Depth Analysis Using Stereoscopic Cameras

Leo Nickerson, lnickers@ualberta.ca
Timo Hohn, thohn@ualberta.ca

Hardware accelerated post processing of stereo images for analysis of image depth.

Group Number: 5
Lab days available: Thursdays

Abstract

It was proposed that the Altera DE2 board is used in conjunction with a pair of identical cameras to analyze the spatial dimensions of a room. It was determined that the best way to process information retrieved from the cameras would be to move the image analysis off of the board to a client program. As such the board was set up to be running as a web server which controlled the camera set up, and would send images to a client program. In order to control the cameras and run a web server on the board, access to the UART and Ethernet interfaces was needed. The FPGA on the board was set up as a processor from the NIOS II chip family and running microC/OS-II, so as to be able to have multiple tasks running. Some of the biggest challenges faced in this project was getting the camera and board to communicate properly through the provided vendor interface and driver software, latency due to time taken to download images, and properly setting up the client software to analyze the images. After testing and verification the project was able to successfully analyze the depth of a environment using images acquired from a stereoscopic camera setup.

Table of Contents

Functional Requirements

This project was intended to accurately estimate the distance of objects, in a natural environment, from a stereoscopic camera setup in a time frame as close to real time as possible. This is done by obtaining jpeg images from two identical cameras set a known distance apart at a users request. The images from the cameras are compared, image depth is calculated, and the resulting information presented as a composite image. The composite image shows the depth information through the use of a color gradient, much like how infrared cameras show heat. The color gradient works by assigning a color value to objects based on the distance from the camera that the objects are: pixels representing closer objects would be red, those of farther objects would be blue, and pixels that cannot be matched are made to be black.

Functionality was achieved for the project and image depth was able to be found for natural environments. However, real-time analysis was not able to be achieved. This was due to a limit in how much data could be downloaded to the board from the camera at a time, as well as the actual download rate. The amount of time it was found to take to request a download and to analyze an image was found to vary from ~45 seconds to ~2 minutes. In addition the quality of the composite image was found to depend on the environment being analyzed, images with lots of distinct features over a variety of depths provided better results. While the project was able to calculate and display which objects were closer and farther away from the camera setup, we were not able to give absolute data on the depth of a scene only relative visual information.

Design and Description of Operation

This project was built on the Altera DE2 board configured with the quartus II development environment. Hardware modules were generated with the SOPC builder available to quartus II, and software was developed using the nios II software build tools for eclipse. The FPGA on the board was set up as the fast NIOS II cpu and configured to utilize SDRAM, SRAM, the green LEDs, 8 of the switches, the LCD display, the Ethernet port, and two UART modules interfaced through the GPIO pins. The SRAM was needed to store the image data read from the camera while SDRAM was used for storing the program information. The green LED's were needed in order to display visual information about the status of the cameras, and the switches to control the number of bytes read from the cameras at a time. The LCD display was used to show the static IP address of the board and the Ethernet port for communicating with the client program. The UARTs were used to interface with the cameras which communicated via 2 pin RS232 protocol at a baud rate of 384000?. The hardware diagram found in the Hardware Documentation section of the Appendix shows the hardware components used and the data lines connecting them. In order to ensure that the SRAM could be accessed a vhdl interface was written conforming to Altera standards. For the Ethernet port a application note from a previous year was used to ensure that the port was properly mapped out in hardware.

The project utilizes the RTOS uC/OS-II in order to achieve multitasking. Multitasking is needed in order to separate control tasks for the web server, and cameras. These tasks need to be separated because if they were integrated then the web server side of the project would not be able to respond to client requests while taking pictures. In order to separate the camera and web server tasks semaphores were used to signal when a request to take a picture was made by a client program as well as when the camera control task was done taking pictures. The memory location at where the image data was stored was also protected by semaphores. Once started the camera control task waits for a request to take a picture, when one is made it takes the picture and stores it in memory, and then once done signals through a semaphore that the pictures have been taken. The server task waits for a client to make a connection, then when a client connects processes any requests the client has. The possible requests the client can make are limited to requesting a picture be take, requesting to download a picture that has been taken, or downloading a file from a read only file system present on the chip.

The client program is used to analyze the images taken by the cameras. It does this through a process known as disparity mapping. This process works by first removing the distortion effects from both images which are produced by the wide field of view from the camera lenses. Once the distortion has been rectified such that pixels of the same object in both images line up on the same horizontal row of pixels (or epipolar line), they can then be iteratively matched from both images. The differences of matched pixels from their respective image centers is known as a that pixel pair's disparity value. Disparity is described as below:

$$D = \frac{b\,f}{r\,x} \qquad \text{Equation (1)}$$

Where D is the disparity value

b is the distance between the centers of the cameras (baseline distance)

f is the focal length of the two sensors (ideally it is identical for both cameras)

r is the range the object is from the camera baseline

x is the pixel size of the image taken

As can be seen in the above equation, the disparity of an object is inversely proportional to the distance that object is from the camera baseline. Larger disparity values would mean closer objects that can be colored with more red hue, while a smaller disparity will imply a farther object that can be colored with a bluer hue. A java library known as BoofCV was used that provides several of the functions for pixel matching, image rectification (distortion removal), and disparity mapping. This library also allowed for image display using the Java Swing library to allow the client to see the resulting disparity mapping.

Bill of Materials

Quantity	Item	Specifications	Supplier	Datasheet	Unit Cost	Total
2	397-ADA TTL Serial JPEG Camera with NTSC Video*	• Frame Speed and image resolution: 680*480 at 30 fps • Default Baud rate for serial communication: 38400 bits per second • Current Draw: 75mA • Uses TTL serial interface for communication at +3.3V • Module operating voltage at +5V • Provides analog CVBS (NTSC encoded) video	ABRA Electronics http://www.abra-electronics.com/products/397-TTL-Serial-JPEG-Camera-with-NTSC-Video.html	On board VC0706 Digital Video Processor http://www.adafruit.com/datasheets/VC0706%20Digital%20Video%20Processor%20Datasheet.pdf PTC08 Camera Module http://www.jameco.com/Jameco/Products/ProdDS/2144340.pdf	$42.00 USD	$84.00 USD
1	Terasic DE2-115 development board	Provided Development Board for ECE492	Terasic http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=502	N/A	$595 USD	$595 USD
1	22 AWD hookup wire		Sparkfun Electronics https://www.sparkfun.com/products/8022	N/A	$2.50 USD	$2.50 USD

8	Camera Screws		N/A	N/A	$0.02 USD	$0.16 USD
1	Camera mount	Custom PVC Camera Mount machined for project	N/A	N/A	Provided by local Machine Shop	Provided by local Machine Shop
1	Lenovo X220T Laptop	Client Machine running java 1.7 run time environment	N/A	N/A	Already Owned	Already Owned

Description and Evaluation of Reusable Design Units

For the initial setup using the economy version of the included NIOS II processor core provided by altera, the post processing will be largely limited to software based calculations that cannot be pipelined. If a custom core is designed, most of the calculation can be offloaded for hardware acceleration, but overall design complexity increases. In order to improve the cycle costs of the software based calculations performed by the RTOS, the Standard or Fast versions of the NIOS II processor could be used to introduce pipelining as well as a built in hardware multiplier.

Open Source Software:

OpenCV: A library of functions for real time computer vision. The main functionality for using this library will be for detection within images as well as performing post processing on those images. It includes C, C++ or Python interfaces and its entire source size is 140.2 MB with a compiled library not including example binaries of 48.3 MB. The majority of this library consists of unneeded functions for this project and will need to be parsed to the minimum functions required. Once accomplished, a more accurate source and compile size can be obtained. The performance of the generic image processing functions will be iterative in nature and therefore fairly slow. The intention is to offload most of the necessary computations to custom hardware on the FPGA. However, during the course of this design, the use of the openCV library will provide a useful backup if this custom hardware is not realised in time. In the case that the OpenCV library is used extensively, reducing the number of data being processed and the frame rate requirements of the project will allow for the OpenCV library to perform adequately.

Cimg.h and libjpeg.h: The Cimg library is a lightweight image processing library in C++. This library does not natively support jpeg file handling, but combined with the library from the Independent JPEG Group, Cimg will be able to handle all of the image processing requirements of this project. the combined source code size of these two libraries comes to 4.1 MB excluding example files and documentation. Once compiled using O2 optimizations of the g++ compiler and excluding the display functions included in Cimg.h, both libraries require 105.4 KB of space. The resulting compile size is much more appropriate to the limited memory space used in this project.

BoofCV: A java implementation of the OpenCV library that specialized further into specific image processing applications. One such specialization is geometric vision that incorporates the intrinsic and extrinsic qualities of the cameras that produced the images to be processed for image recification purposes along with window-based matching techniques to produce a dense disparity mapping of that image stereo pair. Due to the programming language, BoofCV cannot be run directly on the DE2 board as it requires a java runtime environment. In order to incorporate the BoofCV library into this project, images taken will need to be offloaded to a client computer for computation purposes. The library with required dependent libraries included comes to a jar package size of 1.53MB.

Figure 1: Protoboard Connection Diagram

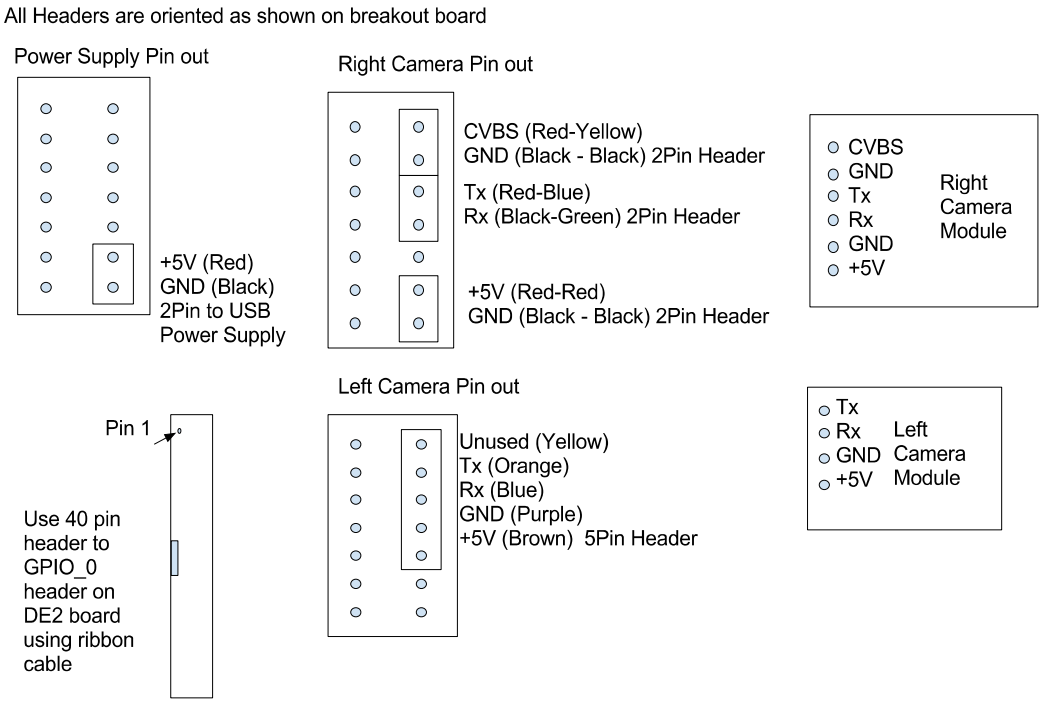
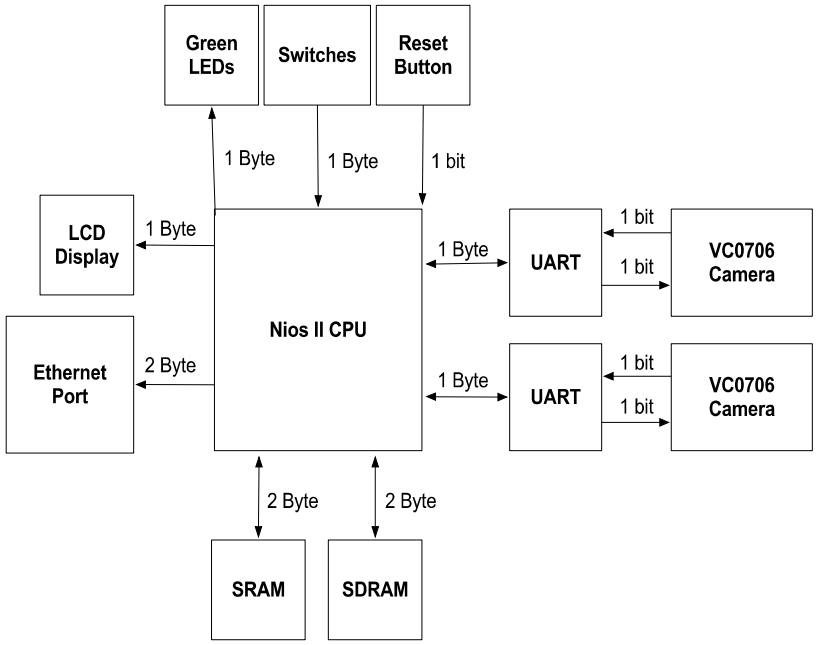


Figure 2: Stereo Camera Hardware Diagram with onboard and offboard peripherals



Nios II System and peripheral IO signals

Syntax: Signal Name : Description [INPUT/OUTPUT] (bus width in bits)

-- 1) global signals:
  clk_0 : 50 Mhz clock [IN] (1)
  reset_n : global reset signal [IN] (1)

-- the_altpll_inst
  altpll_inst_c0_out : 100 Mhz preemptive clock for SDRAM [OUT] (1)
  altpll_inst_c1_out : 100 Mhz clock [OUT] (1)
  altpll_inst_c2_out : 25 Mhz clock for dm9000a core [OUT] (1)
  locked_from_the_altpll_inst : locked from the altpll [OUT] (1)
  phasedone_from_the_altpll_inst : phasedone from the altpll [OUT] (1)

-- the_dm9000a_inst
  ENET_CMD_from_the_dm9000a_inst : command signal [OUT] (1)
  ENET_CS_N_from_the_dm9000a_inst : chip select signal [OUT] (1)
  ENET_DATA_to_and_from_the_dm9000a_inst : data bus [INOUT] (16)
  ENET_INT_to_the_dm9000a_inst : initialize signal [IN] (1)
  ENET_RD_N_from_the_dm9000a_inst : read signal [OUT] (1)
  ENET_RST_N_from_the_dm9000a_inst : reset signal [OUT] (1)
  ENET_WR_N_from_the_dm9000a_inst : write signal [OUT] (1)

-- the_lcd_display
  LCD_E_from_the_lcd_display : enable signal [OUT] (1)
  LCD_RS_from_the_lcd_display : reset signal [OUT] (1)
  LCD_RW_from_the_lcd_display : read write select signal [OUT] (1)
  LCD_data_to_and_from_the_lcd_display : data bus [INOUT] (8)

-- the_led_pio
  signal out_port_from_the_led_pio : on/off signal for green leds [OUT] (8)

-- the_sdram
  zs_addr_from_the_sdram : address bus [OUT] (12)
  zs_ba_from_the_sdram : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
  zs_cas_n_from_the_sdram : OUT STD_LOGIC;
  zs_cke_from_the_sdram : OUT STD_LOGIC;
  zs_cs_n_from_the_sdram : chip select [OUT] (1)
  zs_dq_to_and_from_the_sdram : data bus [INOUT] (16)
  zs_dqm_from_the_sdram : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
  zs_ras_n_from_the_sdram : OUT STD_LOGIC;
  zs_we_n_from_the_sdram : write enable signal [OUT] (1)

-- the_seven_seg_pio
  out_port_from_the_seven_seg_pio : output signal to seven segment display [OUT] (16)

-- the_switch
  in_port_to_the_switch : input signal to first 8 switches [IN] (8)

-- the_tri_state_bridge_0_avalon_slave
  address_to_the_ext_flash : address bus to flash [OUT] (22)
  data_to_and_from_the_ext_flash : data bus to/from flash [INOUT] (8)
  read_n_to_the_ext_flash : read signal [OUT] (1)
  select_n_to_the_ext_flash : select signal [OUT] (1)
  write_n_to_the_ext_flash : write signal [OUT] (1)

-- UART Signals
  txd_from_the_uart_0 : transmit signal to right camera on GPIO_0 [OUT] (1)
  txd_from_the_uart_1 : transmit to left camera on GPIO_0 [OUT] (1)
  rxd_to_the_uart_0 : receive signal from right camera on GPIO_0 [IN] (1)
  rxd_to_the_uart_1 : receive from left camera on GPIO_0 [IN] (1)

-- the_sram_IF_0
  coe_sram_address_from_the_sram_IF_0 : address bus for SRAM chip [OUT] (18)
  coe_sram_chipenable_n_from_the_sram_IF_0 : chip enable [OUT] (1)
  coe_sram_lowerbyte_n_from_the_sram_IF_0 : lower byte signal [OUT] (1)
  coe_sram_outputenable_n_from_the_sram_IF_0 : output enable signal [OUT] (1)
  coe_sram_upperbyte_n_from_the_sram_IF_0 : upper byte signal [OUT] (1)
  coe_sram_writeenable_n_from_the_sram_IF_0 : write enable signal [OUT] (1)
  sram_IF_0_tsb_data : data bus through tristate bridge [INOUT] (16)

The PTC08 Camera Module

includes 6 external ports for both communication and transferring data to and from its host board. These include: 1. CVBS, 2. GND, 3. Tx, 4. Rx, 5. GND, 6. +5V VDD.

The supply voltage is set at +5V DC for onboard operations including communications between the camera as well as the VC0706 digital video processor. The UART interface signals are run at +3.3V. The PTC08 Camera Module draws 75mA peak current. The default baud rate for the TTL signals is 38400 pulses per second and has a series of commands that are available to send across this interface, which is included on the PTC08 datasheet.

Camera Modules will be powered via a +5V power supply through a USB wall power supply (where only power and GND lines are used). USB power supply provides +5V DC with a max current draw of 400mA

DE2-115 development board will be powered with provided power supply (+9V DC). Idle and peak current/voltage were measured during operation of the DE2-115, and were found to be 616mA/9.02V and 636mA/9.02V respectively. This corresponds to an idle power use of 5.56W and a peak power use of 5.74W. Power consumption of the PTC08 Camera Modules were not measured. However, with a 75mA peak current draw and a 5V DC supply voltage, it is expected that peak power use for the PTC08 Module to be at most 375mW.

Background Reading

In “An elliptical function model for fisheye camera correction,” a model is proposed for rectifying the image distortion inherent from imaging through a wide angle lens [3]. Since the camera used in this project has a large field of view, the stereo images will need to be rectified to adhere to the epipolar constraint for pixel correspondence. In order to perform a similar process, a known correction model will need to be experimentally found from images captured by the camera. Once the camera specific correction model is known, a non-linear transform can be applied to the stereo images to rectify them.

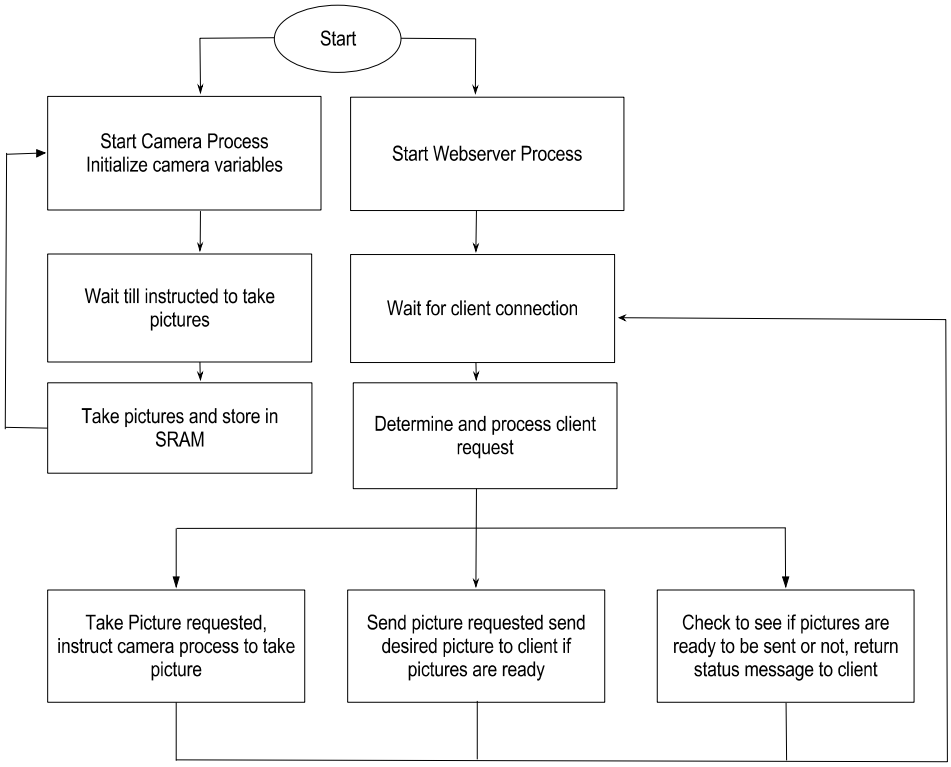
In “A pixel to pixel correspondence and region of interest in stereo vision application,” the block matching algorithm for pixel correspondence between rectified images is explained in depth [4]. Once the two images have been corrected for wide angle distortions and calibrated such that the epipolar line constraint is a permissible approximation, pixels from one image will matched along this epipolar line by finding the block in the opposite image that shows the least differences. Once a match is found, the pairs horizontal offsets from their specific camera centers will be compared to obtain their final disparity value.

In “Tutorial Camera Calibration,” obtained from the BoofCV website, an in depth process for collecting good images for calibration purposes is provided [9]. In order to know how much distortion the camera lens produces in the final image, a collection of calibration images are obtained that consist of a regular pattern that can be easily matched and has a known geometry. The ChessBoard pattern will be used to collect enough stereo images and the example stereo planar calibration program provided by BoofCV will be used to obtain more accurate estimations for the the cameras' specific focal lengths, image centers, and the baseline geometry between the two cameras. This information will improve the accuracy of the rectification process and ultimately the number of pixels than can be matched in the final disparity mapping process.

Software Design

It was determined that the best approach in order to achieve the goals of this project was to break down the main goal into a set of smaller tasks. This was decided due to limitations in hardware that became apparent as progress on the project was made. The tasks were determined to be a camera control task in order to capture images and control the cameras, a web server task which opens a connection to a client program and allows for communication between the board and client, and the client program itself which does the actual image analysis. The reason why image analysis was moved off board to a client program was because of memory limitations that would not allow the desired image processing libraries on the board. Moving the image analysis off board meant that a way to transfer data off the board to the client side process was needed. It was determined that the best approach to send the data to a off board process was through the use of a server-client relationship between the board and client program. It was determined that a camera control task would need to be separate from the web server task due to the amount of time needed to capture a image. If the camera control was integrated into the server task then when the board was taking a picture it would not be able to respond to requests made by the client. For this reason camera control and server was separated so that a client could still be served while images were being captured. As the software was divided into different tasks it was determined that a RTOS was needed and as mentioned above uC/OS-II was used in order to accomplish the multitasking. Each of the tasks involved in this project are described in the sub sections below and the software process flow diagram in the figure below shows the process of each of the on board tasks.

Figure 3: Software Diagram



Camera Control Task

The camera control task is used to initialize and control the cameras. The task functions by initializing a StereoCamera class which allows each of the of the cameras to be controlled through single method calls. The board communicates with the cameras through commands sent over the UART. The driver software for the UARTs provided by Altera was used so that a file streams could be opened and used instead of having to write to registers directly. The driver was interrupt based and handled all of the actual sending and receiving of data behind the scenes. Once the cameras were initialized the camera control task would pend on a semaphore which is posted upon when a request to take a picture is made. Once the semaphore is posted upon the task is able to run where it will send commands to stop the cameras from taking video. Image data is then read from the frame buffer of the cameras into SRAM memory and the cameras are reset. Once this process is complete it signals that the pictures have been taken by posting on a separate semaphore, the cameras are returned to video mode, and the task once again waits for a request to capture a image.

Web Server Task

The web server task utilized to communicate with and send data to the client works in the following way. It sets up a listening socket to set up incoming connections from client processes. Once a connection has been made a new socket is set up to process the request from the client program. Once the request has been processed a response is returned to the client program over the socket made to receive the request. The valid requests that a client program can make are take pictures, check if pictures are ready, download the pictures, and

retrieve a file from a read only file system. For taking a picture the server task posts on the semaphore that the camera control task pends on, and then returns a status message to the client program. When checking if the pictures are ready to download the server task checks the status of the semaphore which signifies that the pictures have been successfully downloaded and returns a status message to the client. Downloading the pictures is separated into downloading the right and left image separately and each image is sent across the connection in chunks. Sending files from the read only file system works in the same way, the server task finds the file and then sends it in chunks. There are also a standard messages which are sent if the request is not recognized.

Image Analyzing Client Process

In order to identify two pixels as being equivalent, several approaches can made. The most exhaustive and thorough method is to perform a greedy matching of every pixel in the right image along the same row (an epipolar line) to the single pixel chosen in the left image. The pixel that requires the least amount of matching cost is identified as the matching pixel and the algorithm continues for each pixel found in the left image. However, this method involves a great deal of ambiguity since several pixels on the same row may have identical color and gradient information, ensuring that several pixels have the least matching cost and forcing the algorithm to make faulty choices. An improved approach is to group several consecutive pixels on an epipolar line into a set and perform the same greedy matching technique with equal sized sets on the opposite image. This improves the information quality for every portion of the image and reduces the number of similarly matched sets. Both of these approaches are exhaustive in nature and would result in very large computation delays. Reducing the number of matches to improve the speed of this algorithm can be performed by only searching for sharp changes in pixel information between consecutive pixels along an epipolar line. These sharp changes would correspond to an edge of an object found within the image. Identifying only edges of the objects within the image would greatly reduce the number of regions to match at the cost of exhaustive depth analysis. The resulting matched regions would only correspond to the outlines of the objects found within the image and the following depth information only provided for those outlines. The BoofCV library handles the window-based matching algorithm as well as the rectification process based on experimentally obtained focal lengths, image centers, skews, and spatial orientations of the stereo camera system. The Client Machine Program, is a sequential design that waits upon the user to request a live capture, allows the user to poll the web server to see if images are available, and finally download ready images for disparity mapping which is finally displayed on the client machine's display.

Test Plan

Software:

Testing Pixel Correspondence Algorithm

Will start with a controlled environment that has one object of a single color against a sharply contrasted background. Accuracy of the matched pixels will be observed on the output image. Accuracy of matched pixels will be stressed by adding more objects to the control environment. Finally the algorithm will be stressed on real environments where the contrast of objects against their background will not be controlled.

Testing Interniche Web Server

Once the client starts the provided executable StereoMatch.jar file, the program will connect with the DE2 webserver. When the client requests a set of the precaptured demo images, the client machine will download the requested images from the read only file system and perform the pixel correspondence algorithm. When the client requests a live capture, they are allowed request a status message from the server while the DE2 Board collects the images from the cameras, when the images have been collected, and the client makes another request for download, the client machine will download the live images from the DE2 Board and perform the pixel correspondence algorithm.

Memory Testing

Collecting image information over a serial connection requires the need to ensure proper read and write functionality of the RAM workspace. Testing the ram usage at startup as well as after an extended period of time will be performed to ensure collection of data and its storage does not interfere with later computations.

Hardware:

Functional Test Procedure

Starting from cold boot, and the device started will initialized the attached cameras and wait upon a client request. When the user initiates a depth capture, the disparity mapping will be presented in an image to the client's display. This initiation of depth capture can occur at the user's discretion and does not affect the performance or functionality of the device.

Calibration of Camera Modules

Ensuring that the two cameras have been calibrated to have identical focal points and vertical positions is crucial to ensure the epipolar constraint of the pixel correspondence method is satisfied. After physical setup of the two cameras has been performed, images will be taken and compared to determine the differences in the cameras' vertical positions, focal points, and orientations. Accommodations to these differences will be adjusted physically and the

comparison process performed again. This will be iterated until the necessary qualities of the images captured by both cameras are met.

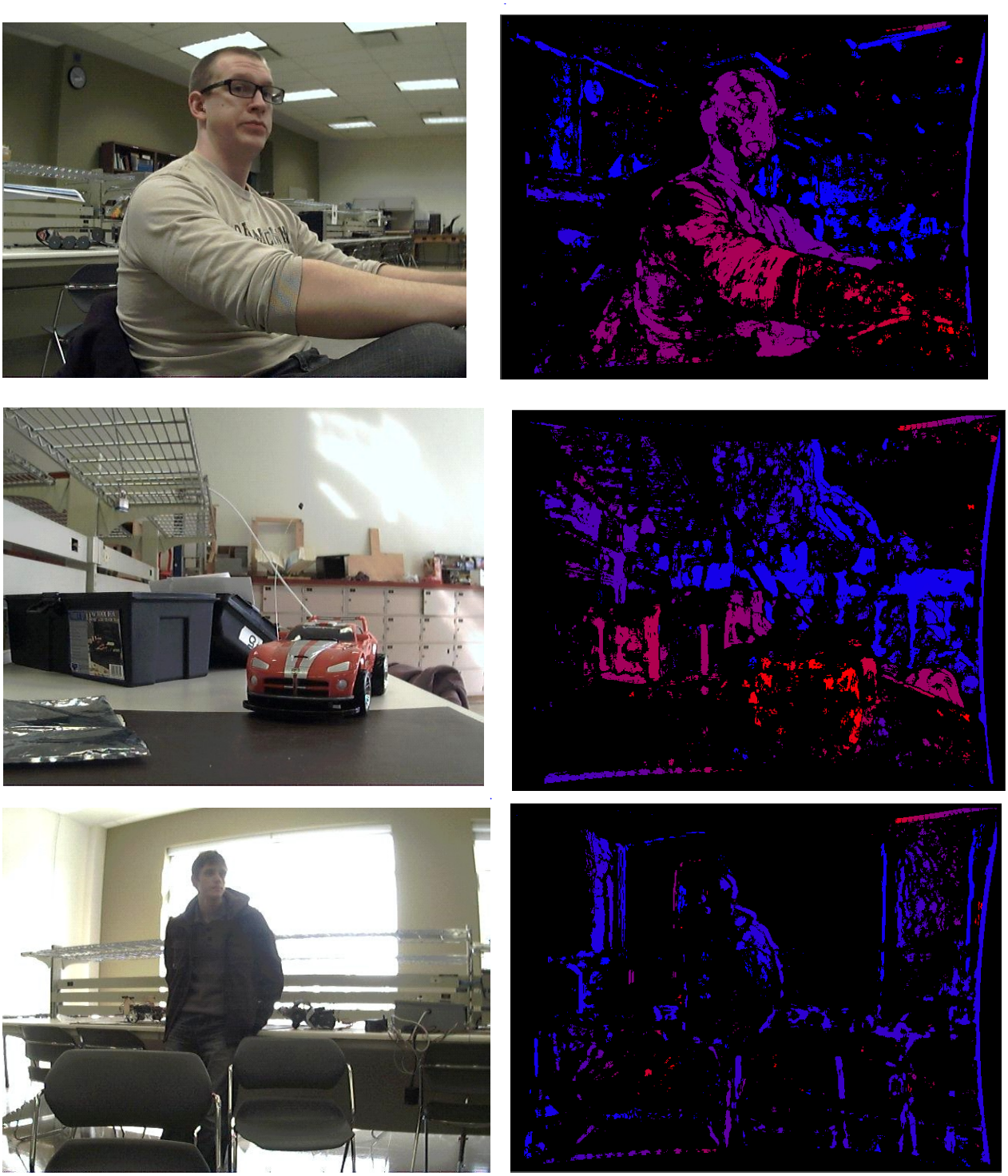
Description of Operation	Expected Performance (Expectations Met?)
Obtain readable images over Ethernet interface on client machine	Download for a single image takes 2 seconds (Tested and Met)
Images captured from both cameras are collected and processed, the output image is presented on client machine's display	Once the user has initiated a capture the output image should be available on their display within 1 minute (Tested)

Results of experiments and characterization

Performance of the serial interface between the DE2 board and the peripheral cameras proved to be the largest bottleneck. The provided UART cores on the Nios II system provide internal transmit and receive buffers and can be read from/written to using the HAL API as though writing to or from a file pointer in memory. These drivers were modified to ensure error checking due to buffer overrun. Reading a larger number of bytes at a time from these internal buffers increased the frequency of buffer overrun errors that resulted in erroneous artifacts appearing in the final collected JPG file, however larger data reads significantly improved the latency of the data collection from the Camera modules. Reading from the buffer in 64 byte chunks allowed for data collection from both cameras to occur within 50 seconds, although the greatest number of erroneous artifacts were present in the final image. Reducing the read chunk to 16 bytes, removes all of the erroneous artifact occurrences, but increased the latency of the data collection to 1 minute and 50 seconds to 2 minutes.

On the following page the resulting input and output images taken using the Stereo Camera System. The Left image is the unmodified captured image while the right image is the modified disparity mapping representation calculated from both the left and right (not shown here) images.

Figure 4: Unmodified Images and Final Disparity Mapping Image For various Testing Environments



As can be seen above, the resulting success rate of the disparity mapping is largely dependent on the environment being captured. Objects that exhibit a larger variety of color variations and textures were more easily matched.

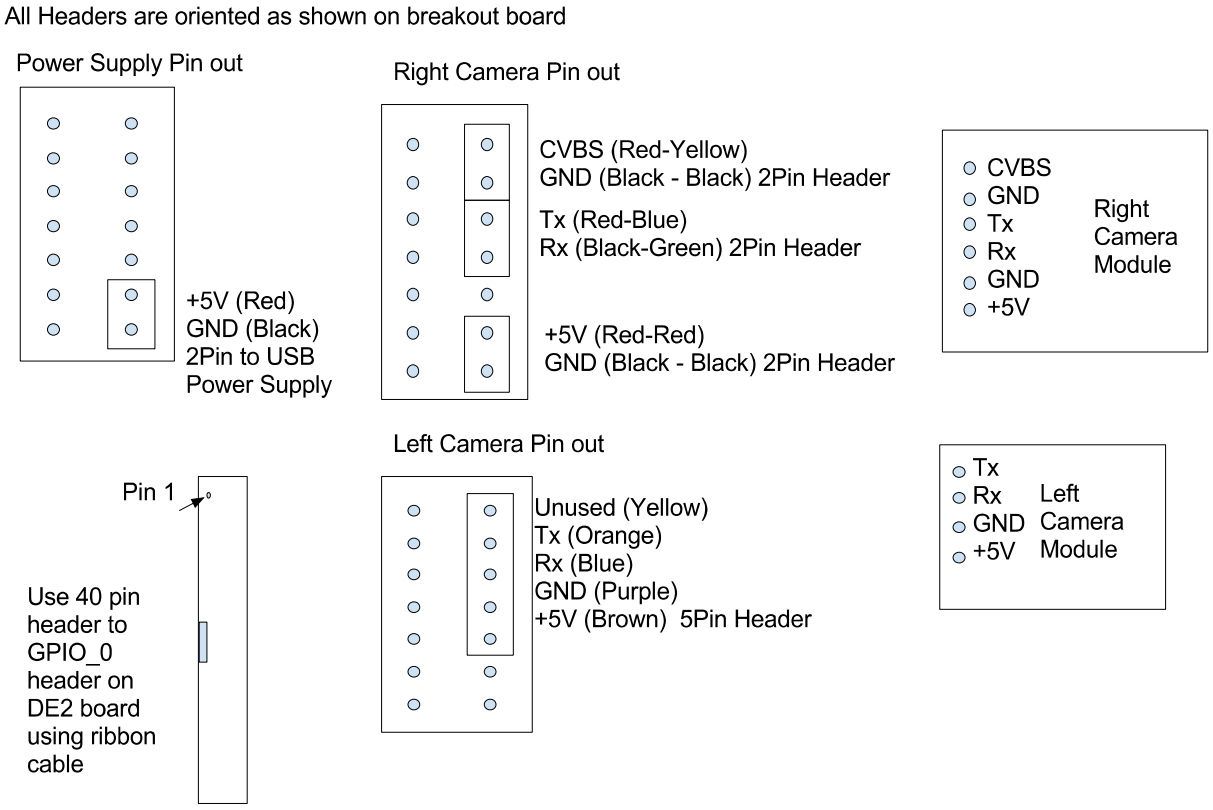
References

1. H. Zhu, X. Wang, C. Yi. "An elliptical function model for fisheye camera correction." *Intelligent Control and Automation (WCICA). 2011 9th World Congress on*. pp.248-253. 21-25 June 2011.
2. “OpenCV (Open Source Computer Vision.” Internet: http://opencv.org/, Nov. 3, 2012 [Feb 3, 2013].
3. L. Fried et al. “Adafruit-VC0706-Serial-Camera-Library.” Internet: https://github.com/adafruit/Adafruit-VC0706-Serial-Camera-Library, [Feb. 14, 2013].
4. R.A. Hamzah, K.A.A. Aziz, A.S.M. Shokri. "A pixel to pixel correspondence and region of interest in stereo vision application." *Computers & Informatics (ISCI). 2012 IEEE Symposium on*. pp.193-197. 18-20 March 2012.
5. D. Jacobs et al. “Stereo Vision: Reading Chapter 11.” Internet: http://www.cs.ubc.ca/~lowe/425/slides/6-Stereo-6up.pdf, [Feb. 2, 2013].
6. “Altera Corporation.” Internet: http://www.altera.com/ , [Apr. 4, 2013].
7. “BoofCV.” Internet: http://boofcv.org/index.php?title=Main_Page, Feb.16, 2013 [Mar. 14, 2013].
8. “Index of /~elliott/ece492/projects/2012w/g8_Facial_Recognition.” Internet: http://www.ece.ualberta.ca/~elliott/ece492/projects/2012w/g8_Facial_Recognition/, [Mar. 13, 2013].
9. “Tutorial Camera Calibration,” Internet: http://boofcv.org/index.php?title=Tutorial_Camera_Calibration, Mar. 30, 2013 [Apr. 1, 2013].

Appendices

Quick Start Manual

Figure 6: Hardware connection diagram for interfacing Camera mount to Altera DE2-115



DE2 board Usable peripherals:

KEY(0) = Reset Button

SW(0-7) = Read Line length, read in binary form, ie: when SW(6) is on and all others are off, this corresponds to a readline length of 64bytes and will be the fastest data collection rate from the cameras over UART. The shortest programmed readline length is 16 bytes and can be obtained with all SW(0-7) in the off position, this is the slowest data collection rate. Readline lengths 64, 32, and 16 are the most reliable and have been tested.

For both versions of setup, follow the diagram to setup the camera mount and connect it to the DE2 board. Setup the client machine to have a static IP address as follows:

IP: 192.168.1.1

Subnet Mask: 255.255.255.0

Gatway: 192.168.1.1

Connect the client machine to the DE2 board using an Ethernet Cat5 or 6 cable. Ensure the client machine can execute java version 1.7 runtime environment from the command line and include an “image” folder in the same directory as the StereoMatch.jar from where the client program will be run.

For running the volatile version:

1. Open the included Quartus project and program the DE2 board with the webServerFlashFinal.sof file over a JTAG connection.
2. Open the Nios II Software Build Tools for Eclipse, via the SOPC Builder
3. Flash the software/system/ro_zipfs.zip file using the Nios II Flash Programmer, ensuring its offset is 0x300000
4. Run the StereoCameraServer.elf as “Nios II Hardware”
5. Run the client StereoMatch.jar executable on the client machine using “java -jar StereoMatch.jar”

For running the flash version:

1. Open the included Quartus project and flash the DE2 board with the webServeFlashFinal.pof file over an Active Serial Connection.
2. Open the Nios II Software Build Tools for Eclipse, via the SOPC Builder
3. Flash the software/system/ro_zipfs.zip file using the Nios II Flash Programmer, ensuring its offset is 0x300000
4. Flash the StereoCameraServer.elf using the Nios II Flash Programmer, ensuring it has no offset.
5. Run the client StereoMatch.jar executable on the client machine using “java -jar StereoMatch.jar”

Once both the client machine running the client program and the board have been properly set up the project can be run by starting the client program. The client program should prompt you to press enter to start a new thread and then display a list of possible options. Entering “capture” should cause a request to take a picture be made and the green LEDs on the board should turn off. The pictures will not be ready to download until all 8 LEDs turn back on. Once the LEDs are on enter “c” and the images should be downloaded and analyzed. If errors are occurring they can be reduced by switching only switch 4 to the on position and trying a image capture again. The switches are used to set the number of bytes downloaded from the camera to the board at a time. Switch 4 is used for 16 bytes, switch 5 is used for 32 bytes and switch 6 for 64 bytes. The larger size while quicker is prone to more errors being encountered in the images.

Note: If the client program is run and makes a request to capture images immediately after the on board tasks are started up the images are likely to be unreadable, and it is best to give the board a minute to start up.

Future Work

As the project was not able to compute actual distances of objects in it's current state it is proposed that the triangulation process described below used to find the actual distance of objects from the disparity map this project generates. In addition it is also believed that performance can be improved by increasing the baud rate at which images are downloaded from the camera, and introducing better error checking into the Altera UART module and software drivers.

Triangulation can be used to determine the distance *d* an object is away from the camera setup. This is done by finding the angle that the object is located at in reference to a common base line at two points along said line. The intersection point of the lines that are formed from these two angles is then the point at which the object is located. The three points then are able to make a triangle where two of the angles and the length of one side are known. The two angles are the ones measured, and the length of the side is found by calculating the distance along the reference line between the two points, where the two angles are measured. Using this data the third angle can be found and using a combination of trigonometric formulas the exact location of the third point can be calculated. However, as we only need to find the distance the third point is from the line of reference, the calculations become slightly simplified. Before this distance can be calculated the first two angles must be found, this is done by knowing the object's pixel based location (that is it's x pixel coordinate) on each of the images provided by the two cameras. Once the pixel based location is found we need to find the angle it corresponds to, this is done by using the following mathematical formulas and application of logic.

Given a resolution and an angle of view the radians (or degree) per pixel can be calculated with the following formula:

$$\frac{rad}{pixel} = \frac{view\ angle}{image\ width} \qquad \text{equation (2)}$$

Assuming the specifications on our camera are accurate this gives us:

$$\frac{60^o * \pi\, rad}{640\ pixels * 180^o} = 0.00163624617\ \frac{rad}{pixel} \qquad \text{equation (3)}$$

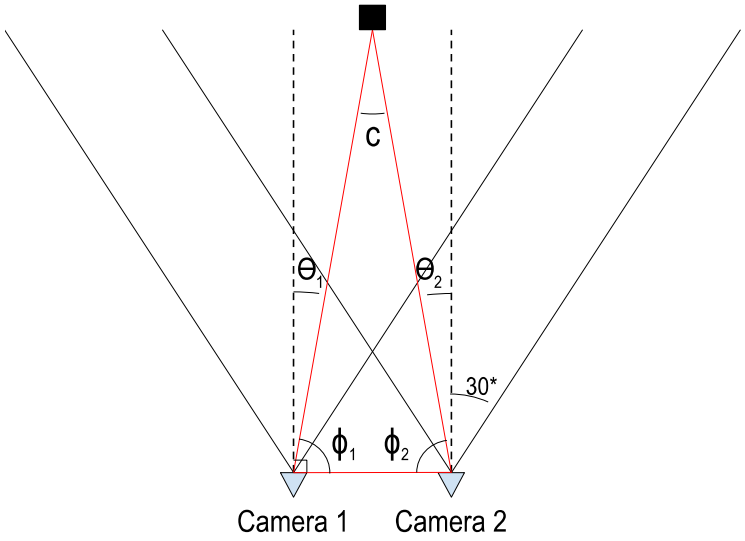
Since there are an even number of pixels in the video input there are 320 pixels on either side of the centre of the field of view of the camera each pixel accounts for a offset of 0.00163624817 radians farther than the previous pixel offset. Thus, if the edge of any given pixel corresponds to a particular angle Θ, then the other edge of the pixel corresponds to a angle Θ±0.00163624817 radians. With this in mind we shall say that the edge between the centre pixels of a camera is 0, and according to the above logic the opposing edge of one of the centre pixels would then be at an angle of ±0.00163624817 rads; furthermore, the centre of one of these pixels can then be said to be 0.00081812408 radians away from the centre line. By this reasoning we can convert a x-pixel coordinate into a corresponding angle with the following equation:

$$\theta = \frac{60\pi}{650 * 180 * 2} + |x - 320| \left(\frac{60\pi}{640 * 180} \right), 0 \leqslant x < 640 \qquad \text{equation (4)}$$

In order to calculate the actual angles of the triangle the sign (positive or negative) of the angles must also be determined. For the left camera, called camera 1, pixels to the left of centre correspond to a positive angle and pixels to the right correspond to a negative angle; however, for the right camera, called camera 2, the inverse is true (pixels left of centre correspond to a negative angle and pixels right of centre correspond to a positive one). Then assuming that the centre lines of both cameras are perpendicular to the line of reference, the

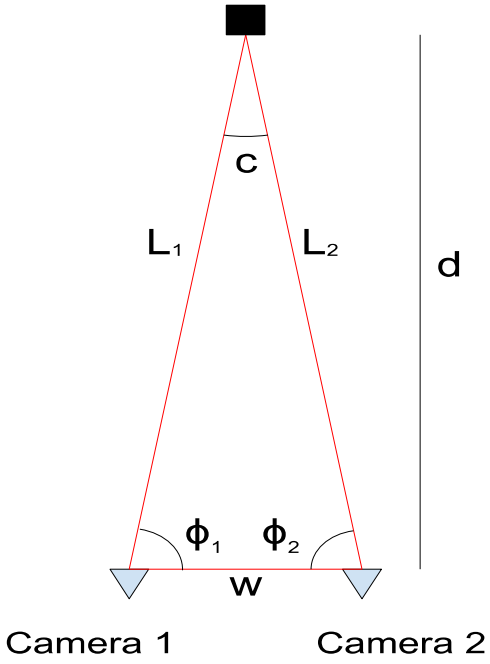
interior angle of the triangle can be found by adding $\frac{\pi}{2}$ radians to the offset. This works because the signs of Θ_1 and Θ_2 are assigned such that $\phi_i = \frac{\pi}{2} + \theta_i$ where $\phi_i, i = 1,2$ are interior angles of the triangle. The third angle, c, can then be determined due to the fact that the three interior angles of the triangle should add up to π radians. The **figure below** shows the angles and lines of reference in relation to the cameras.

Figure 7: Stereo Camera Field of View Illustration

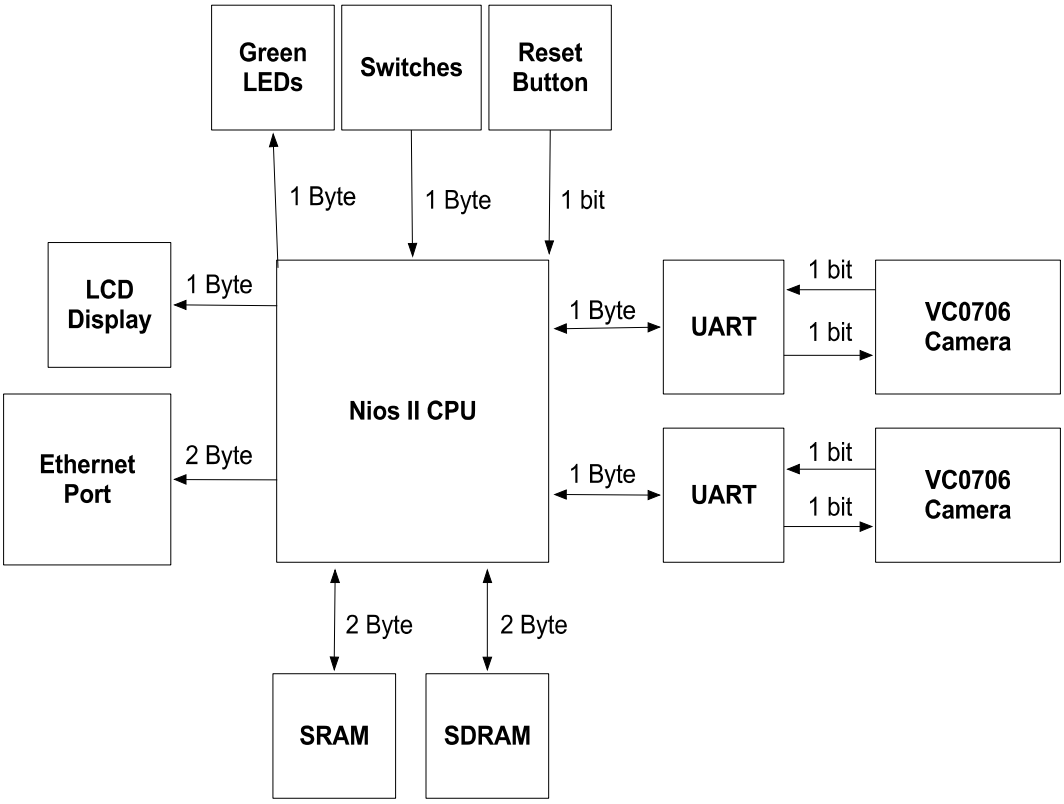


Once the the interior angles of the triangle, whose vertices are at camera 1, camera 2, and the object, have been determined the distance to the object can then be calculated. The distance is calculated using trigonometric formulas. First the sine law is used to find the lengths of each of the sides. The sine law states that $\frac{A}{\sin(a)} = \frac{B}{\sin(b)} = \frac{C}{\sin(c)}$ where *A*, *B*, and *C* are the lengths of each of the sides; *a*, *b*, *c* are the interior angles of the triangle such that *A* is opposite of *a*, *B* is opposite *b*, and *C* is opposite *c*. It can be used to find the lengths of the sides, L_1 and L_2, given that we already know the interior angles of the triangle and the length of the third side, the distance between camera 1 and camera 2. Calculating the other lengths of the other sides is a matter of plugging the known values into the formula and solving for the unknowns. Once the length of each side is found the distance to the object is calculated by solving the sine formula for right triangles for the opposite side length. The sine formula, $\sin(\phi_i) = \frac{opposite}{hypotenuse}$, is set up such that ϕ_i is the more acute angle between ϕ_1 and ϕ_2 and the opposite side length is the distance d. The figure below shows the relationship between the values talked about with respect to the cameras.

Figure 8: Triangulation Illustration

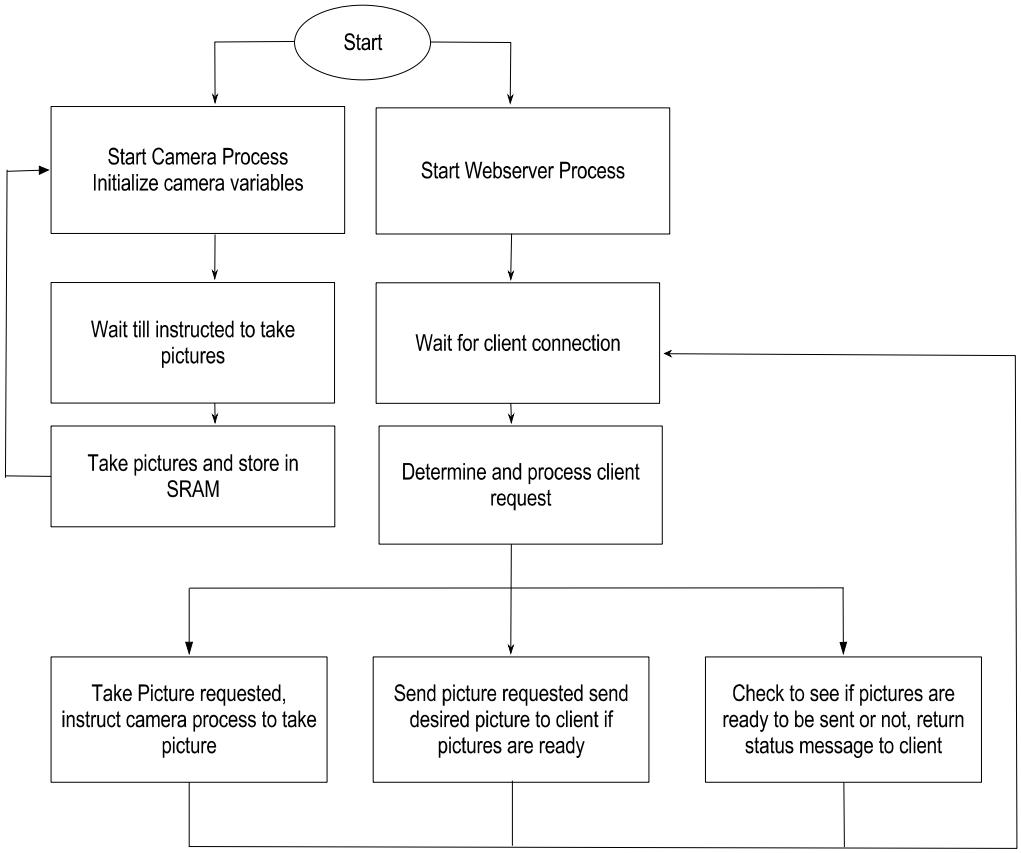


Hardware Documentation



Source Code Section

Interactions Between Processes



Index To Source Code
(denoted below as File in index {N|B|E|T})
Comments for file precedes the file and is prefixed with #

In directory StereoMatchingClient

Main compiled executable
StereoMatch.jar {T}

image directory for Main executable
image/

BoofCV and its required libraries
StereoMatch/lib/BoofCV.jar {T}
StereoMatch/lib/boofcv_examples.jar {T}
StereoMatch/lib/DDogleg.jar {T}
StereoMatch/lib/EJML.jar {T}
StereoMatch/lib/GeoRegression.jar {T}

source code for client program
StereoMatch/src/StereoMatch/server.java {T}
StereoMatch/src/StereoMatch/StereoMatch.java {T}

In directory StereoCameraServer
#Ethernet Core and its required drivers
dm9000a/HAL/inc/... {T}
dm9000a/UCOSII/src/... {T}
#webserver software
software/StereoCameraServer/alt_error_handler.c {T}
software/StereoCameraServer/alt_error_handler.h {T}
software/StereoCameraServer/create-this-app {T}
#main request handler that provides process flow for initialize image capture
software/StereoCameraServer/http.c {T}
software/StereoCameraServer/http.h {T}
software/StereoCameraServer/image.h {T}
software/StereoCameraServer/Makefile {T}
software/StereoCameraServer/network_utilities.c {T}
software/StereoCameraServer/srec_flash.c {T}
software/StereoCameraServer/web_server.c {T}
software/StereoCameraServer/web_server.h {T}
#Stereo Camera API that simplifies camera function calls in the web server
software/StereoCameraServer/StereoCamera.cpp {T}

#Camera API that interfaces the camera communication over the UART Cores for image
capture and camera management
software/StereoCameraServer/Camera.cpp {T}
#Defined Camera task that uses semaphores to facilitate image capture without blocking
the interniche web server
software/StereoCameraServer/cam_control.cpp {T}
software/StereoCameraServer/StereoCamera.h {T}
software/StereoCameraServer/Camera.h {T}
software/StereoCameraServer/cam_control.h {T}
#Modified UART Core functions to improve error handling that reduces image capture
errors, they will take precedence over the same driver functions produced by BSP
software/StereoCameraServer/altera_avalon_uart_read.c {T}
software/StereoCameraServer/altera_avalon_uart_init.c {T}

#BSP is required for generating HAL drivers and underlying sourceware for the
interniche web server
software/StereoCameraServer_bsp/alt_sys_init.c {T}
software/StereoCameraServer_bsp/libucosii_bsp.a {T}
software/StereoCameraServer_bsp/linker.h {T}
software/StereoCameraServer_bsp/linker.x {T}
software/StereoCameraServer_bsp/Makefile {T}
software/StereoCameraServer_bsp/system.h {T}
software/StereoCameraServer_bsp/drivers/inc/... {T}
software/StereoCameraServer_bsp/drivers/src/... {T}
software/StereoCameraServer_bsp/HAL/inc/... {T}
software/StereoCameraServer_bsp/HAL/src/... {T}
software/StereoCameraServer_bsp/iniche/inc/... {T}
software/StereoCameraServer_bsp/iniche/src/... {T}
software/StereoCameraServer_bsp/UCOSII/inc/... {T}
software/StereoCameraServer_bsp/UCOSII/src/... {T}

VHDL Source

 # In directory StereoCameraServer:

#main flash and volatile hardware compilations
webServerFlashFinal.pof
webServerFlashFinal.sof

#top level vhdl of system
webServer.vhd
#system generated components using sopc builder
altpll_inst.vhd
#cpu is a fast version of the nios II processor
cpu.vhd
cpu_inst.vhd
cpu_inst_jtag_debug_module_sysclk.vhd
cpu_inst_jtag_debug_module_tck.vhd
cpu_inst_jtag_debug_module_wrapper.vhd
cpu_inst_mult_cell.vhd
cpu_inst_oci_test_bench.vhd
cpu_inst_test_bench.vhd
cpu_jtag_debug_module_sysclk.vhd
cpu_jtag_debug_module_tck.vhd
cpu_jtag_debug_module_wrapper.vhd
cpu_mult_cell.vhd
cpu_oci_test_bench.vhd
cpu_test_bench.vhd
dm9000a.vhd
dm9000a_inst.vhd
high_res_timer.vhd
jtag_uart.vhd
lcd_display.vhd
led_pio.vhd
niosII_system.vhd
niosII_system_burst_0.vhd
niosII_system_burst_1.vhd
niosII_system_burst_2.vhd
niosII_system_burst_3.vhd
niosII_system_burst_4.vhd
niosII_system_burst_5.vhd
niosII_system_burst_6.vhd
niosII_system_burst_7.vhd
niosII_system_burst_8.vhd
niosII_system_burst_9.vhd
niosII_system_burst_10.vhd
niosII_system_burst_11.vhd

niosII_system_burst_12.vhd
niosII_system_burst_13.vhd
niosII_system_burst_14.vhd
niosII_system_burst_15.vhd
niosII_system_burst_16.vhd
niosII_system_burst_17.vhd
niosII_system_burst_18.vhd
niosII_system_burst_19.vhd
niosII_system_burst_20.vhd
niosII_system_burst_21.vhd
niosII_system_burst_22.vhd
niosII_system_clock_0.vhd
niosII_system_clock_1.vhd
niosII_system_clock_2.vhd
niosII_system_clock_3.vhd
niosII_system_clock_4.vhd
niosII_system_clock_5.vhd
niosII_system_clock_6.vhd
niosII_system_clock_7.vhd
niosII_system_clock_8.vhd
niosII_system_clock_9.vhd
niosII_system_clock_10.vhd
niosII_system_clock_11.vhd
niosII_system_clock_12.vhd
niosII_system_clock_13.vhd
niosII_system_clock_14.vhd
niosII_system_clock_15.vhd
niosII_system_inst.vhd
nvram_controller.vhd
nvram_controller_0.vhd
onchip_memory2_0.vhd
onchip_memory2_inst.vhd
sdram.vhd
seven_seg_pio.vhd
switch.vhd
sysid.vhd
altpll_inst.vhd
cpu_jtag_debug_module_sysclk.vhd
cpu_jtag_debug_module_tck.vhd
cpu_jtag_debug_module_wrapper.vhd
cpu_mult_cell.vhd
cpu_oci_test_bench.vhd
cpu_test_bench.vhd
high_res_timer.vhd

jtag_uart.vhd
lcd_display.vhd
led_pio.vhd
memory.vhd
niosII_system_inst.vhd
niosII_system.vhd
sdram.vhd
seven_seg_pio.vhd
#custom vhdl glue logic for interfacing with the onboard SRAM chip
sram_IF.vhd
sram_IF_0.vhd
switch.vhd
sys_clk_timer.vhd
sysid.vhd
uart_0.vhd
uart_1.vhd