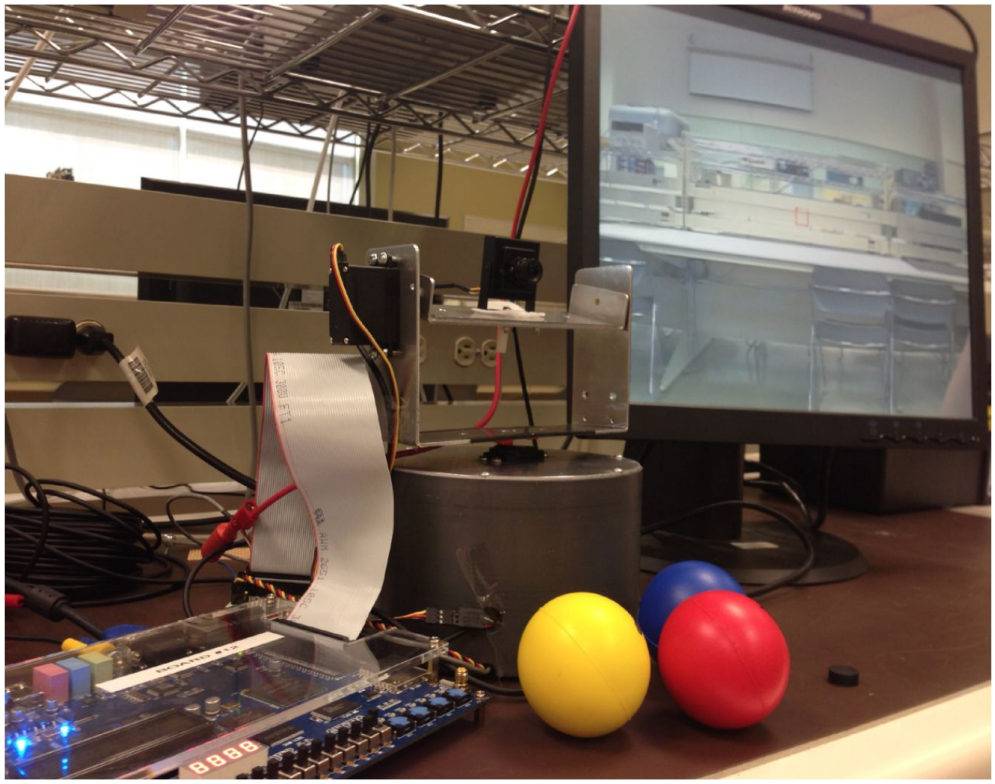
Coloured-Object Tracking Camera

ECE 492 Design Project Final Report

A rotating camera which detects and tracks an object via specific colour.



Ryan Corpuz | rcorpuz@ualberta.ca
Hang Peng | hpeng2@ualberta.ca
Jingjing Liang | jliang3@ualberta.ca

Abstract:

The objective of this project is to design a camera mount that can continuously track an object by its specific colour. Using image processing, it is capable of continually centering the trained colour of the image in the frame. Using servo motors, the camera mount is able to track the colour with two dimensions of motion, vertical tilt and horizontal pan. The camera captures the live video and compares each pixel within a threshold value we set in using the HSV (Hue, Saturation, Value) image format for continuously tracking the object when light intensity changes due to varying lighting conditions. The design is implemented on a Altera Terasic DE2 development board. The coloured-object tracking camera has been successfully implemented, meeting most of our functional requirements.

Table of contents

FUNCTIONAL REQUIREMENTS

The project is to implement a coloured-object tracking camera, which will use a camera to view an area, recognize the designated colour to be tracked, and adjust the camera's orientation to keep that particular coloured object within the frame of the camera.
The functionality of this project involves interfacing the camera as the input for the DE2 board, and servo;, which the camera is mounted to, as the output for repositioning. A VGA monitor for the output of the camera image would be used for demonstration and debugging purposes.

Detecting Set Colour in Image

The camera will take composite video signals from camera and send it to the TV decoder chip built into the Altera DE2 board through the video-in port. The frame data will then go through a series of colour space conversions, and finally to HSV format signals used for threshold comparison. From there, we will compare each pixel's colour to the target colour predefined in the system.

Position of Coloured Object in Image

After threshold comparison, the system records the first and last pixel which pass the comparison algorithm as each pixel goes into the image processor. At the end of the frame, the center of the moving object can be calculated by averaging x and y coordinate of first and last pixels.

Horizontal and Vertical Tracking

After computing the position of the center of the object from the image processor, the system will calculate the displacement between the center of the frame and the centre of the target. It then converts the displacement further into the moving angles of the servo motors. The servo motor will take the instructions (of the proper angle) and drive the camera to face the centre of the target.

Monitor Implementation

The system outputs the image which is received from the camera to a monitor using the VGA cable. The monitor can illustrate the functionality of the tracking system. As an extension, it can output the threshold image, showing the pixel which passes the threshold comparison as white, and everything else black. This will be used for debugging, testing, and demonstrating purposes.

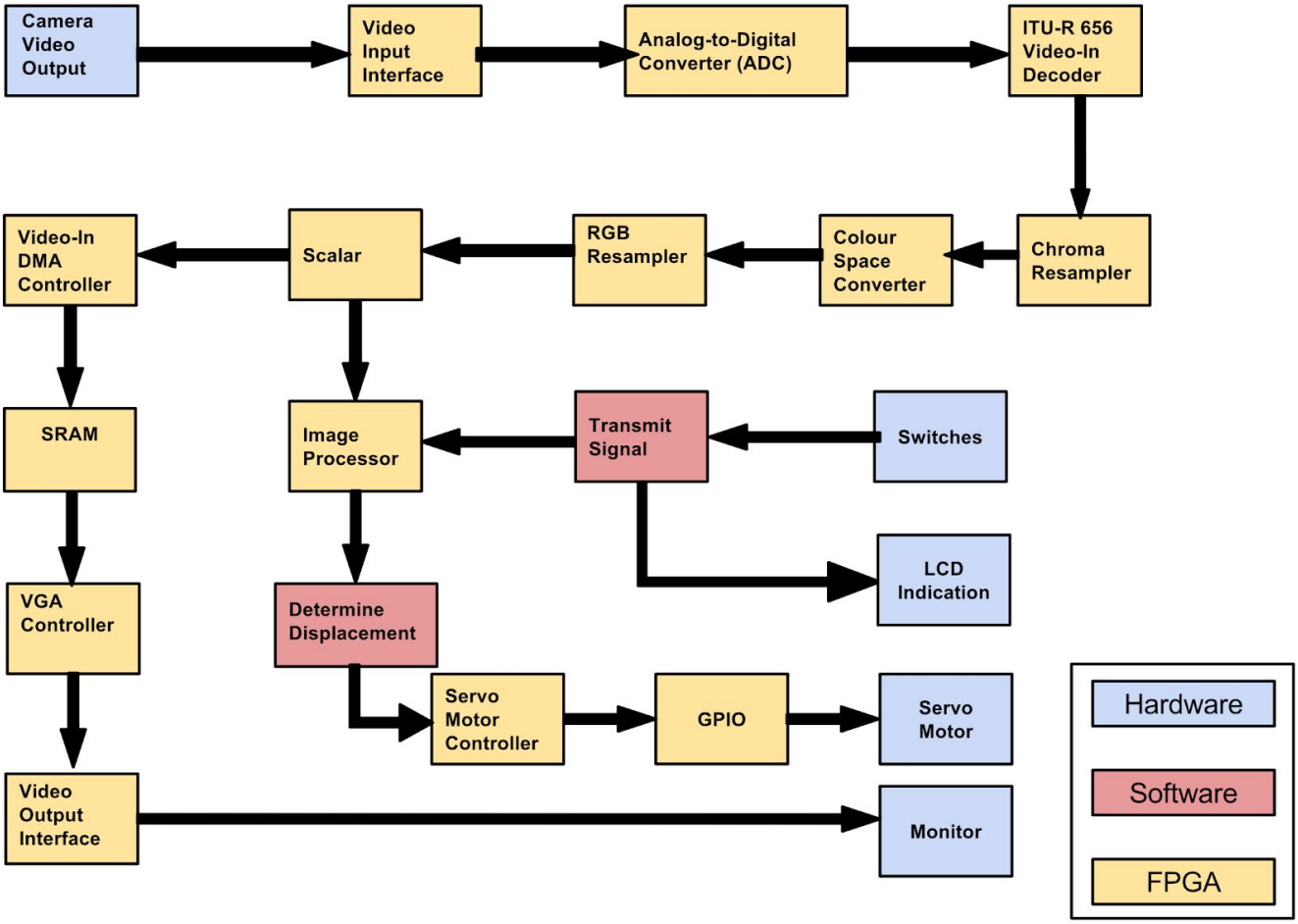
Changing the Colour Being Tracked

During operation, users can change the colour that is being tracked by flipping a configure switch. To achieve this, the system will clear the old colour being tracked and change it to the new one by capturing the colour from center of frame.

Movement lock

Horizontal and vertical direction lock are implemented into the tracking system and this functionality is controlled by switches. When user locks one direction, then the tracking camera is only able to track the object along with the other direction.

DESIGN AND DESCRIPTION OF OPERATION



This is a block diagram of our design. The external video and other peripherals are marked as blue, and the hardware components of FPGA are marked as yellow, and the red blocks indicate the process we did in the software layer.

Hardware Layer and Data flow

NTSC input signals from the camera will go to the FPGA board through composite video cable. The stream of signals is firstly passed into the analog-to-digital converter chip to normalize data to digital signals. Then, it will be decoded and converted into YCrCb (4:2:2) colour space. The chroma resampler will convert the aspect ratio of YCrCb (4:4:4) with the original frame resolution. After that, it will be further converted into the 24-bit RGB format signals. The above processes mentioned is predefined by Video IP cores in Altera University Program. After scaler, one stream of 16bit 320 x 240 RGB format signals is written directly into the video buffer as the output for display, while the other stream of signals is passed into image processor. RGB format is converted into HSV to have threshold comparison with predefined threshold value and the register will record the position of first and last pixels which pass the comparison. When it reaches the last pixel of the frame, the center of the tracking object can be calculated by averaging the distance of x and y directions. The center coordinate is retrieved by software layer to calculate the displacement between the center of the frame and the center of the

object and convert it to the number of degrees that servo motor need to move in order to re-center the object. The horizontal and vertical movement instructions are then sent to another hardware component (called a servo motor controller in separate tasks). The servo motor controller is a pulse width modulator, which sends proper pulse widths to GPIO for driving the servo motors to track the target. Also, switches are implemented to define the threshold value of the image processor, enable the threshold image on the display, and lock either horizontal or vertical direction of the tracking system. The current state of the tracking system and some brief instructions about how to set the colour will be shown on the LCD screen.

RGB to HSV Conversion

The reason why we choose HSV instead of RGB to do the threshold comparison is because HSV separates the colour value from the intensity of lighting. Therefore, the lighting condition will not affect the HSV format pixels seriously. The main conversion algorithm is implemented as a part of image processor FPGA component. The conversion is processed according to the formula:

$$S = MAX(R,G,B) - MIN(R,G,B)$$
$$V = MAX(R,G,B)$$
$$H = 0, \ (if\ V = 0)$$
$$H = \frac{G-B}{S} \times 60°, \ (if\ MAX = R)$$
$$H = \frac{B-R}{S} \times 60° + 120, \ (if\ MAX = G)$$
$$H = \frac{R-G}{S} \times 60° + 240, \ (if\ MAX = B)$$

We process one pixel at a time, used immediately to do the threshold comparison which is the second part of image processor.

Threshold Comparison

In this part, we will take the H, S, V values converted from RGB format pixels to compare a threshold range predefined in the system. This range is calculated by a threshold value defined by users, plus and minus a reasonable range. This range is determined according to many tests because we need to set a considerable range to avoid both of over-shrinking the relevant pixels and having too much noise.

$$(H_{threshold} - H_{range} \leq H \leq H_{threshold} + H_{range}) \cap (S_{threshold} - S_{range} \leq S \leq S_{threshold} + S_{range}) \cap$$
$$(V_{threshold} - V_{range} \leq V \leq V_{threshold} + V_{range})$$

Orientation

While the processor goes through each pixel in a frame, it will record the first pixel which passes the threshold comparison, and keeps updating the upcoming passed pixels as the last pixels. After processing the whole frame of pixels, the center of the object will be located by taking average of x and y of both the first pixel and the last pixel. And the output of the coordinate is stored in the register and can be retrieved.

$$x_{centre} = \frac{x_{first} + x_{last}}{2} \qquad y_{centre} = \frac{y_{first} + y_{last}}{2}$$

Displacement Calculation

This displacement can be calculated by subtracting the center of the frame by center of the moving object which is obtained from the previous step. The relative displacement then is calculated by subtracting the current position to the previous position. Then, we need to convert the oriented displacement to the number of degrees which the servo motor needs to rotate and send it to Servo Motor Controller. Horizontal and vertical direction are using different tasks, with a negligible delay, to ensure the rotation is synchronized.

Pulse Width Modulator

The Servo Motor Controller generates proper pulse widths to drive the servo motors by reading the data calculated from the software layer. For our servo motors, we have 1.5ms pulse width to generate a neutral position signal, running on a 50Mhz clock rate, and we need to count 75000 to generate that instruction. As well, the boundaries can also be calculated using clock rate to multiply the desired pulse width. With one degree of rotation matching a certain number of counts, the servo motor is able to accomplish the tracking by pointing to the center of object.

Bill of Materials

Qty	Part	Specification	Supplier	Cost(CAD)
1	Altera Terasic DE2 Development Board including USB cable and 9V DC 1.3A power brick Datasheet: https://www.altera.com/support/training/university/de2.html	(See Data Sheet Site)	University	(USD $284) $357.71[1]
1	ADS-120 Home Indoor Security Camera Datasheet: http://www.swann.com/downloads/product/2017ADS-120_M120CAM041012E_web.pdf	• Viewing Angle: 53° • White Balance: Automatic • Effective Pixels: NTSC 640 x 480 • Weight: 0.2 kg	FutureShop	$39.99
1	Hitec HS-422 Servo Motor Datasheet: http://www.robotshop.com/media/files/pdf/hs422-31422s.pdf	• Operating voltage: 4.8V to 6V • Operating speed: 0.21 to 0.16 sec/60° • Stall torque: 3.3 kg·cm to 4.1 kg·cm	University	$12.18[2]
1	Hitec HS-635HB Servo Motor Datasheet: http://hitecrcd.com/products/servos/sport-servos/analog-sport-servos/hs-635hb-karbonite-high-torque-servo/product	• Operating Voltage: 4.8V to 6V • Operating speed: 0.18 to 0.15 sec/60° • Torque: 5.0 kg·cm to 6.0 kg·cm	Universty	$35.25[3]
1	Mount Fabrication Cost	Estimate cost of re-purposing an old mount: 5.5 machine shop hours → $25 x first 4 hours + $50 x 1.5 hours = $175 $175 + $3 for new faceplate = $178	ECE Machine Shop	$178.00

1	Ribbon Cable		University	$15.00
1	5V Power Supply * Required but not included in total		University	N/A
1	Mini-breadboard		Team Member	$5.00
2	Alligator Clips		University	$1.50
2	Power Supply Cables		University	$20.00
			Total Cost:	$ 664.63

1. US price converted using exchange rate of 1 US : 1.26 CAD as of April 13, 2015 at 3:16 pm
2. Price collected from http://www.robotshop.com/ca/en/hitec-hs422-servo-motor.html; used regular price
3. US price collected from http://www.hobbyhorse.com/hitec_hs635hb.shtml converted using exchange rate of 1 US : 1.26 CAD as of April 13, 2015 at 3:16 pm

REUSABLE DESIGN UNITS

1. Altera University Program Video IP cores [1]

Used to convert the NTSC input video signals into the 320 x 240 16-bit RGB formal colour space stored in the video buffer (SRAM) and then output the signals to the monitor.

2. “Tutorial of displaying pixels on VGA monitor”[2]

This tutorial explain the basic procedure on how to set up the video IP core from Altera University Program and demonstrate how to using build-in functions to write a pixel intot the video buffer.

And the video buffer set-up functions are based on this source code.

BACKGROUND READING

1. “Development of a Generic RGB to HSV Hardware” [3]

Notes: This paper overviews the conversion from RGB to HSV. It also describes hardware to convert RGB to HSV.

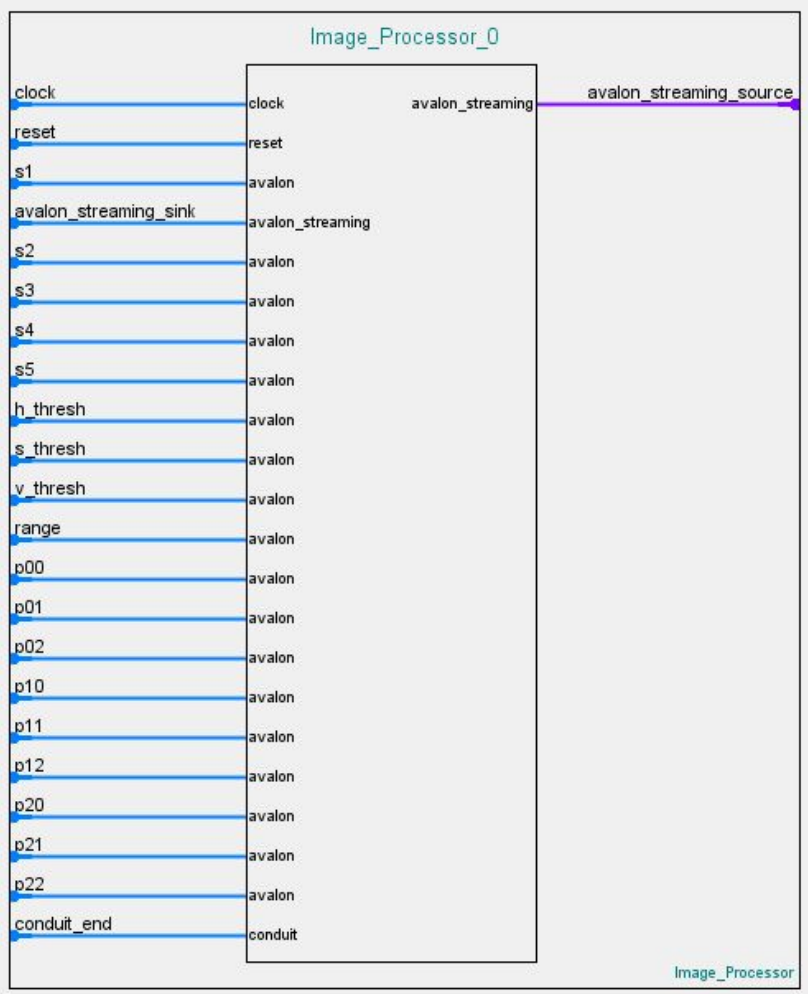
2. “Tracking coloured objects in OpenCV” [4]

Notes: This OpenCV project inspired our method for using a threshold for. We won’t be using OpenCV in this project. This project is where the idea of using HSV for the threshold came from.

DATA SHEET

Defined FPGA component:

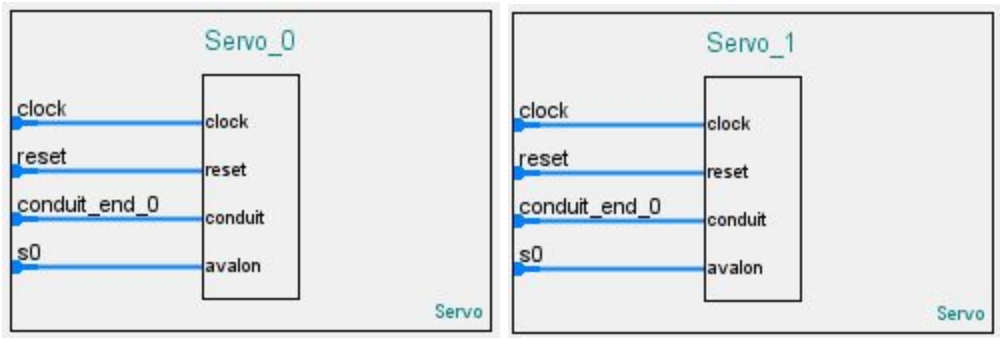
Image processor



Pin Name	Note	Connection Type
clock	System clock	Internal connection
reset	Reset signal	Internal connection
s1	Store coordinate of x and y	Internal connection
avalon_streaming_sink	Receive incoming RGB signals	Internal connection
s2 … s5	Temporary register for debugging and testing	Internal connection
h_thresh	Store H threshold value and threshold range	Internal connection
s_thresh	Store S threshold value and threshold range	Internal connection
v_thresh	Store V threshold value and threshold range	Internal connection

p_00	Store colour information on coordinate (Xcenter-5, Ycenter-5)	Internal connection
p_01	Store colour information on coordinate (Xcenter, Ycenter-5)	Internal connection
p_02	Store colour information on coordinate (Xcenter+5, Ycenter-5)	Internal connection
p_10	Store colour information on coordinate (Xcenter-5, Ycenter)	Internal connection
p_11	Store colour information on coordinate (Xcenter, Ycenter)	Internal connection
p_12	Store colour information on coordinate (Xcenter+5, Ycenter)	Internal connection
p_20	Store colour information on coordinate (Xcenter-5, Ycenter+5)	Internal connection
p_21	Store colour information on coordinate (Xcenter, Ycenter+5)	Internal connection
p_22	Store colour information on coordinate (Xcenter+5, Ycenter+5)	Internal connection
conduit_end	Check value of threshold switch	On-board connection

Servo motor controller



Pin Name	**Note**	**Connection Type**
clock	system clock	Internal connection
reset	reset signal	Internal connection
conduit_end_0	send pulse width to GPIO	On-board connection
s0	store movement instruction	Internal connection

FPGA Board to Servo Motor

FPGA Pin #	Connection	Note
PIN_D25	GPIO_0[0] => White[Signal] (Horizontal)	connect GPIO pin to signal line of the horizontal servo motor
PIN_K25	GPIO_1[0] => White[Signal] (Vertical)	connect GPIO pin to single line of the vertical servo moter
N/A	GND => Black [Vss]	Connect the ground.

Top-Level Pin Assignment

System control

FPGA Pin #	Connection
PIN_N2	clk_0 => CLOCK_50
PIN_G26	reset_n => KEY(0)

Video Out signal

FPGA Pin #	Connection
PIN_B8	vga_controller_external_interface_CLK => VGA_CLK
PIN_D6	vga_controller_external_interface_BLANK => VGA_BLANK
PIN_A7	vga_controller_external_interface_HS => VGA_HS
PIN_D8	vga_controller_external_interface_VS => VGA_VS
PIN_B7	vga_controller_external_interface_SYNC => VGA_SYNC
Multiple Pins	vga_controller_external_interface_R => VGA_R[0-9]
Multiple Pins	vga_controller_external_interface_G => VGA_G[0-9]
Multiple Pins	vga_controller_external_interface_B => VGA_B[0-9]

Video In signal

FPGA Pin #	Connection
Multiple pins	tv_decoder_controller_external_interface_DATA => TD_DATA[0-7]
PIN_D5	tv_decoder_controller_external_interface_HS => TD_HS
PIN_K9	tv_decoder_controller_external_interface_VS => TD_VS
PIN_C16	tv_decoder_controller_external_interface_CLK27 => TD_CLK27
PIN_C4	tv_decoder_controller_external_interface_RESET => TD_RESET

PIN_A6	tv_decoder_controller_external_interface_I2C_SCLK => I2C_SCLK
PIN_B6	tv_decoder_controller_external_interface_I2C_SDAT => I2C_SDAT

SDRAM controller signal

FPGA Pin #	**Connection**
Multiple pins	sdram_0_wire_addr => DRAM_ADDR[0-11]
PIN_AE2 & PIN_AE3	sdram_0_wire_ba => DRAM_BA
PIN_AB3	sdram_0_wire_cas_n => DRAM_CAS_N
PIN_AA6	sdram_0_wire_cke => DRAM_CKE
PIN_AC3	sdram_0_wire_cs_n => DRAM_CS_N
Multiple pins	sdram_0_wire_dq => DRAM_DQ[0-15]
PIN_Y2 & PIN_AD2	sdram_0_wire_dqm => DQM
PIN_AB4	sdram_0_wire_ras_n => DRAM_RAS_N
PIN_AD3	sdram_0_wire_we_n => DRAM_WE_N

SRAM controller signal

FPGA Pin #	**Connection**
Multiple pins	sram_0_external_interface_DQ => SRAM_DQ[0-15]
Multiple pins	sram_0_external_interface_ADDR => SRAM_ADDR[0-17]
PIN_AE9	sram_0_external_interface_LB_N => SRAM_LB_N
PIN_AF9	sram_0_external_interface_UB_N => SRAM_UB_N
PIN_AC11	sram_0_external_interface_CE_N => SRAM_CE_N
PIN_AD10	sram_0_external_interface_OE_N => SRAM_OE_N
PIN_AE10	sram_0_external_interface_WE_N => SRAM_WE_N

LCD signal

FPGA Pin #	**Connection**
Multiple pins	character_lcd_0_external_interface_DATA => LCD_DATA
PIN_L4	character_lcd_0_external_interface_ON => LCD_ON

PIN_K2	character_lcd_0_external_interface_BLON => LCD_BLON
PIN_K3	character_lcd_0_external_interface_EN => LCD_EN
PIN_K1	character_lcd_0_external_interface_RS => LCD_RS
PIN_K4	character_lcd_0_external_interface_RW => LCD_RW

Switch signal:

FPGA Pin #	**Connection**	**Note**
PIN_N25	config_switch_external_export => SW(0)	Set SW(0) as configuration switch
PIN_V2	h_lock_switch_external_export => SW(17)	Set SW(17) as x-lock switch
PIN_V1	v_lock_switch_external_export => SW(16)	Set SW(16) as y-lock switch
PIN_N26	image_processor_0_conduit_end_export => SW(1)	Set SW(1) as threshold image switch
PIN_P25	square_switch_external_connection_export => SW(2)	Set SW(2) as indication switch

Non-volatile signal

FPGA Pin #	**Connection**
PIN_W17	tristate_conduit_bridge_0_out_generic_tristate_controller_0_tcm_read_n_out => FL_OE_N
Multiple pins	tristate_conduit_bridge_0_out_generic_tristate_controller_0_tcm_data_out => FL_DQ[0-7]
Multiple pins	tristate_conduit_bridge_0_out_generic_tristate_controller_0_tcm_address_out => FL_ADDR[0-21]
PIN_AA17	tristate_conduit_bridge_0_out_generic_tristate_controller_0_tcm_write_n_out => FL_WE_N
PIN_V17	tristate_conduit_bridge_0_out_generic_tristate_controller_0_tcm_chipselect_n_out => FL_CE_N

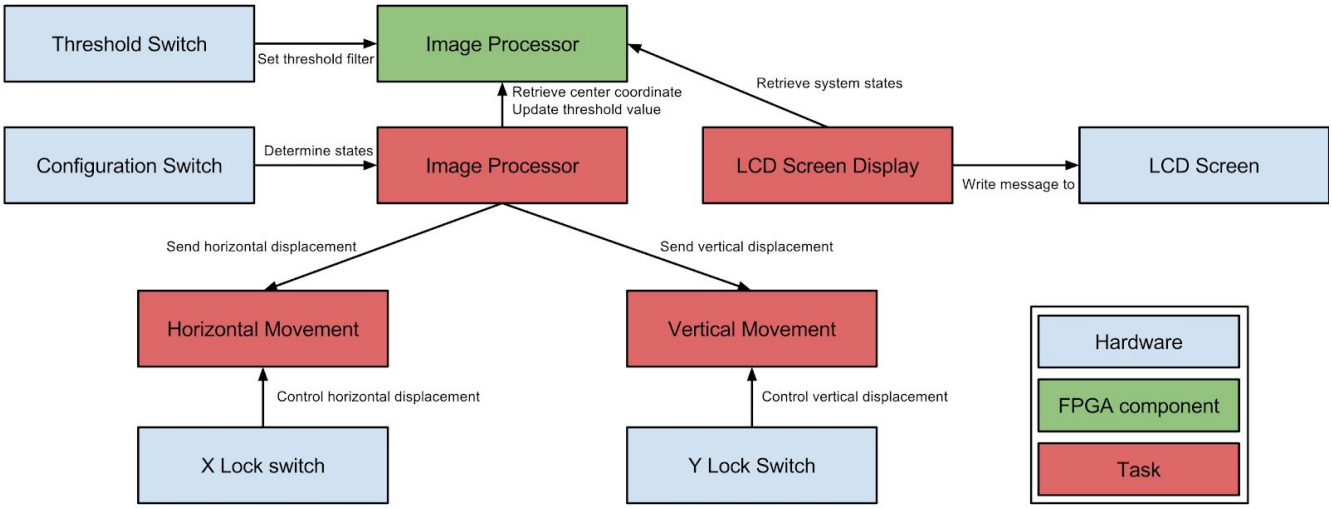
Power Supply:

High Speed Continuous Rotation Servo Motor :6V DC

Swann Security Camera: 12V DC

Altera DE2 Board: 9V AC/DC

SOFTWARE DESIGN



As per the figure shown above, we define 4 tasks in software layer.

When users flip the configure switch to set the colour they want to track, we need to write the input register of image processor through the software layer. The center pixel of the frame will be recorded directly, immediately after the value of switch goes to one, and the threshold value inside the image processor will be changed. Therefore, the incoming pixel will be filtered according to the new threshold value. To achieve this, we define a task (Image processor task) used to monitor the current state of the configuration switch to check whether the value is high (setting the color) or low (tracking the object). When the system is in the set-up state, the servos will stop working and go to its natural position. A 5x5 green square will be drawn at the center of the screen to indicate users to put the object inside the square. Initial_tracking function will be used to read the center pixel of the frame. The reading HSV value is retrieved from a register used to specifically record the center pixel. And it will also based on the one calculated by image processor FPGA component to ensure the consistency of colour information. After that, the H, S and V values of that pixel are directly written to the register used to define the threshold value. And it will keep changing the threshold value until the value of configuration switch goes low. When the system enters the tracking state, servos will resume working and locate pixels that pass the threshold comparison. Because this process is done by hardware and the register value always stays until it is updated by writing instruction, there is no extra synchronization needed additionally. The system will detect the change of threshold value every clock cycle.

We define two tasks (Horizontal Movement and Vertical Movement) to handle the horizontal movement and vertical movement independently, with a negligible delay to ensure x and y rotation are processed approximately simultaneously. X and Y lock switches can control the movement of servos. If the value of lock switches are high, then the servos of that direction will stop moving.

Also we used another task (LCD screen display) to write some information on the LCD screen. When the user wants to set colour, the LCD will display some brief instruction about how to set the colour. And when the camera tracks some object, the LCD screen indicates the current states tracking the

system - such as showing states of the X and Y direction locks.

Test plan

Software:

Our original plan is to have all image process in the software layer. That includes all unit tests of the image process functions:

Test 1: memory layout pattern of video buffer

Intention: use Altera video buffer function to figure out how pixels arranges on the screen.

alt_up_pixel_buffer_dma_draw(pixel_buf_dev, colour , x, y);

Result: it stores pixels in inside video buffer as a pattern of x-y coordinate like:

(0,0) (0,1) (0,2)
(1,0) (1,1) (1,2)
(2,0) (2,1) (2,2)

Test 2: threshold comparison unit test

Intention: we test a number of pixels with defined threshold range to check if the function is implemented properly.

Result: The function successfully filter the pixels that fail the comparison test.

Test 3: Output 2D pass-fail array unit test

Intention: processing the whole frame data and store the output result, a pass-fail value to a 2D array.

Result: The function can output an 2D array with correct size.

Test 4: Positioning unit test

Intention: this will test the functionality of algorithm used to locate the center of the object.

Result: It correctly outputs the center coordinate.

Test 5: Servo controller test

Intention: ensure the function can convert the displacement to angle and write it to servo motor controller register.

Result: The angle conversion algorithm is working properly and it can store the correct value into the register.

Test 6: Communication between switch to FPGA hardware component

Intention: When user flips switches, software layer is responsible to send the information to image processor and also give feedback to LCD screen by retrieving the current states of image processor.

Unfortunately, we failed to do the image process in the software layer (Test 1-4), the speed of locating the center of moving target is 18 seconds per each frame. This is due to the algorithm implemented for positioning. Therefore, we defined a FPGA component to process image with the same rate as it transmitting pixels to the monitor.

Hardware:

Image processor: We tested the image processor in three stages, firstly we tested if it can correctly convert form RGB colour space to HSV format. To do that, we send instructions to convert a certain RGB pixel to HSV, and then output the value to check if the values are identical to what we expect. Secondly, image processor needs to filter the pixels properly. We can test that by storing the result to a temporary register inside image processor, and use software layer to read the register to verify the correctness of the output. Lastly, image processor will locate the center of the object after the last pixel of the frame is processed. We can directly check if the algorithm is properly implemented by retrieving the coordinate, and then draw it on the monitor.

Servo motor controller: We use an oscillator to test if the Servo Motor Controller is generating the given pulse width.

SAFETY

The main external devices used in this project are two servo motors. Therefore there are no restricted or dangerous materials used.

Since this project simply rotates in place, horizontally and vertically, the only safety issue with the motion would be the swinging energy of the mount. For the horizontal rotation servo motor, the rotational speed is reported at 0.18sec/60° or 1.08 sec/360°.

Converting to rads/sec:

$$\omega = 2\pi \frac{rads}{360°} \times \frac{360°}{1.08\ sec} = 5.8178 \frac{rads}{sec}$$

Using the rotational kinetic energy equation:

$$KE_{rotational} = \frac{1}{2} I \cdot w^2$$

Where: I − *Moment of inertia*

ω − *Angular velocity*

We get:

$$I = 8.1471 \times 10^{-4} kg \cdot m^2 \quad (see\ appendix: Polar\ Moment\ of\ Inertia\ Calculations)$$

$$KE_{rotational} = \frac{1}{2} I \cdot w^2 = \frac{1}{2}(8.1471 \times 10^{-4} kg \cdot m^2) \cdot (5.8178 \frac{rads}{s})^2 = 0.013788\ J$$

Since the value for this rotation energy is quite small (magnitude of -2), the rotational energy of the mount would cause negligible harm.

ENVIRONMENTAL IMPACT

There are hazardous substances used in our project and therefore is RoHS compliant.

SUSTAINABILITY

Calculating CO_2 emissions

$$CO_{2,\ Emission} = (2.81 \frac{CO_2}{kWh}) \cdot (0.45739\ kg_{CO_2}) = 0.989 \frac{kg_{CO_2}}{kWh}$$

Calculating Servo Power Consumption

$Operating\ Voltage,\ V_{op} = 5V$
$Running\ Current,\ I_{op} = 0.26A$

$Running\ Servo\ Power,\ P_{Servo,\ running} = V_{op} \cdot I_{op} = (5V) \cdot (0.26A) = 1.3W$

*Note: It is intuitive to simply power down system when not running, therefore, the system is never idle

Calculating DE2 Power Consumption

$Operating\ Voltage,\ V_{op} = 9.1V$
$Operating\ Current,\ I_{op} = 0.555A$

$Operating\ Power,\ P_{DE2,\ op} = V_{op} \cdot I_{op} = (9.1V) \cdot (0.555A) = 5.0505W$

Calculating Camera Power Consumption

$Operating\ Voltage,\ V_{op} = 12.08V$
$Operating\ Current,\ I_{op} = 0.055A$

$Operating\ Power,\ P_{Camera,\ op} = V_{op} \cdot I_{op} = (12.08V) \cdot (0.055A) = 0.6644W$

Total Power Consumption (CO_2 Emission and Cost)

$Total\ Running\ Power,\ P_{Running} = P_{Servo,\ running} + P_{DE2,\ op} + P_{Camera,\ op} = 1.3W + 5.0505W + 0.6644W$
$P_{Running} = 7.0149W$

Assume 3 hours of continuous, recreational use a day, everyday:

$CO_2\ Emissions = 7.0149W \times \frac{1kW}{1000W} \times 3\frac{h}{day} \times 365.25\frac{days}{year} \times 0.989\frac{kg_{CO_2}}{kWh} = 7.602\frac{kg}{year}\ of\ CO_2\ emission$
$Energy\ Cost = 7.0149W \times \frac{1kW}{1000W} \times 3\frac{h}{day} \times 365.25\frac{days}{year} \times 8.7\frac{¢}{kWh} = \$0.67/year$

The CO_2 emissions for this project is estimated at 7.602 kg/year and the cost to run is $0.67/year.

REFERENCES

1). Altera Corporation. Video and Image Processing Suite User Guide. Data Accessed: Feb 1, 2015
http://www.altera.com/literature/ug/ug_vip.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=video%20ip

2).JunZhao. Tutorial of displaying pixels on VGA monitor, Data Accessed: Feb 1, 2015.
https://www.ualberta.ca/~delliott/local/ece492/appnotes/2014w/G3_VGA/G3_VGA_AppNote_V2.pdf

3). Tatsuya Hamachi , Hiroyuki Tanabe, Akira Yamawaki. Development of a Generic RGB to HSV Hardware. Data Accessed: Jan 26,2015
https://www2.ia-engineers.org/iciae/index.php/iciae/iciae2013/paper/viewFile/8/103

4). Utkarsh Sinha. Tracking coloured objects in OpenCV. Data Accessed: Jan 26, 2015
http://www.aishack.in/tutorials/tracking-coloured-objects-in-opencv/

5). Swann. ADS-120, Data Accessed: Feb 23, 2015
http://www.swann.com/downloads/product/2017ADS-120_M120CAM041012E_web.pdf

6). Hitec Inc. HS-422 Servo Motor. Data Accessed: Mar 2, 2015.
http://www.robotshop.com/ca/en/hitec-hs422-servo-motor.html

7).Hitec Inc. HS-635HB Servo Motor. Data Accessed: April 13, 2015.
http://hitecrcd.com/products/servos/sport-servos/analog-sport-servos/hs-635hb-karbonite-high-torque-servo/product

8).Altera Corporation. Altera DE2 board. Data Accessed: Mar 25, 2015
https://www.altera.com/support/training/university/de2.html

9).Control Engineering. How to size servo motors: Advanced Inertia Calculation. Data Accessed: April 14, 2015
http://www.controleng.com/single-article/how-to-size-servo-motors-advanced-inertia-calculations/477dd9f0979e8d1cf698d752c685fb85.html

QUICK START MANUAL

Hardware Configuration

1. Download the G4_Coloured_Object_Tracking_Camera.qar file from the web page.
2. Open Quartus and select **Project** > **Restore Archived Project**, and select the G4_Coloured_Object_Tracking_Camera.qar in your download directory and click **OK**.
3. Double Click **Compile Design** to run the program
4. When it completes compile, double click **Program Device**
5. Set up the connection in hardware:
    a. connect power adapter from Altera DE2 Board to power source
    b. connect USB cable from Alter DE2 Board to Computer
    c. connect composite video cable to Video-In port on DE2 Board
    d. connect VGA cable from a monitor to VGA-port on DE2 Board.
    e. connect horizontal servo signal line to GPIO_0[0] pin.
    f. connect vertical servo signal lien to GPIO_1[0] pin.
    g. connect power adapter to from camera to power source
    h. connect power supply to both of servos and change the voltage to 5V.
6. Click **Auto Detect** on the left side bar.
7. Select niosII_micror_lab_1.sof and click **Start**.

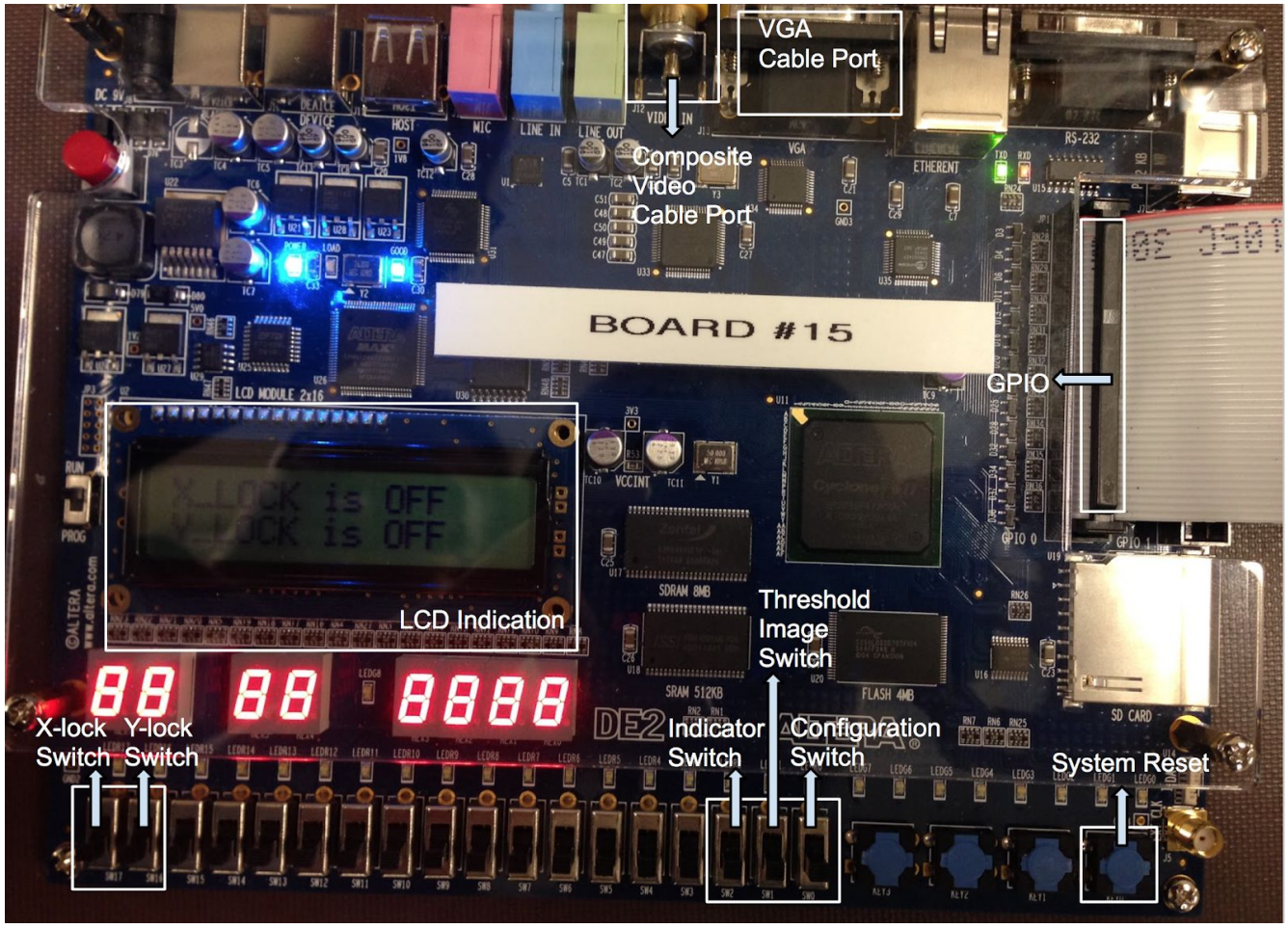
Now, the monitor should be able to show the image from the camera.

Software configuration

1. Open Eclipse from the Quartus by selecting **Tools** -> **Nios II software Build Tools for Eclipse**
2. Choose your Workspace directory to the software file inside the project.
3. Click **File** -> **New** -> **Nios II Application and BSP from Template**
4. Choose the **video_sys.sopcinfo** in SOPC Information File Name field.
5. Name your project
6. Select Templates as **Hello MicroC/OS-II**
7. Click **Finish**
8. Right Click your project in Project Explorer window, and select **Build Project**
9. Delete hello_ucosii.c file from the project.
10. Copy image_processor_software.c into the project.
11. Right click on project and select **Run As -> Nios II Hardware**

User Guide

Configuration Switch: used to change the state of setting color or tracking color
Threshold Image Switch: used to show pass fail pixels on the screen, all the passed pixels will be shown as white pixels and others will be shown as black.
Indicator Switch: used to show the center coordinate of the object on the screen by a green square while tracking.
X-Lock Switch: used to stop horizontal rotation of camera
Y-Lock Switch: used to stop vertical rotation of camera



How to set the colour:

1. Flip configuration switch and place the object we wish to track at the green square showing on the screen.
2. (Optional) Flip Threshold Image Switch to see if the object is marked as white pixel on the screen.
3. Flip configuration switch again and the camera should start to track the object.
4. For changing colour, re-do the procedure 1-3.

FUTURE WORK

Custom Settings

Focusing on artistic side of videography, implementing multiple settings for tracking style would improve filming variety as well as increase the demographic of targeted users. Examples of this would being able to change the rotational speeds of the servos, improving smoothness, and perhaps viewing experience. Another custom setting would be to preset tracking limits, making the target leave the camera's view at predefined angles which can add an artistic approach to filming.

Minimize Form Factor

In our camera's current form factor, the desired intention to be utilized for action sports, is highly improbable. The next step to this design would be to scale down the size. Multiple things can be implemented to realize this objective. One would be custom circuit boards, with a more optimized chip layout (instead of using the development board, design a board with only the components needed). Also scale down the size of the physical mount, making the cradle and cradle support much smaller and more compact. Consolidating the power source for all the components will also decrease the size. Since video output is necessary for colour configuration, either adding a small display to the mount itself or have the video streamed from the mount to your phone using local wifi of bluetooth.

Modernize

A hindrance for future development and success for this project is that we implemented our design using NTSC composite video. The issue with this is that NTSC is an older technology and most cameras don't use this technology. For our intents and purposes, we wish to have the source of the video to be modular, as in, the goal is compatibility with any camera the user wishes. By modernizing this, either with USB or HDMI video in, we can use any modern camera, such as GoPros or DSLRs.

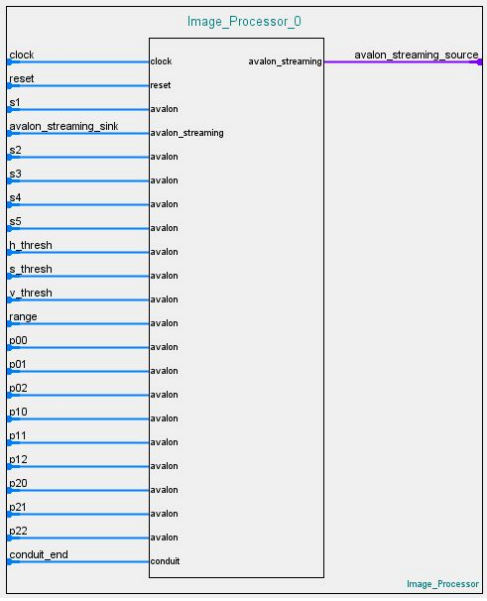
Improve Tracking Algorithms

Since we were under time constraint for the development of this project, we decided to use simple algorithms for calculating the centre of the object and finding our threshold ranges. To improve our algorithm for calculating the centre of the object, instead of using the simple method of averaging the coordinates of the first and last pixel, we would like to keep track of the longest continuous line of horizontal and vertical pixels that pass. Also, filtering out noise by ignoring lone pixels or lone anomalies. Implementing algorithms to detect colour patterns and instead of having simply scalar ranges for the threshold values (like ± 5 for hue), we would design algorithms that dynamically shift ranges according to white balance shifts from the camera, lighting changes, and camera quality.

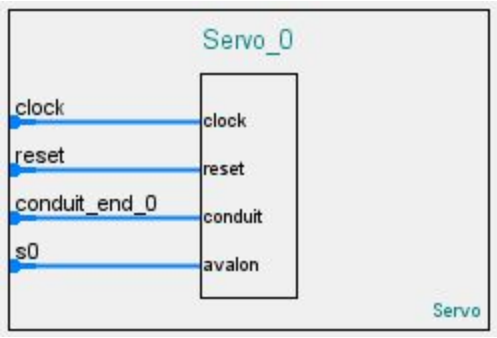
HARDWARE DOCUMENTATION

Defined FPGA component:

Image Processor : The main functionalities of the image processor include converting RGB to HSV, filter the pixels that pass the threshold comparison test and find the center coordinate of the tracking object and stored it into a register.



Servo Motor Controller: Used to generate given pulse width modulation and send it to GPIO pins



SOFTWARE DOCUMENTATION:

main.c

```
/*************************************************************************
 * Description:                                                          *
 * This file shows the software layer of our design, The functionalities*
 * of this file includes calculating the pulse width of servos, controlling
 * movement of the servo and communicate between image processor and user *
 * interface                                                             *
 *************************************************************************/
#include <stdio.h>
#include <system.h>
#include <stdlib.h>
#include <time.h>
#include "includes.h"
#include "altera_up_avalon_video_pixel_buffer_dma.h"
#include "io.h"
#include "altera_up_avalon_character_lcd.h"

/* Definition of Colour */
#define   GREEN 0xF800

/* Definition of Image processor  Parameter */
#define H_MASK 0x00000007FFF
#define S_MASK 0x000007F8000
#define V_MASK 0x0007F800000

#define H_OFFSET 0
#define S_OFFSET 15
#define V_OFFSET 23

#define DEFAULT_H_RANGE 10
#define DEFAULT_S_RANGE 20
#define DEFAULT_V_RANGE 80

#define HSV_RANGE_REGISTER_OFFSET 24

#define H_REGISTER_MASK 0x000001ff
#define S_REGISTER_MASK 0x000000ff
#define V_REGISTER_MASK 0x000000ff

#define X_MASK 0x3ff
#define Y_MASK 0xFfC00
#define X_OFFSET 0
#define Y_OFFSET 10

/* Definition of Screen  */
#define   CENTER_X 160
#define   CENTER_Y 120
#define   MARGIN   5


/* Definition of Servo Moter Parameter*/
#define   MAX_PULSE_WIDTH 125000
#define   MIN_PULSE_WIDTH 75000
#define   PULSE_WIDTH_PER_DEGREE 425
#define   PIXEL_PER_DEGREE 26.5


/* Definition of Task Stacks */
#define   TASK_STACKSIZE      2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
OS_STK    task3_stk[TASK_STACKSIZE];
OS_STK    task4_stk[TASK_STACKSIZE];

/* Define Global Variable*/
```

```
alt_up_pixel_buffer_dma_dev * pixel_buf_dev;
alt_up_character_lcd_dev * character_lcd_dev;

float degree_x,degree_y;
int z = 20;
int flip =0;

/* Definition of Task Priorities */
#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2
#define TASK3_PRIORITY      3
#define TASK4_PRIORITY      4

/*
 * Servo control function for the HS-422
 * Input: The number of degrees of horizontal rotation needed
 * Output:
 *                      1  : If the servo angle is at max rotation (? ms pulse width)
 *                      -1 : If the servo angle is at min rotation (? ms pulse width)
 *                      0  : Servo doesn't reach limits
 */
int horizontal_servo(float number_of_degrees){
        // (-) left, (+) right
        // +- 950 to move 1 degree
        // Max is ~ 250000, min is ~ 60000
        int pulse_width = 0,err = 0;
        int old = IORD_32DIRECT(SERVO_0_BASE, 0);
        pulse_width = old + number_of_degrees*PULSE_WIDTH_PER_DEGREE;

        if (pulse_width > MAX_PULSE_WIDTH){
                pulse_width = MAX_PULSE_WIDTH;
                err = 1;
        }
        else if(pulse_width < MIN_PULSE_WIDTH){
                pulse_width = MIN_PULSE_WIDTH;
                err = -1;
        }

                old = IORD_32DIRECT(SERVO_0_BASE, 0);
                IOWR_32DIRECT(SERVO_0_BASE, 0, pulse_width);
                old = IORD_32DIRECT(SERVO_0_BASE, 0);
                return err;
}

/*
 * Servo control function for the HS-635HB
 * Input: The number of degrees of vertical rotation needed
 * Output:
 *                      1  : If the servo angle is at max rotation (? ms pulse width)
 *                      -1 : If the servo angle is at min rotation (? ms pulse width)
 *                      0  : Servo doesn't reach limits
 */
int vertical_servo(float number_of_degrees){
        // (-) up, (+) down
        // +- 950 to move 1 degree
        // Max is ~ 125000, min is ~ 25000
        int pulse_width = 0, err = 0, old = 0;
        old = IORD_32DIRECT(SERVO_1_BASE, 0);
        pulse_width = old - number_of_degrees*PULSE_WIDTH_PER_DEGREE;
        if (pulse_width > MAX_PULSE_WIDTH){
                pulse_width = MAX_PULSE_WIDTH;
                err = 1;
        }
        else if(pulse_width < MIN_PULSE_WIDTH){
                pulse_width = MIN_PULSE_WIDTH;
                err = -1;
        }
        old = IORD_32DIRECT(SERVO_1_BASE, 0);
        IOWR_32DIRECT(SERVO_1_BASE, 0, pulse_width);
```

```
		old = IORD_32DIRECT(SERVO_1_BASE, 0);
		return err;
}

/*
 * Initial_tracking function will be used to replace the value of the register which stores
 * threshold value in the image processor by the value read from the center of the frame.
 */
void inital_tracking(){
		int h_11, v_11, s_11, center_pixel,
	h_reg, s_reg, v_reg;

   // Read the center pixel from the image processor
   center_pixel = IORD_32DIRECT(IMAGE_PROCESSOR_0_P11_BASE,0);

   // Get H, S, V value of the center pxiel
		v_11 = (hsv_11&V_MASK)>>V_OFFSET;
		s_11 = (hsv_11&S_MASK)>>S_OFFSET;
		h_11 = (hsv_11&H_MASK)>>H_OFFSET;

   // Generate a 32bit number which contains the threshould range and threshold value of each colour fields
		h_reg = ((DEFAULT_H_RANGE<<HSV_RANGE_REGISTER_OFFSET) | (h_11&H_REGISTER_MASK));
		s_reg = ((DEFAULT_S_RANGE<<HSV_RANGE_REGISTER_OFFSET) | (s_11&S_REGISTER_MASK));
		v_reg = ((DEFAULT_V_RANGE<<HSV_RANGE_REGISTER_OFFSET) | (v_11&V_REGISTER_MASK));

   // Updating the value to the register.
		IOWR_32DIRECT(IMAGE_PROCESSOR_0_H_THRESH_BASE,0,h_reg);
		IOWR_32DIRECT(IMAGE_PROCESSOR_0_S_THRESH_BASE,0,s_reg);
		IOWR_32DIRECT(IMAGE_PROCESSOR_0_V_THRESH_BASE,0,v_reg);


   //Draw a green suqare at the center of the frame for indication
		alt_up_pixel_buffer_dma_draw_rectangle(pixel_buf_dev, CENTER_X-MARGIN, CENTER_Y-MARGIN,
CENTER_X+MARGIN,CENTER_Y+MARGIN, GREEN, 0);

}

/*
 * This function is used to clear the LCD screen by erase the content of all the cursor postions
 */

void clear_lcd(alt_up_character_lcd_dev *lcd){
		int x = 0;
		int y = 0;
		alt_up_character_lcd_set_cursor_pos(lcd,0,0);
		for(y = 0; y < 2; y++){
			for(x = 0; x < 16; x++){
				alt_up_character_lcd_erase_pos(lcd,x,y);
			}
		}
		alt_up_character_lcd_set_cursor_pos(lcd,0,0);
}


/*
 * This function is used to write a string onto the LCD screen
 */
void write_lcd_msg(alt_up_character_lcd_dev *lcd, char* msg1, char* msg2){
		clear_lcd(lcd);
		alt_up_character_lcd_set_cursor_pos(lcd,0,0);
		alt_up_character_lcd_string(lcd, msg1);
		alt_up_character_lcd_set_cursor_pos(lcd,0,1);
		alt_up_character_lcd_string(lcd, msg2);
}

/*Clears the lcd*/
void servo_init(){
		IORD_32DIRECT(SERVO_1_BASE, 0);
		IOWR_32DIRECT(SERVO_1_BASE, 0, 100000);
```

```
		IORD_32DIRECT(SERVO_1_BASE, 0);
		IORD_32DIRECT(SERVO_0_BASE, 0);
		IOWR_32DIRECT(SERVO_0_BASE, 0, 75000);
		IORD_32DIRECT(SERVO_0_BASE, 0);

}

/*
 *This task is used to showing the message about the current states of the tracking system when
 *the configuration switch goes low.
 */
void task4(void* pdata){

		while(1){

			char* msg;
			char* msg2;
			if (!IORD_8DIRECT(CONFIG_SWITCH_BASE,0)){
				if (IORD_8DIRECT(H_LOCK_SWITCH_BASE,0)){
					msg = "X_LOCK is ON";
				}
				else{
					msg = "X_LOCK is OFF";
				}

				if (IORD_8DIRECT(V_LOCK_SWITCH_BASE,0)){
					msg2 = "Y_LOCK is ON";
				}
				else{
					msg2 = "Y_LOCK is OFF";
				}
				write_lcd_msg(character_lcd_dev,msg,msg2);
				OSTimeDlyHMSM(0, 0, 0, 250);
			}

		}
}

/*
 * This task is used to control the horizontal servo movement.
 */
void task2(void* pdata)
{
		while(1){
			if(!IORD_8DIRECT(H_LOCK_SWITCH_BASE,0) && !IORD_8DIRECT(CONFIG_SWITCH_BASE,0)){
				horizontal_servo(degree_x);

			}
			OSTimeDlyHMSM(0, 0, 0, z);
		}
}

/*
 * This task is used to control the vertical servo movement.
 */
void task3(void* pdata)
{
		while(1){
			if(!IORD_8DIRECT(V_LOCK_SWITCH_BASE,0)  && !IORD_8DIRECT(CONFIG_SWITCH_BASE,0)){
				vertical_servo(degree_y);
			}
			OSTimeDlyHMSM(0, 0, 0, z);
		}
}

/*
 * This task is used to retrieve and process the center of object in order to send movement instruction
 * to servo for tracking. It also handles two different states of the tracking system, setting the initial colour and tracking the colour.
 */
```

```
void task1(void* pdata)
{
	int r = 0, g = 0, b = 0;
	int xy = 0,h = 0, s = 0,v = 0,x = 0,y = 0, thr= 0;
	int counter = 0,row=0,row_index = 0;
	int temp1,temp2,temp3,temp4;
	int max,min;
	int xmin,xmax,ymin,ymax,relative_x,relative_y;
	character_lcd_dev = alt_up_character_lcd_open_dev("/dev/character_lcd_0");
	// open pixel buffer device
	pixel_buf_dev = alt_up_pixel_buffer_dma_open_dev("/dev/Pixel_Buffer_DMA");
		if (pixel_buf_dev == NULL)
			{
				// if pixel buffer cannot be found, type error message.
				printf("Error: could not open pixel buffer device \n");
			} else
			{
				printf("Opened pixel buffer device \n");
			}
				// Make sure the pixel buffer device has the primary buffer address
			if (pixel_buf_dev->buffer_start_address != 0)
			{
				alt_up_pixel_buffer_dma_change_back_buffer_address(pixel_buf_dev, 0);
				alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);
			}
			// swap buffer to the back buffer (start address of SDRAM for as frame buffer)
			alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);

	// Showing welcome message when the program is initialized properly
			alt_up_character_lcd_init(character_lcd_dev);
			char* welcome_string = "Hello, welcome!";
			alt_up_character_lcd_string(character_lcd_dev,welcome_string);
			alt_up_character_lcd_cursor_off(character_lcd_dev);
			OSTimeDlyHMSM(0, 0, 1, 0);


	  while(1){

				if(IORD_8DIRECT(CONFIG_SWITCH_BASE,0)){
					flip = 1;
					servo_init();
	    // showing a brief instruction about setting colour
					char* msg = "Place object";
					char* msg2 ="in green square";
					write_lcd_msg(character_lcd_dev, msg, msg2);
	    // it will keep updating the threshold value in the register until the value
	    // of configuration switch goes low.

					while(IORD_8DIRECT(CONFIG_SWITCH_BASE,0)){
						inital_tracking();
					}
	    // servo process will be suspended in setting state.
	    servo_init();

				}

	  // If the state is tracking, then it will retrieve the center coordinate from register
	  // in the image processor
				xy = IORD_32DIRECT(IMAGE_PROCESSOR_0_S1_BASE,0);
				x = (xy&X_MASK)>>X_OFFSET;
				y = (xy&Y_MASK)>>Y_OFFSET;


				relative_x = x - CENTER_X;
				relative_y = y - CENTER_Y;

	  // IF the movement is not siginificant, then the servo can ignore it
				if ((relative_x < 10) && (relative_x > -10)){
					relative_x = 0;
```

```
                                        }
                                        if ((relative_y < 10) && (relative_y > -10)){
                                                relative_y = 0;
                                        }

                                        degree_x = relative_x/PIXEL_PER_DEGREE;
                                        degree_y = relative_y/PIXEL_PER_DEGREE;



                        OSTimeDlyHMSM(0, 0, 0, 5);
                        // 8 fps
                }
}


/* The main function creates two task and starts multi-tasking */
int main(void)
{
 servo_init();
 alt_up_character_lcd_init(character_lcd_dev);
 OSTaskCreateExt(task1,
                NULL,
                (void *)&task1_stk[TASK_STACKSIZE-1],
                TASK1_PRIORITY,
                TASK1_PRIORITY,
                task1_stk,
                TASK_STACKSIZE,
                NULL,
                0);

 OSTaskCreateExt(task2,
                 NULL,
                 (void *)&task2_stk[TASK_STACKSIZE-1],
                 TASK2_PRIORITY,
                 TASK2_PRIORITY,
                 task2_stk,
                 TASK_STACKSIZE,
                 NULL,
                 0);

 OSTaskCreateExt(task3,
                  NULL,
                  (void *)&task3_stk[TASK_STACKSIZE-1],
                  TASK3_PRIORITY,
                  TASK3_PRIORITY,
                  task3_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);
 OSTaskCreateExt(task4,
                   NULL,
                   (void *)&task4_stk[TASK_STACKSIZE-1],
                   TASK4_PRIORITY,
                   TASK4_PRIORITY,
                   task4_stk,
                   TASK_STACKSIZE,
                   NULL,
                   0);


 OSStart();
 return 0;
}
```

MOUNT DESIGN

Figure 1: CAMERA MOUNT

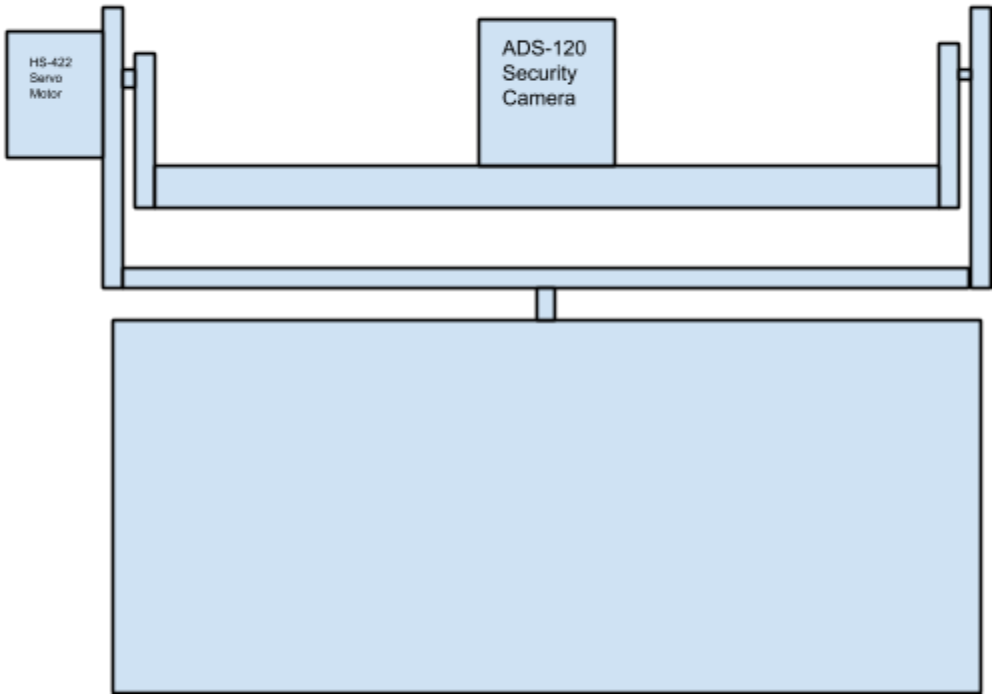


Figure 2: CRADLE

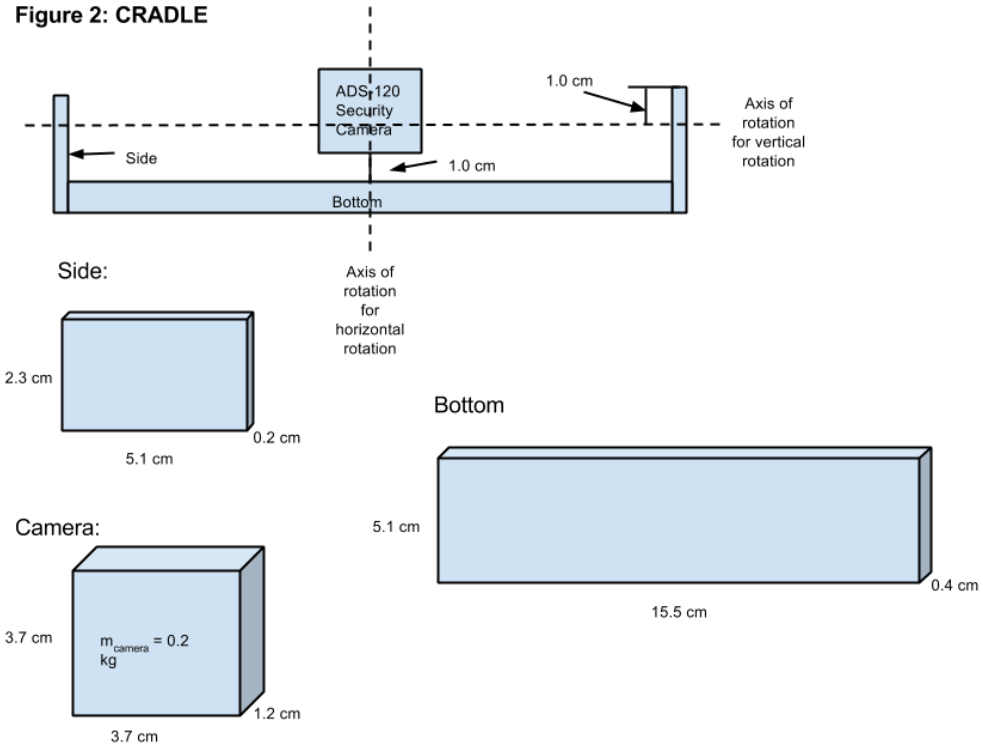
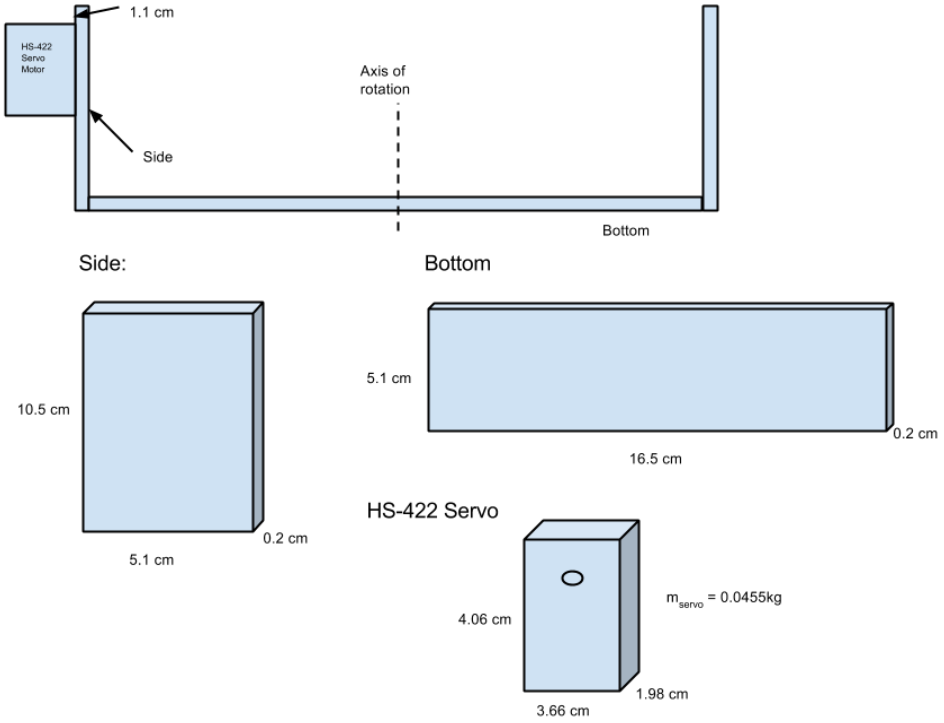


Figure 3: CRADLE SUPPORT



POLAR MOMENT OF INERTIA CALCULATIONS FOR CAMERA MOUNT

Polar moment of inertia:

$\tau = J \cdot \alpha$

where: τ – *Torque* $[N \cdot m]$

J – *Polar Moment of Inertia* $[kg \cdot m^2]$

α – *Angular Acceleration* $[\frac{rad}{s^2}]$

Parallel axis theorem:

$J = J' + m \cdot d^2$

where: J – *Polar Moment of Inertia about axis of rotation* $[kg \cdot m^2]$

J' – *Polar Moment of Inertia about centre axis* $[kg \cdot m^2]$

m – *Mass* $[kg]$

d – *Distance between centre axis and axis of rotation* $[m]$

c - cradle

cs - cradle support

Here we assume material is aluminium $(\rho = 2660\frac{kg}{m^3})$.

<u>Part 1: Polar Moment of Inertia for Vertical Rotation:</u> (Figure 2)

Side:

$m_{c,\,side} = V \cdot \rho = (0.051m) \cdot (0.023m) \cdot (0.002m) \cdot (2660\frac{kg}{m^3}) = 6.24 \times 10^{-3} kg$

$J_{side} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(6.24 \times 10^{-3} kg)[(0.023m)^2 + (0.051m)^2] = 1.6277 \times 10^{-6} kg \cdot m^2$

Parallel axis theorem:

$d = |\frac{1}{2}(0.023m) - (0.01m)| = 0.0015m$

$J_{c,\,side} = J_{side} + m \cdot d^2 = (1.6277 \times 10^{-6} kg \cdot m^2) + (6.24 \times 10^{-3} kg)(0.0015m)^2$

$J_{c,\,side} = 1.6417 \times 10^{-6} kg \cdot m^2$

Bottom:

$m_{c,\,bottom} = V \cdot \rho = (0.155m) \cdot (0.051m) \cdot (0.004m) \cdot (2660 \frac{kg}{m^3}) = 8.41 \times 10^{-2} kg$

$J_{bottom} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(8.41 \times 10^{-2} kg)[(0.004m)^2 + (0.051m)^2] = 1.8343 \times 10^{-5} kg \cdot m^2$

Parallel axis theorem:

$d = |0.023m - 0.01m - 0.002m| = 0.011m$

$J_{c,\,bottom} = J_{bottom} + m \cdot d^2 = (1.8343 \times 10^{-5} kg \cdot m^2) + (8.41 \times 10^{-2} kg)(0.011m)^2$

$J_{c,\,bottom} = 2.852 \times 10^{-5} kg \cdot m^2$

Camera:

$m_{camera} = 0.2kg$

$J_{camera} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(0.2kg)[(0.012m)^2 + (0.037m)^2] = 2.5217 \times 10^{-5} kg \cdot m^2$

Parallel axis theorem:

$d = |0.013m - 0.01m - \frac{0.037m}{2}| = 0.0155m$

$J_{c,\,camera} = J_{camera} + m \cdot d^2 = (2.5217 \times 10^{-5} kg \cdot m^2) + (0.2kg)(0.0155m)^2$

$J_{c,\,camera} = 7.3267 \times 10^{-5} kg \cdot m^2$

Total:

$J_{vert,\,total} = 2 \cdot J_{c,\,side} + J_{c,\,bottom} + J_{c,\,camera}$

$= 2 \cdot (1.6417 \times 10^{-6} kg \cdot m^2) + (2.852 \times 10^{-5} kg \cdot m^2) + (7.3267 \times 10^{-5} kg \cdot m^2)$

$J_{vert,\,total} = 1.0507 \times 10^{-4} kg \cdot m^2$

<u>Calculating Angular Acceleration</u>

For Vertical rotation, $\tau = 3.3kg \cdot cm \times 9.81 \frac{m}{s^2} \times 1 \frac{m}{100\,cm} = 0.32373\ N \cdot m$

(torque @ 4.8V according to data sheet)

$J_{vert,\,total} = 1.0507 \times 10^{-4} kg \cdot m^2$

$\alpha = \frac{\tau}{J} = \frac{0.32373\ N \cdot m}{1.0507 \times 10^{-4} kg \cdot m^2} = 3.0811 \times 10^3 \frac{rad}{s^2}$

Since our angular acceleration is very large, the cradle of the mount should move instantly with no lag caused by a lack of torque, i.e. the torque provided by the servo for the vertical rotation is sufficient for this.

<u>Part 2: Polar Moment of Inertia for Horizontal Rotation:</u>

Cradle: (Figure 2)

Side:

$m_{c,\,side} = 6.24 \times 10^{-3} kg$ (*from above*)

$J_{side} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(6.24 \times 10^{-3} kg)[(0.002m)^2 + (0.051m)^2] = 1.3546 \times 10^{-6} kg \cdot m^2$

Parallel axis theorem:

$d = |\frac{1}{2}(0.155m) - (0.001m)| = 0.0765m$

$J_{c,\,side} = J_{side} + m \cdot d^2 = (1.3546 \times 10^{-6} kg \cdot m^2) + (6.24 \times 10^{-3} kg)(0.0765m)^2$

$J_{c,\,side} = 3.7873 \times 10^{-5} kg \cdot m^2$

Bottom:

$m_{c,\,bottom} = 8.41 \times 10^{-2} kg$ (*from above*)

$J_{c,\,bottom} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(8.41 \times 10^{-2} kg)[(0.155m)^2 + (0.051m)^2] = 1.8660 \times 10^{-4} kg \cdot m^2$

Camera:

$m_{camera} = 0.2kg$

$J_{camera} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(0.2kg)[(0.012m)^2 + (0.037m)^2] = 2.5217 \times 10^{-5} kg \cdot m^2$

Cradle Support: (Figure 3)

Side:

$m_{cs,\,side} = V \cdot \rho = (0.105m) \cdot (0.051m) \cdot (0.002m) \cdot (2660 \frac{kg}{m^3}) = 2.8489 \times 10^{-2} kg$

$J_{side} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(2.8489 \times 10^{-2} kg)[(0.002m)^2 + (0.051m)^2] = 6.1844 \times 10^{-6} kg \cdot m^2$

Parallel axis theorem:

$d = |\frac{1}{2}(0.165m) - (0.001m)| = 0.0815m$

$J_{cs,\,side} = J_{side} + m \cdot d^2 = (6.1844 \times 10^{-6} kg \cdot m^2) + (2.8489 \times 10^{-2} kg)(0.0815m)^2$

$J_{cs,\,side} = 1.9542 \times 10^{-4} kg \cdot m^2$

Bottom:

$m_{cs,\,bottom} = V \cdot \rho = (0.165m) \cdot (0.051m) \cdot (0.002m) \cdot (2660 \frac{kg}{m^3}) = 4.4768 \times 10^{-2} kg$

$J_{cs,\,bottom} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(4.4768 \times 10^{-2} kg)[(0.165m)^2 + (0.051m)^2] = 1.1127 \times 10^{-4} kg \cdot m^2$

HS-422 Servo:

$m_{servo} = 0.0455\ kg$

$J_{servo} = \frac{1}{12} m(h^2 + w^2) = \frac{1}{12}(0.0455kg)[(0.0366m)^2 + (0.0198m)^2] = 6.5657 \times 10^{-6} kg \cdot m^2$

Parallel axis theorem:

$d = |\frac{1}{2}(0.0165m) + (0.002m) + \frac{1}{2}(0.0198m)| = 0.02015m$

$J_{c,\,servo} = J_{servo} + m \cdot d^2 = (6.5657 \times 10^{-6} kg \cdot m^2) + (0.0455kg)(0.02015m)^2$

$J_{c,\,servo} = 2.5040 \times 10^{-5} kg \cdot m^2$

Total:

$J_{horz,\,total} = 2 \cdot J_{c,\,side} + J_{c,\,bottom} + J_{c,\,camera} + 2 \cdot J_{cs,\,side} + J_{cs,\,bottom} + J_{c,\,servo}$

$= 2 \cdot (3.7873 \times 10^{-5} kg \cdot m^2) + (1.8660 \times 10^{-4} kg \cdot m^2) + (2.5217 \times 10^{-5} kg \cdot m^2)$

$+ 2 \cdot (1.9542 \times 10^{-4} kg \cdot m^2) + (1.1127 \times 10^{-4} kg \cdot m^2) + (2.5040 \times 10^{-5} kg \cdot m^2)$

$J_{horz,\,total} = 8.1471 \times 10^{-4} kg \cdot m^2$

<u>Calculating Angular Acceleration</u>

For Horizontal rotation, $\tau = 5kg \cdot cm \times 9.81 \frac{m}{s^2} \times 1 \frac{m}{100\ cm} = 0.4905\ N \cdot m$
(torque @ 4.8V according to data sheet)
$J_{horz,\ total} = 8.1471 \times 10^{-4} kg \cdot m^2$
$\alpha = \frac{\tau}{J} = \frac{0.4905\ N \cdot m}{8.1471 \times 10^{-4} kg \cdot m^2} = 6.021 \times 10^2 \frac{rad}{s^2}$

Since our angular acceleration is very large, the entire mount should move instantly horizontally with no lag caused by a lack of torque, i.e. the torque provided by the servo for the horizontal rotation is sufficient for this.