delay_ms(length);
    PORTB &= ~(DEBUG_LED + IN_RANGE_LED + MCU_POW_LED + CPU_POW_LED);
    _delay_ms(length);
}

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// This initializes the timer0 overflow for the microcontroller
// After this is set the timer 0 overflow vector is called
// after every overflow
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
inline void init_T1OVF(void)
{
    // Timer/Counter Control Register 1 B
    // 0x04 is CS 100, T1 src = CLKcpu / 1024
    TCCR1B = (1<<CS12);
    // Timer Interrupt Flag Register
    // Clears pending interrupt
    TIFR = (1<<TOV1);
}

```

```

    // Timer Interrupt Mask Register
    // Enable Timer 1 overflow interrupt trigger
    TIMSK = (1<<TOIE1);
}

inline void init_ADC(void) //This just chooses the channel for the microcontroller
{
    //ADMUX bits 7:6 (REFS1 REFS0) settings and description
    //0 0 AREF, Internal Vref turned off
    //0 1 AVCC with external capacitor at AREF pin
    //1 0 Reserved (do not use)
    //1 1 Internal 2.56V Voltage Reference with external capacitor at AREF pin
    //bit 4 Unused
    //ADLAR bit 5 = 0 Don't left adjust result
    //MUX bits 3:0 = 0000 channel selection, set to chan 1
    ADMUX = (1<<REFS0) | (1<<MUX0);
    //Wait for the channel selection to settle
    // _delay_ms(100);
    //Enable ADEN, set prescaler bits (2:0) to divide by 64 (2^0b110 = 2^6)
    ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1);
}

//Performs an atomic write to the TCNT1 register
inline void TIM16_addTCNT1(unsigned int i)
{
    unsigned char sreg;
    sreg = SREG; // Save
global interrupt register
    cli();
    // Disable global interrupts
    TCNT1 += i;
    // Set counter
    SREG = sreg; //
Restore global interrupt flags
    sei();
    // Enable global interrupts
}

//inline void init_TOF(void)
//{{
// // Timer/Counter Control Register 0
// // Set the clock source for timer 0 to (I/O Clk / 1024)
// // This is the slowest internal clock setting
// TCCR0 = (1<<CS02)|(1<<CS00);
// // Timer Interrupt Flag Register
// // clears a pending interrupt
// TIFR = 1<<TOV0;
// // Timer Interrupt Mask Register
// // enables Timer zero overflows to trigger an interrupt
// TIMSK = 1<<TOIE0;
//}}

```

```

////////////////////////////////////
////////////////////////////////////
// Timer Support
////////////////////////////////////
////////////////////////////////////
inline void timer0_enable(void)
{
    TIMSK |= _BV(TOIE0);
};

inline void timer0_disable()
{
    TIMSK &= ~_BV(TOIE0);
};

inline void timer0_config(unsigned char config)
{
    TCCR0 = (TCCR0 & 0xF8) | (config & 0x7);           // Replace the lowest 3 bits
};

inline void timer0_set(unsigned char val)
{
    TCNT0 = val;
};

inline unsigned char timer0_read(void)
{
    return(TCNT0);
};

////////////////////////////////////
////////////////////////////////////
// Timer 1
////////////////////////////////////
////////////////////////////////////
inline void timer1_enable(void)
{
    TIMSK |= _BV(TOIE1);
};

inline void timer1_disable()
{
    TIMSK &= ~_BV(TOIE1);
};

inline void timer1_config(unsigned char config)
{
    TCCR1B = (TCCR1B & 0xF8) | (config & 0x7);       // Replace the lowest 3 bits
};

```

```

inline void timer1_set(unsigned int val)
{
    TCNT1 = val;
};

inline unsigned int timer1_read(void)
{
    return(TCNT1);
};

// Timer 2
inline void timer2_enable(void)
{
    TIMSK |= _BV(TOIE2);
};

inline void timer2_disable()
{
    TIMSK &= ~_BV(TOIE2);
};

inline void timer2_config(unsigned char config)
{
    TCCR2 = (TCCR2 & 0xF8) | (config & 0x7); // Replace the lowest 3 bits
};

inline void timer2_set(unsigned int val)
{
    TCNT2 = val;
};

inline unsigned int timer2_read(void)
{
    return(TCNT2);
};

// Serial interface support
// I/O interface (enables things like getchar(), putchar(), printf(),
// scanf() to work
// void uart_rcv_flush (void)
//
// Clear out the contents of the receive buffer

```

```

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
void uart_recv_flush (void) // p.144
{
    unsigned char dummy;
    while(bit_is_set(UCSRA, RXC)) dummy = UDR;
}

// int uart_send(char c)
// Send a byte to the serial port
int uart_send(char c, FILE *fp) // p.137
{
    loop_until_bit_is_set(UCSRA, UDRE);
    UDR = c;
    return 0;
}

// int uart_recv(void)
// Receive a byte from the serial port
int uart_recv(FILE *fp) // p.140
{
    loop_until_bit_is_set(UCSRA, RXC);
    return UDR;
}

// char kbhit(void)
// Return: 1 = character waiting in buffer
//        0 = no character waiting
char kbhit(void)
{
    return (UCSRA & (1<<RXC)) >> RXC; // RCX, p.151
}

// void ioinit(void)
// Initialize PDO/PDI as a serial device, p.136
void ioinit(void)
{
    UCSRB = (1<<RXEN)|(1<<TXEN);
    UCSRB |= (1<<RXCIE); // Open
    USART_UDRE_vect Interrupt
    unsigned int baud = (7800000 / (16 * 1200UL)) - 1; //3.3v 7800000, 5.0v 800000
    UBRRH = (unsigned char) (baud>>8);
    UBRRL = (unsigned char) baud; // Use defaults for
    UCSRC, 8N1, p.153
    fdevopen(uart_send, uart_recv);
}

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// A/D Conversion Measured from voltages and A/D readouts

```

```

////////////////////////////////////
////////////////////////////////////

// Analog reference
inline void adc_set_reference(uint8_t ref)
{
    ADMUX = (ADMUX & 0x3F) | ((ref & 0x3) << 6);
};

// ADC Left Adjust Result
inline void adc_set_adlar(uint8_t adlar)
{
    ADMUX = (ADMUX & 0xDF) | ((adlar & 0x1) <<5);
}

// Channel selection
inline void adc_set_channel(uint8_t chan)
{
    ADMUX = (ADMUX & 0xF0) | ((chan & 0xF));
}

inline void adc_set_enable(uint8_t cmd)
{
    ADCSRA = (ADCSRA & 0x7F) | ((cmd & 0x1) << 7);
}

inline void adc_start_conversion(void)
{
    ADCSRA = ADCSRA | 0x40;
}

inline void adc_set_auto_trigger(uint8_t cmd)           // cmd is one of:
{
    // ADC_AUTO_TRIGGER_ENABLE           // ADC_AUTO_TRIGGER_DISABLE
    ADCSRA = (ADCSRA & 0xDF) | ((cmd & 0x1) << 5);
}

inline uint8_t adc_interrupt_flag(void)
{
    return((ADCSRA&0x10) >> 4);
};

inline void adc_interrupt_enable(uint8_t cmd)         // cmd is one of:
{
    // ADC_INTERRUPT_ENABLE           // ADC_INTERRUPT_DISABLE
    ADCSRA = (ADCSRA & 0xF7) | ((cmd & 0x1) << 3);
};

inline void adc_set_prescalar(uint8_t cmd)           // cmd is one of:
{
    // ADC_PRESCALAR_*

```

```

    ADCSRA = (ADCSRA & 0xF8) | (cmd & 0x7);
};

// Note: we assume adlar =0
// Blocking read of ADC result
inline uint16_t adc_read(void)
{
    uint16_t out;
    while(!adc_interrupt_flag()); // Wait for ADC data to
    be ready                       // Read out the
    out = (uint16_t) ADCL;          // Read out the
    value: order of register access is important!
    out |= (((uint16_t) (ADCH&0x3)) << 8);
    return(out);
};

unsigned int ADC_conversion(unsigned char chan)
{
    unsigned int result = 0;
    // Set the A/D channel for
    //ADMUX &= ~ADC_MUX_MASK;
    //ADMUX |= (ADC_MUX_MASK & chan);
    ADMUX = (ADMUX & 0b11110000) | chan;
    // _delay_ms(60); //!!!!Where in
    documentation is this necessary? // Start
    conversion...
    while(!(ADCSRA & (1<<ADIF))); // ...and wait
    //while(!(ADCSRA & 0x10));
    ADCSRA |= (1<<ADIF);
    result = ADCL|(ADCH<<8);
    //printf("%d\r\n", result);
    return result;
}

////////////////////////////////////
////////////////////////////////////
#ifdef atmega88

// inline void adc_set_trigger_source(uint8_t cmd)
// cmd is one of:
// ADC_TRIGGER_MODE_*
inline void adc_set_trigger_source(uint8_t cmd)
{
    ADCSRB = (ADCSRB & 0xF8) | (cmd & 0x7);
};

// ACME?
inline void adc_set_digital_disable(uint8_t cmd)
{

```



```
DDIRO = (DDIRO & 0xC0) | (cmd & 0x3F);  
};
```

```
#endif
```

```
////////////////////////////////////  
////////////////////////////////////  
////////////////////////////////////
```

Appendix F: KOUN Runway 17 CRM-Box System Sighting

Station	Latitude	Longitude	EllHgt(m)
ZX01A	35 14 37.52730	-97 28 19.25797	331.964,Good
ZX01B	35 14 37.50285	-97 28 26.54952	331.899,Good
ZX02A	35 14 39.40140	-97 28 19.25965	332.445,Good
ZX02B	35 14 39.56745	-97 28 26.56197	332.214,Good
ZXMaster	35 14 32.35064	-97 28 00.75543	342.608,Published-3D
17T	35 15 23.02000	-97 28 23.22000	333.654 246_B
17E	35 14 31.65000	-97 28 22.84000	332.319 246_B

Digital Aeronautical Database System (DADS)
(Version 2.8/06)

US DOT/Federal Aviation Administration
Aviation System Standards
Information Technology Staff
NAS Management Systems Branch
1305 East-West Highway
Silver Spring, MD 20910
(301) 713-1186

*** INVERSE (GRS 80) *** 3/7/2008 (2008067), 3:44:19 PM

*** Distance conversion factor:

*** 1 Nautical mile = 1852.00 Meters

*** 1 meter = 3.28083 US standard foot

17T

ZX02B ZX02A

KOUN2 KOUN1

ZX01B ZX01A

17E

Station	Latitude	Longitude	Variation	Tag
17T	35 14 31.65000N	097 28 22.84000W	000.00000W	--
17E	35 15 23.02000N	097 28 23.22000W	000.00000W	--

1A	35 14 37.52731N	097 28 19.25796W	000.00000W	--
1B	35 14 37.50287N	097 28 26.54952W	000.00000W	--
2A	35 14 39.40139N	097 28 19.25966W	000.00000W	--
2B	35 14 39.56742N	097 28 26.56195W	000.00000W	--
C	35 14 37.51517N	097 28 22.88338W	--	ANS

From--To	Azimuth	Magnetic	Distance
17E17T	359.65235	359.65235	1583.15474 M
1A1B	269.76649	269.76649	184.34965 M
17EC	359.65235	359.65235	180.75651 M
17TC	179.65229	179.65229	1402.39823 M
1AC	269.76642	269.76642	091.66007 M
1BC	089.76539	089.76539	092.68958 M

DbtRV 333.654-331.964=1.69

DbtLV 333.654-331.899=1.755

Appendix G: CRM Server Linux Perl Scripting Language

sortp.pl

```
#!/usr/bin/perl -w

use strict;
use Getopt::Std;    #must use "my"

my %options;
getopts("s:p:d:p",\%options);  # "-s KOUN" or "-d yyyyymmdd"

my $indir = "/home/ben/incoming/";
my $outdir = "/home/ben/data/";

my $processDay;
unless (defined $options{d})
{
    my ($sec,$min,$hour,$mday,$mon,$year,$yday,$yday) = gmtime(time-2*86400);
    $year += 1900;
    $mon++;
    $processDay = sprintf "%04d%02d%02d", $year, $mon, $mday;
}

chdir $indir or die "cannot enter incoming dir\n";

my %p;    # @ { $p{site}{cam} } is files

# build hash of all packages, process each site/camera group
foreach (<*.bz2>)
{
    # parse out CRM compressed package name
    next unless (/(\w+)-(\w+)-(\w+)\.bz2/);
    my $site = $1;
    my $cam = $2;
    my $time = $3;

    # if site argument specified, only process that site
    if (defined $options{s})
    {
        if ($site =~ /$options{s}/)
        {
            push @ { $p{$site}{$cam} }, $_;
        }
    }
    else
    {
        push @ { $p{$site}{$cam} }, $_;
    }
}
}
```

```

my %proc;

# seperate dates into per day bins
foreach my $site (keys %p)
{
    foreach my $cam (keys %{ $p{$site} })
    {
        foreach my $f (@{ $p{$site}{$cam} })
        {
            $f =~ /(\w+)-(\w+)-(\w+)\.bz2/;

            my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday) = gmtime($3);
            $year += 1900;
            $mon++;
            my $datestr = sprintf "%04d%02d%02d", $year, $mon, $mday;
            push @{ $proc{$site}{$cam}{$datestr} }, $f;
        }
    }
}

# dont process current date or not the select day, let it accumulate
foreach my $site (keys %proc)
{
    foreach my $cam (keys %{ $proc{$site} })
    {
        foreach my $date (sort keys %{ $proc{$site}{$cam} })
        {
            if ( (defined $options{d}) && ($date != $options{d}) )
            {
                print "Dropping date- $date from $site $cam\n";
                delete $proc{$site}{$cam}{$date};
            }
            elsif (!(defined $options{d})) #process only
            pick up day (today-2)
            {
                if ( $date != $processDay )
                {
                    print "Dropping date: $date from $site $cam\n";
                    delete $proc{$site}{$cam}{$date};
                }
            }
        }
    }
}

my $total_files = 0;

foreach my $site (sort keys %proc)
{
    foreach my $cam (sort keys %{ $proc{$site} })

```

```

{
    foreach my $date (sort keys %{ $proc{$site}{$cam} })
    {
        my $airport = substr $site, 0, 4;
        my $dir = "$outdir/$airport/$date/tmp";

        print "Output dir: $dir\n";
        system("rm -rf $dir");    # delete temporary directory
        system("mkdir -p $dir");

        $total_files += ${ $proc{$site}{$cam}{$date} };
        print("Site: $site cam: $cam date: $date " .
            "${ $proc{$site}{$cam}{$date} }\n");
        next if defined $options{p};    # only print file stats

        foreach my $f (sort @{ $proc{$site}{$cam}{$date} })
        {
            system("cp", $f, "$dir");
            system("bunzip2", "$dir/$f");
        }
        my $cc = substr($cam,1,1);

        # repackage data into one large file
        system("/home/ben/src/repack/repack $cc $dir > $dir.bigfile");

        # remove all the small files
        foreach (@{ $proc{$site}{$cam}{$date} })
        {
            unlink $_;
        }

        my $newname = "$outdir/$airport/$date/$site-$cam-$date";
        while (-f $newname)
        {
            $newname .= "B";
        }
        rename "$dir.bigfile", $newname;

        system("md5sum $newname >> $outdir/$airport/$date/md5sum.txt");
        system("rm -rf $dir");    # delete temporary directory
    }
}
print "site: $site\n";
}

print "Total number of CRM files: $total_files\n";

```

weather_p.pl

```
#!/usr/bin/perl -w

use strict;
use Time::Local;

$ENV{PATH} .= ":/usr/local/mysql/bin/";

my $root = "/home/ben/src/logger";

chdir $root or die "cannot chdir to $root";

my $SECHR = 3600;          # seconds in hour

if (open(F, "CURRENT"))   # save old weather
{
    my $a = <F>;
    if ($a =~ /\d+/)
    {
        rename("CURRENT", "CURRENT.archive/CURRENT.$1");
    }
}

open STDOUT, ">CURRENT" or die "cannot change STDOUT to output file";

my @st;
my %pos;
open F, "STATIONS" or die "cannot open station ID file";
while (<F>)
{
    chomp;
    @_ = split /\s+/, $_;
    push @st, $_[0];
    $pos{$_[0]}{'lat'} = $_[3];
    $pos{$_[0]}{'lon'} = $_[4];
}
my $st = join '%20', @st;    # all KXXX stations together

my %w = ();    # hash of weather: $w{"station"}{"code"} = "altitude"

unlink("METARS");    # delete old weather data file

my $query = 'http://adds.aviationweather.noaa.gov/metars/index.php?station_ids=' .
    $st .
    '&std_trans=standard&chk_metars=on&chk_tafs=off';

# download METARS with wget, check that file is Okay
my $try2 = 0;
my $done = 0;
my $skip_proc = 0;
```

```

while (!$done)
{
    system("wget -qO METARS $query");
    system("echo $query");

    if (system("grep \"no data available\" METARS") == 0 && $try2)
    {
        print "both stations down... going by sun position\n";
        foreach my $s (@st)
        {
            ${$s}{'CLR'} = 1e9;
            ${$s}{'TIME'} = time;
        }
        $done = 1;
        $skip_proc = 1;
    }
    elsif (system("grep \"no data available\" METARS") == 0)
    {
        print "weather.aero is down, trying adds\n";
        unlink "METARS";

        $query = 'http://adds.aviationweather.noaa.gov/metars/index.php'.
            '\?station_ids=' . $st .
            '\&std_trans=standard&chk_metars=on&chk_tafs=off';

        $try2 = 1;    # indicate second site attempt
    }
    elsif ((stat("METARS"))[7] < 50)
    {
        # if file is small, METARS didn't work correctly
        print "unable to get METARS (zero byte file)\n";
        sleep(20);
        unlink "METARS";
    }
    else
    {
        print "Downloaded ", (stat("METARS"))[7], " bytes weather.\n";
        $done = 1;
    }
}

$/ = '<BR>';    # break input lines on HTML tag instead of \n

open(F, "METARS") or die "unable to open METARS raw data";

##### Build hash of weather information #####
while (<F>)
{
    print "$_\n";
    s/\n//g;
}

```



```

next unless /\>(K.*?)\</;

# get GMT (UTC) day, month, year since ADDS doesn't provide
my ($sec,$min,$hour,$mday,$mon,$year,$yday,$isdst) = gmtime(time);

my @a = split(/ /, $1);

my $station = $a[0];

##### time processing section #####
$a[1] =~ /(..)(..)(..)Z/;
my $a_day = $1; my $a_hour = $2; my $a_min = $3;

# check if ADDS data is in the future
if ($a_day > $mday)
{
    print "ADDS data (day) is in future: $a_day > $mday\n";
}
if ($a_day == $mday && $a_hour > $hour)
{
    print "ADDS data (hour) is in future: $a_hour > $hour\n";
}
if ($a_day == $mday && $a_hour == $hour && $a_min > $min)
{
    print "ADDS data (minute) is in future: $a_min > $min\n";
}

# check if ADDS day rolled passed end of month (before us)
if ($a_day < 6 && $mday > 20)
{
    # fixup $mon and $year for ADDS measurement ahead
    $mon++;
    if ($mon == 12) # end of year rollover
    {
        $mon = 0;
        $year++;
    }
    print "ADDS data rolled passed the month\n";
    print " -- rolled month: $mon year: $year\n";
}

# timegm converts back to UTC seconds
# has parameters: ($sec,$min,$hour,$mday,$mon,$year)
# where $mday is 0..30, $mon is 0..11, and $year is 105 for 2005
my $utcsec = timegm(0,$a_min,$a_hour,$a_day,$mon,$year);

$w{$station}{'TIME'} = $utcsec;

foreach (@a) # find cloud altitude tokens
{
    last if /RMK/; # ignore RMK onward
}

```

```

if ((CLR|FEW|SCT|BKN|OVC|VV)(\d\d\d)?/)
{
    if ($1 eq 'CLR' and not defined $2)
    {
        $w{$station}{$1} = 1e9;
    }
    else
    {
        if (not defined $w{$station}{$1} || $w{$station}{$1} >
$2)
        {
            $w{$station}{$1} = $2;
        }
    }
}
}

```

```

# since LAX has dual stations, we copy the data
#
$w{'KLAY'} = $w{'KLAX'} if exists $w{'KLAX'};
$pos{'KLAY'} = $pos{'KLAX'} if exists $pos{'KLAX'};
#$w{'KLAZ'} = $w{'KLAX'} if exists $w{'KLAX'};
#pos{'KLAZ'} = $pos{'KLAX'} if exists $pos{'KLAX'};
    $w{'KOUN'} = $w{'KOKC'} if exists $w{'KOKC'};
    $pos{'KOUN'} = $pos{'KOKC'} if exists $pos{'KOKC'};

```

```

#### Data output #####

```

```

print 'Weather grab, UTC time: ', time, "\n";

```

```

foreach my $st (sort keys %w)
{

```

```

#     insert data into this table:
#     CREATE TABLE stations(
#     id          varchar(5),
#     sys_time    integer,
#     diff_time   integer,
#     turn_off_time integer,
#     turn_on_time integer,
#     cloud_string varchar(80)
#     );

```

```

my $sun_is_up = is_sun_up($pos{$st}{'lat'}, $pos{$st}{'lon'});
$sun_is_up = 1000000; # dont use sundown as data

```

```

# clear old measurements (only if ADDS hasnt been updated)
my $de = "echo \"delete from stations where id = '$st' \" .
        \"and sys_time = \${w{\$st}{TIME}}\" \" .
        \"| mysql --user=root --password=lal23 crm\";
system($de);

my $in = 'echo "insert into stations (id, sys_time, diff_time,' .
        'turn_off_time, turn_on_time, cloud_string) values (';

$in .= "'$st', ";
$in .= "\${w{\$st}{TIME}}, ";

my $dt = time - ${w{\$st}{TIME}};
next if (abs($dt) > 20000); # weather time off
$in .= "$dt, ";

# grab all altitudes and sort (to find lowest)
my @alts = sort {$a <=> $b} values %{$w{\$st}};

# turn off until sunset (max time) if clear
if ($sun_is_up > 0 && $alts[0] > 100)
{
    $in .= "0, "; # next shutdown (now)
    $sun_is_up = 2.0*$SECHR if ($sun_is_up > 4.0*$SECHR);
    $in .= $sun_is_up-600 . ", ";
}
elseif ($sun_is_up == -1) # sun is down, stay online
{
    ${w{\$st}{SUNDOWN}} = 1;
    $in .= 2.0*$SECHR . ", "; # next shutdown (future)
    $in .= 0.5*$SECHR . ", "; # 0.5 hr sleep (ignored)
}
elseif ($alts[0] <= 20) # stay up if ceiling is 2000 ft
{
    $in .= 2.0*$SECHR . ", "; # next shutdown (future)
    $sun_is_up = 0.5*$SECHR if ($sun_is_up > 0.5*$SECHR);
    $in .= $sun_is_up . ", ";
}
elseif ($alts[0] <= 50) # 5,000 ft, come back in .5 hour
{
    $in .= "0, "; # next shutdown (now)
    $sun_is_up = 0.5*$SECHR if ($sun_is_up > 0.5*$SECHR);
    $in .= $sun_is_up . ", ";
}
elseif ($alts[0] <= 100) # 10,000 ft, come back in 1.0 hour
{
    $in .= "0, "; # next shutdown (now)
    $sun_is_up = 1.0*$SECHR if ($sun_is_up > 1.0*$SECHR);
    $in .= $sun_is_up . ", ";
}
else # clear, sleep 2.0 hour

```

```

    {
        $in .= "0, ";
        $sun_is_up = 1.5*$SECHR if ($sun_is_up > 1.5*$SECHR);
        $in .= $sun_is_up . ", ";
    }

    $in .= "";
    # store each cloud altitude, for completeness
    foreach (sort keys %{ $w{$st} })
    {
        next if /TIME/;
        # if CLR defined and no other terms, weather is clear
        if (/CLR/) # && scalar keys %{ $w{$st} } == 2
        {
            $in .= 'CLEAR ';
        }
        elsif ($_ eq 'SUNDOWN')
        {
            $in .= 'SUNDOWN ';
        }
        else # just print the term
        {
            $in .= "$_ " . $w{$st}{$_} * 100 . " ";
        }
    }
    $in .= "";

    $in .= ');" | mysql --user=root --password=la123 crm';
    print("insert string: $in\n");
    system($in);
}

close(F);

sub is_sun_up
{
    my $lat = shift;
    my $lon = shift;

    my $next_day = 0;

NEXTDAY:
    my @t = gmtime(time + $next_day);

    $t[5] += 1900; # year (+1900)
    $t[4] += 1; # mon (+1)
    $t[3] -= 1; # day
    my $sun_args = "$t[5] $t[4] $t[3] $lat $lon";

    # get sunrise/sunset information from sunrise command
    print "./sunrise $sun_args |";
}

```

```

open F, "./sunrise $sun_args 1";
my $sun = <F>;
close F;

$sun =~ /Sunrise\s+(\d+)-(\d+)-(\d+)\s+(\d+):(\d+):(\d+)\s+
        Sunset\s+(\d+)-(\d+)-(\d+)\s+(\d+):(\d+):(\d+)/x;

# fixup returned dates for timegm
my $year1 = $3 - 1900;
my $mon1 = $1 - 1;
my $year2 = $9 - 1900;
my $mon2 = $7 - 1;

# mon mday year hour min sec ==>
# timegm($sec,$min,$hour,$mday,$mon,$year);
# 03 12 2005 14 22 56, 03 13 2005 02 12 51
#my $sunrise_time = timegm($6,$5,$4,$2,$1,$3);
#my $sunset_time = timegm($12,$11,$10,$8,$7,$9);
my $sunrise_time = timegm($6,$5,$4,$2,$mon1,$year1);
my $sunset_time = timegm($12,$11,$10,$8,$mon2,$year2);

print "sunrise: $sunrise_time sunset: $sunset_time\n";
print "now: " . time . "\n";
# print "$1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12\n";

# if time is greater than sunset time, that day ended
# try the next day:
if ($sunset_time < time)
{
    print "sunset info is old, redoing loop on next day\n";
    $next_day += 24*60*60;
    goto NEXTDAY;
}

if ($sunrise_time < time && $sunset_time > time)
{
    print "Sun is up! sunset in: ", $sunset_time - time, "\n";
    if ($sunset_time - time < 1800) {
        print "less than 30 min. until sunset, overriding\n";
        return -1;
    } else {
        return ($sunset_time - time);
    }
} else {
    print "Sun is DOWN\n";
}

return -1;
}

```

