ECE 492

Ball Tracking and Trajectory Prediction

Final Report

Group Members (Group G13):
Shenhao Li -- shenhao@ualberta.ca
Paolo Villadarez -- villadar@ualberta.ca
Preferred lab: Thursday

Embedded video processing system which tracks ball movements, predicts trajectory paths, and outputs simulations via VGA.

Abstract

This report documents the development of a video processing system, which aims to track spherical objects and predict movement trajectories. This project was created on an Altera DE2 development FPGA board, using a combination of custom written VHDL hardware tracking components, Quartus SOPC generated FPGA configuration files, and software written for the uC/OSII RTOS. Our project features an orthogonal overhead and side-facing double-camera system with a VGA user interface that operates on button interrupts in two completed modes, tracking mode and prediction mode. In tracking mode, our system is capable of detecting the centroid of spherical objects in real time (using both cameras), with simulated visualizations outputted to VGA. In prediction mode, pre-calibrated parameters allow the user to roll a ball through the camera view, and have the system predict the ball's 2D impact point with a preset target using image processing and software physical calculations. Again, the outputs are simulated via VGA. We implemented DE2 switches to modify the most significant bits of red, green and blue values in the RGB color space to allow run-time input for user desired tracking color. We were successful in meeting our proposed functional requirements, with the exception of utilizing our side-facing camera for triangulation and 3D impact prediction.

Table of Contents

Functional Requirements	4
Design and Description of Operation	5
Bill of Materials	9
Reusable Design Units	10
Datasheet	11
Background Reading	13
Software Design	15
Test Plan	18
Test Results	20
References	23
Appendix	24

Functional Requirements

Ultimately, we are building an orthogonal double camera system to track and predict trajectories of spherical objects passing through its field of view. Aside from calibrating overhead and side-facing cameras for optimal viewing area and lighting, we proposed two core components as functional requirements of the project.

Ball Tracking:
We require that our system be capable of successfully capturing and processing a video stream to track the centroid of a spherical object whose color is defined by the user, and to be capable of performing correct software calculations to predict the impact location of a thrown projectile. The whole process flow of the system must be performed within a specific time interval bounded by the time the first sample of the projectile's position is taken and the instant the projectile collides with the preset target. To achieve the rapid execution of the path prediction calculations, the system must perform the video processing and object tracking function without the use of any buffers. Doing so would cut the buffer read/write time from the total processing time, thereby reducing the time delay for the output of an object's coordinates. The time interval between coordinate sampling times must also minimized to further reduce the overall delay. Two Altera DE2 boards are used in order to utilize their individual ADV7181 video-in decoders for each camera, as each decoder only allows one input. Detailed description of our tracker design and tracking algorithm can be found in the Design and Hardware sections.

VGA:
Furthermore, we require that a functional VGA controller be implemented to regulate the creation, buffering and syncing of VGA video frames to complement the project functionality via visual display of tracking simulations and user interface. Specifically, a dartboard simulation is projected to emulate the preset target. Impact animations should be drawn as projectile balls pass through the active viewing areas. A simple UI must be implemented to aid users in navigating through project functional modes. Video frames must also be displayed in a fashion which minimizes visual lag and meets timing requirements described in the ball tracking section above. Detailed description of our VGA controller and frame creation process can be found in the Design and Software sections.

We have achieved all of the functional requirements set out during our project proposal to a satisfactory degree, with the exception of utilizing the second side-facing camera to correctly calculate the height of a projectile's impact point. Although we have completed the integration of a second camera and synchronized its data accordingly, we were unable to resolve bugs in y-axis calculations. We attribute this to errors in conversion of required parameters needed for the projectile motion equation in question, and perhaps to failure of the system to meet timing requirements for accurate physics.

Design and Description of Operation

An overview of our integrated system is shown in Figure 1. Our system utilizes the NIOS II/F processor, running off a PLL at 50 MHz. At this speed, we achieved acceptable performance while maximizing system efficiency. Both our software source files and the output video frame buffer utilizes the 8MB SDRAM. DE2's onboard SRAM is used to feed the ADV7123 DAC for universal VGA output. Video-in data from both cameras are processed by various hardware components, including the ADV7181 ADC, and are connected to NIOS via the PIO interface. An elaboration follows.

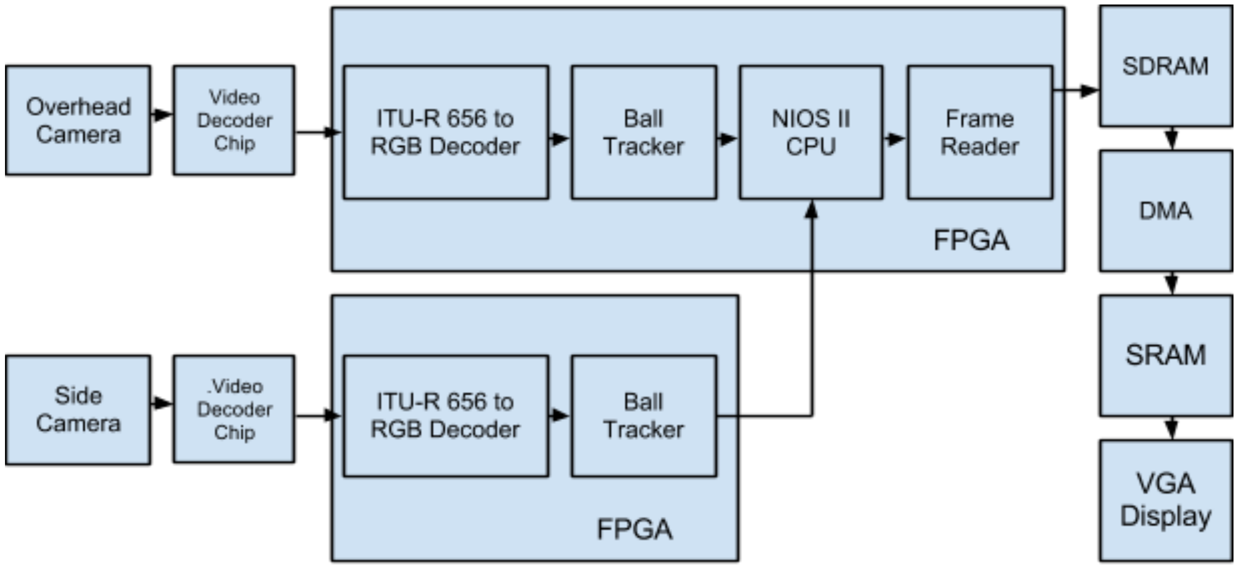

Figure 1: System Block Diagram

Hardware
The sampling of an object's position is performed by two orthogonally place cameras which covers both the x-y plane and y-z plane in the sampling zone. The Altera DE2 board that was used for this project has only one video decoder so two interconnected DE2 boards must be utilized. The communication between the boards only goes one way; data signals from the sender board are connected to GPIO pins while the receiver board connects its GPIO pins to a PIO interface of its CPU. Connecting the boards in this way allows simultaneous access to both camera's information directly by the CPU.

The conversion and object tracking performed by the FPGA hardware is described in detail in the following Hardware Design section. The calculation of the projected impact point and simulation output to a VGA display is described in the Software Design section.

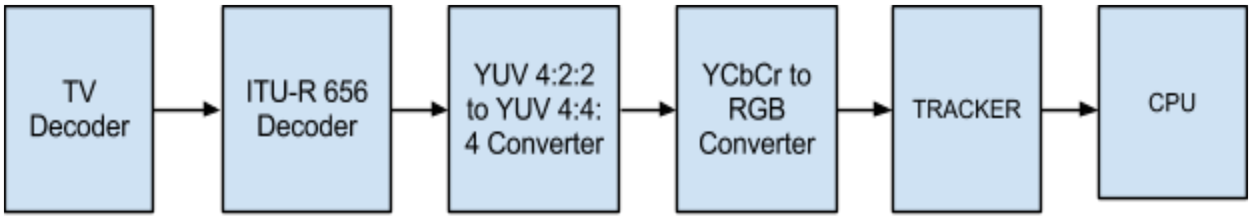

Figure 2: Data Flow for Path Prediction

Video Signal Decoding and Object Tracking

Camera signals undergo various hardware modules during processing. Figure 2 shows the data flow of our hardware component that inputs ITU-R 656 encoded data from the video decoder and outputs the current coordinates of an object to the CPU. Pixel data from the video decoder uses YCbCr formatting which consists of an 8-bit luma component (forms black and white images) and two 8-bit chroma components. We converted YCbCr data into RGB format before we perform pixel tracking for two reasons: better colour selectivity and increased tolerance to colour fluctuations. Colour selectivity refers to the ability to distinguish two different colours from one another by using only the provided colour bits of each pixel. YCbCr provides limited selectivity because red, green, and blue colours are distributed out in each component resulting in primary colours, which are easily distinguished by the human eye, having moderate bit values in each component. RGB formatting, on the other hand, will have high bit values in one or two components and lower bit values for the other components for primary colours and their complements. For instance, an orange pixel will have high red and green value but low blue value allowing us to easily pick out this colour from a frame. RGB formatting also provides increased resistance to colour fluctuations because each color component consists of 10 bits of data as opposed to the 8 bit wide components of YCbCr. The increased width of each component means that the most significant bits, which are sampled during tracking, are less likely to change when a slight colour fluctuation occurs. This is important for accurate tracking as our cameras are sensitive to even the slightest lighting changes and tend to self-adjust if and when this happens.

The conversion of YCbCr to RGB requires the use of several modules placed sequentially along a pipelined data path. Pipelining ensures that we keep a high throughput to allow the system to keep up with the output frequency from the TV decoder. ITU-R 656 conversion is performed first to combine the YCbCr components that are delivered separately over the 8-bit bus into alternating YCb and YCr packets over a 16-bit bus. The ITU-R 656 also performs a secondary function of providing the 10-bit coordinates of each bit in the frame. A conversion to YUV 4:4:4 is performed afterwards to combine the alternating YCb and YCr packets into a single 24-bit YCbCr packet. The YCbCr data is then converted into 30-bit RGB format which will be used by the tracking module.

A SDRAM buffer is normally placed in between the ITU-R 656 Decoder and the YUV 4:4:4 converter since decoding to RGB usually involves the output of the video images to a VGA display. Because we require video processing with minimum delay, the buffer is omitted and all the modules are programmed to use the same clock as the TV decoder. We didn't perform any downsampling in the hardware so that we can provide high precision values for the software calculations.

The tracking algorithm is done within a single module that takes in pixel data in RGB format and its respective coordinates within the frame and outputs a 20-bit coordinate to the CPU (10-bits for x value and 10-bits for y-value). Each pixel is compared to a user-defined colour that is set using the switches on the DE2 Board. Only the three most significant bits of the three colours are compared because lower significant bits are prone to fluctuations from slight changes in colour. Some matching pixels that may not belong to the projectile may be detected by our system so it will only take into account consecutive matching pixels. The coordinates of the center of a line of matching pixels will be saved as the module scans an entire frame. The saved coordinates gets replaced with a new set if a longer matching line is found within the same frame. The longest line of matching pixels should correspond to the middle of a spherical projectile so we should end up with the coordinates of the projectile's center by the end of the frame which is marked by an input coordinate of (0,0). The camera's output images are a superimposition of an odd frame with an even frame that have different images between each other due to the interlacing of Raster scan, and the output coordinate will rapidly switch between two values if we scan every frame. We made a conscious design decision to update coordinates only on odd frames - this reduces the amount of inevitable fluctuation in the coordinates.

Video Output

The video output portion of our project uses design cues as well as custom components described by Billy Kozak and Jeff Theriault in their CMPE 490 2011 application notes [3]. The Altera provided VGA controller proved to lack support and presented many bugs during interfacing.

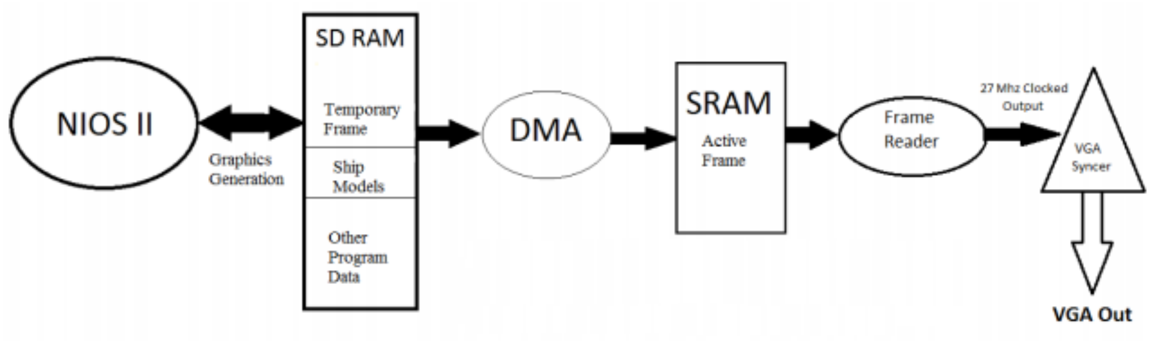

Figure 3: VGA Controller from CMPE 490 G5 Final Report [11]

This VGA controller design utilizes a sequential data flow that is highlighted in Figure 3. Two RAM memories, a SDRAM and SRAM, along with a direct memory access (DMA) module, are used to write to DE2's ADV7123 DAC converter before outputting via universal VGA.

As our software component receives coordinates from the Ball Tracker via the PIO interface, new graphical frames are created and saved to a temporary buffer frame inside the 8MB SDRAM. The DMA module can then pass the buffer frame to the SRAM, bypassing the CPU. This “active frame” in SRAM is then read by reader and syncer components to continuously display updated frames to a connected VGA external display. By separating the active frame and the next-frame-to-be, any changes made between the active frame and the next frame (drawing, erasing, redrawing) will not show up in the form of real-time flickering and inconstancy [11]. By placing DMA memory transfers in parallel with CPU tasks, instruction and data caches prevent competition between SDRAM and DMA, and the result is increased output stability, and minimized performance impact [11].

Our VGA controller also makes liberal use of the components described in the *VHDL frame reader and VGA syncer* application notes [3]. The frame reader acts as an Avalon memory mapped master interface to the SRAM [11]. It reads pixel data from the SRAM as fast as possible, buffering reads and writes accordingly without errors. It is needed because the SRAM is too slow for the VGA syncer to read pixels on demand without errors [11]. When a swap signal is received from syncer, read and write buffers are swapped, allowing the syncer to read already buffered lines in a fashion that is faster than could be achieved by reading directly from SRAM. The VGA syncer generates RGB, horizontal sync, and vertical sync signals at precise timings to produce an image on the screen [11]. The frame reader also performs resolution division, from 640x480 to 320x240, which is the resolution of our output frames. Additional information on either of these custom components can be found in the original documents in [3] and [11].

Software

The software portion of our project uses uC/OS II RTOS in the Nios II IDE. Its main focus is to create a UI that can navigate a user through the functional modes of the system, and to output relevant VGA frames during runtime execution of such a mode. It is also responsible for calculating the necessary kinematic functions when executing predictor mode.

Frames are created by writing to 320x240 sized arrays. Pixel color is depicted in hexadecimal. For example, writing 0xFFFF to every member of a 320 by 240 *short int* array would be equivalent to creating an all-white frame. By utilizing the “sys/alt_dma.h” library, we are able to control DMA write operations to transfer frames to SRAM for output.

Our UI alerts the user to the current system state by using 3x5 pixel text frames, and switches between states based on two button interrupts, for KEY1 and KEY3 respectively on the DE2 board. During tracker mode execution, a circle is drawn around a calculated centroid position in real-time for every frame outputted, simulating successful tracking of a spherical object in display. During predictor mode, several coordinates are saved after KEY3 is pressed. A kinematic calculator function then calculates impact point based on calibrated parameters. A detailed description of software design can be found in the software design section.

Calibration

Calibration must be performed to ensure that the projected point of impact displayed by the VGA display matches the actual collision point of the projectile within the display. The software calculation will be performed using pixels as units so we have to convert real world measurements of distances into pixels perceived by the camera. We will setup an appropriate sampling are by running the tracking function of our system to find the viewing limits of the camera. Once the sampling area is set, we would measure the dimensions of the viewing area to get a conversion parameter for converting inches into pixels. This is needed because our calculations are done in pixel units. Once we have the conversion parameter, we can set the display at a set distance from the sampling area and hard code this distance measurement and display dimensions into the software in pixel units.

Bill Of Materials

Parts List

CMOS Camera Module

Robust camera used as our source of path detection and provides the data necessary to compute path prediction. The 2 cameras will be placed to detect 3 dimensional path traversal by thrown projectiles.

Power Supply	12 V
Current Comsumption	150 mA
Video Bandwidth:	4.2 MHz
Frame rate:	(H)15.734 KHz (V)59.94 Hz

Units Ordered:	2
Unit Cost:	$34.95 (not including shipping)

Website: https://www.sparkfun.com/products/8739

Altera/Terasic DE2 development board

Units Used:	2
Unit Cost:	$517.72

40 Pin Ribbon Cable

Units Used:	1
Unit Cost	$10

Wooden Support

Units Used:	1

Unit Cost:	$2.99

Irwin Corner Clamp
Used to hold the horizontal beam which supports the overhead camera.

Item Depth	9 In.
Item Height	1.938 In.
Item Weight	0.75 lbs
Item Width	6 In.

Units Ordered:	1
Unit Cost:	$12.69

Website: http://www.homedepot.ca/product/irwin-corner-clamp-3-in/904190

Irwin Quick-Grip 990 Degree Angle Clamp
Holds the vertical beam that supports the overhead camera in place

Item Depth:	11.875 In.
Item Height	3 In.
Item Weight	2.5 lbs.
Item Width	4.75 In.

Units Ordered:	1
Unit Cost:	$27.99

Website: http://www.homedepot.ca/product/irwin-quick-grip-90-degree-angle-clamp/904202

Reusable Design Units

frameReader.vhd and vga_syc.vhd from 2012 winter application notes by Billy Kozak and Jeff Theriault
Modules that takes images drawn by the CPU and prints it unto a VGA display

I2C_AV_Config.v and I2C_Controller.v from Altera DE2_demonstrations
I2C interface and controller for initializing the ADV7181 Video Decoder

SEG7_LUTv and SEG_LUT_*.v from Altera DE2_demonstrations
Modules that outputs hex digits to the 7-segment display on the DE2 board

ITU_656_Decoder.v, MAC_3.v, DIV.v, Reset_Delay.v, YCbCr_to_RGB.v, and YUV422_to_444.v
Modules for converting YCbCr in ITU-R 656 encoding into RGB format

User Controls	
Inputs	
Control Name	Function
Analog Video In (Board 1)	Video input from the top camera
Analog Video In (Board 2)	Video input from the side camera
KEY(0)	Hardware Reset
KEY(1)	Start / Switch Modes
KEY(3)	Start path Prediction
SW(17) to SW(15)	Sets the 3 most significant bits of the red colour to match
SW(14) to SW (12)	Sets the 3 most significant bits of the green colour to match
SW(11) to SW(9)	Sets the 3 most significant bits of the blue colour to match
Outputs	
VGA port	Displays the position of the tracked object and the predicted point of contact
LEDG(7) TO LEDG(0)	Shows the output of Video Decoder

Ball Tracker Interface		
Inputs		
Port name	Function	Signal Width
CLK_50	50 MHz clock input	1
RESET	Hardware Reset	1
TD_DATA	Data input from the Video Decoder	8
TD_HS	Horizontal Sync from the Video Decoder	1
TD_VS	Vertical Sync from the Video Decoder	1

TD_CLK	27 MHz clock input used by the Video Decoder	1
Outputs		
HEX7 to HEX0	Output to the 7-segment displays	7 × 7
TD_RESET	Reset signal for the video decoder	1
TRACKED_x	X coordinates of object being tracked	10
TRACKED_y	Y coordinates of object being tracked	10

I^2C Interface		
Inputs		
Port Name	Function	Signal Width
iCLK	50 MHz clock input	1
iRST_N	Hardware reset	1
I2C_SDAT	Serial input/output data port to I^2C module	1
Outputs		
I2C_SCLK	Clock output to syncrhonize data signals with I^2C module	1

Top Camera	
Voltage	5V
Current	150 mA
Data	1 bit data stream to video decoder

Board		
Standby Mode		
Voltage	9.07 V	
Current	3.6 mA - 4.3 mA	Average: 4.95 mA

Power	44.9mW
Tracking Mode	
Voltage	9.07 V
Current	5.95 mA
Power	54mW
Prediction Mode	
Voltage	9.07 V
Current	7.35 mA
Power	66.7 mW

Background Reading

Due to the nature of our project, we began our search of background reading material with the methods employed by the Hawk-Eye tennis system. We took special interest in Hawk-Eye's impact-point determination method, 3D path reconstitution using 2D tracklets, and the use of a Kalman Filter for improving the path prediction [8]. In Hawk-Eye, upwards of 10 high speed cameras are calibrated to track the tennis ball on each frame of a 2D image plane. Because the Hawk-Eye project must take noise into heavy consideration for use in live sporting events, they employ spatially adaptive thresholding through the use of a Local Mean Removal algorithm to extract straight lines (the lines of the tennis court) as control points in the image plane [8]. With many 2D tracklets created, say after a rally, triangulation combines the 2D tracklets into several 3D partial tracks. A quadratic model is then used to find approximate joint points between 3D partial track pairs. Finally, a Kalman Filter is applied to these initial estimates, which produces the best prediction of projectile path and impact point possible [8]. In our project, we did not plan to implement our overhead camera's image processing with heavy noise mitigation, as we have a fairly controlled environment. As it stands, the heavy computation and complex calculations required by the methods employed by Hawk-Eye would not suit this task. However, the use of a Kalman Filter may provide us with a great alternative to our initial implementation-plan of our side-facing camera. Unlike the overhead camera, the side-facing camera will not be pointed towards a stationary tabletop or floor. Given that the Kalman Filter requires an initial measurement, or "given region" in one frame, and responds by finding the corresponding region in the next frame by finding the maximum correlation score in a search region, we may perhaps be able to utilize a Kalman Filter to accurately track our ping pong ball in a potentially noisy picture. An initial plan would be to use a constant velocity model for the motion model required by the Kalman Filter for estimation.

Further exploration of this possibility was researched through Hosie and West [9], where the prediction of a projectile ball's trajectory was explored in an uncontrolled environment. Here, a generalized behaviour model of a ball projectile was developed, and a Kalman Filter was used to refine parameters of the ball projectile's flight. Furthermore, path trajectory was calculated using iterative prediction using the last known velocity and position [9]. Although most of this article depicted the errors arising from uncontrolled, generalized environments of the projectile, it gave a clear description of the steps taken to apply a Kalman Filter for tracking a ball projectile. The ball tracking algorithm used depicts a 3 stage process - boot, update, extrapolate [9]. In the boot stage, the initial position and velocity of the ball is determined and a Kalman filter is initialized. Both the position and velocity are used to initialize the parameter prediction. An estimate of the state parameter error and measurement error are also determined in the boot stage [9]. A Kalman filter loop is used in the update stage to iterate the steps in the boot stage from remaining frames in the sequence, and extrapolation is performed after analysis of all image frames [9]. If we have time, we will refer to the technical specifics in this article to attempt to initialize our own Kalman Filter for implementation of the side camera.

Software Design

The software architecture for this project was written in uC/OSII RTOS, via Nios II IDE. The final integrated Nios project consists of 3 source files, *main.c, screen.c* and *fontGraphics.c* along with their corresponding header files. `Main` contains the project's main function, its primary FSM, and interrupt service routines. `Screen` contains DMA control functions, frame manipulation functions, as well as kinematic calculation functions. `fontGraphics` contain all alpha-numeric character array declarations used for the user interface, as well as some other UI specific state declaration graphic frames. Important color information, parameters, and structs reside within `screen.h`. Other than the standard `<io.h>, <stddef.h>, <stdio.h>, <stdlib.h> and <string.h>` C headers, we also used the Altera PIO interface library, `altera_avalon_pios_regs.h,` as well as their interrupt and DMA libraries, `sys/alt_irq.h, sys/alt_dma.h`. These libraries allowed us to interface correctly with hardware generated data, as well as control the operation of our DMA. The main purpose of software in our project is to draw relevant frames in response to tracking data, and also present a UI that can navigate through the functional states of the project.

Pixel frames, drawing functions, DMA writes:
Pixel frames are created using `short int` arrays. The main array representing the entirety of the screen, `pixelMap`, is declared as a 320x240 large array in which each member represents a color pixel to be outputted to VGA. Frame drawing functions are declared in `screen.c` and `fontGraphics`.c. Functions manipulating the main `pixelMap` frame receive the `pixelMap` as an argument, and manipulate its pixel members, essentially "drawing" on a blank array canvas. For example, the simple function `drawBkgnd` simply iterates through `pixelMap` using a *for* loop, writing a specific hexadecimal color representation to all of its members. This essentially draws a blank frame of a specific color.

With the exception of a big smiley face in the middle of our simulated dartboard graphic, all of the frame drawings modifications made in the main pixel map are done either via *for*-loop iterations for changing background color, alphanumeric shape drawing functions declared in `fontGraphics.c`, or a smart circle drawing function in `screen.c`. Alphanumeric characters are declared as 3 by 5 pixel graphics, and alphanumeric drawing functions simply iterate through the vertical dimension of the alphanumeric shape rather than drawing all of its members as a straight line. Our circle drawing function utilizes a double nested *for*-loop to iterate through the outer radius of a circle, and then fills it with a desired color. Simulated ping pong balls in tracking mode as well as the dartboard graphic in prediction mode are all created using this function.

Manipulations of integer arrays are not enough to output images to the external VGA display - they merely store the buffered frame into the SDRAM, from which the CPU is operating. In order for the frame reader, the VGA syncer, and the ADV7123 ADC to synchronize video output, we need to control DMA to write an active frame into the SRAM. This is done in three parts. First, we declared a "dmaState" struct to represent the *receive* and *transmit* channels statuses of the

DMA. Next, this variable is initialized via the `initVGAwrite()` function in `screen.c`, via DMA functions provided in the `sys/alt_dma` library. Finally, the `copyScreen()` takes a pixelMap as argument, and on return copies the pixelMap over to the SRAM, effectively writing the active frame. Notable operations in this function include flushing the DMA cache, the DMA functions `alt_dma_txchan_send` to queue sends, and `alt_dma_rxchan_prepare` to queue receives. In summary, a frame is displayed by initializing an array, initializing the DMA's state, and then calling `copyScreen` to copy SDRAM contents to the SRAM.

User interface, system states, state switching:
We created our system using one main task, with state switching based on using an interconnected series of button interrupts. Figure 4 shows the data flow diagram for the operation of software within our CPU.

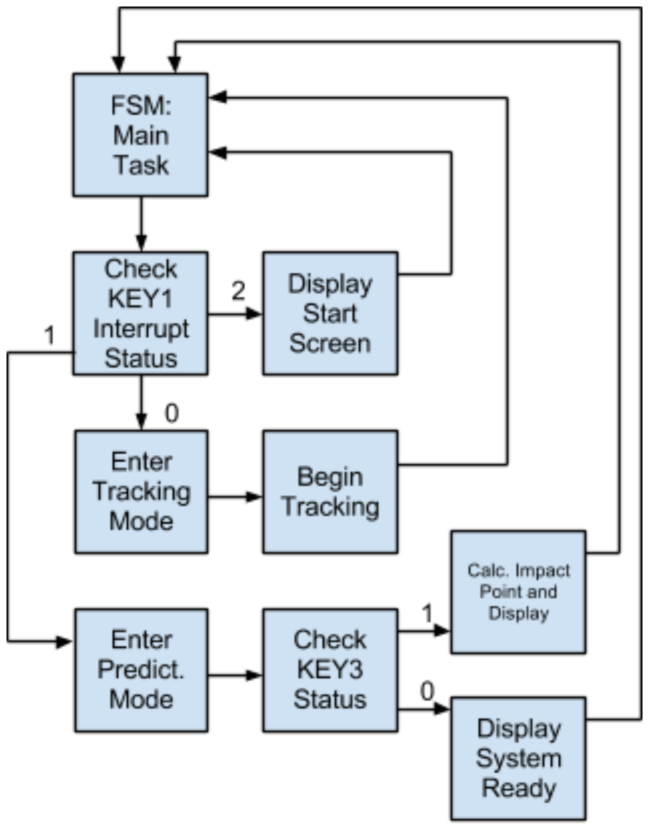

Figure 4: Software Operation

As you can see, the main finite state machine of our CPU undergo various state changes pending on variables controlled via pointers by interrupt service routines. For example, when the system initially begins execution, the pointer KEY1 points to a value of 2, allowing the start screen to be displayed. If button1 is pressed, its ISR will flip the pointer value between 1 and 0, effectively controlling system operating mode. In predictor mode, a similar philosophy is employed with KEY3 to control the beginning of a impact prediction run. A key press is required to regulate the start of tracking as the system needs to differentiate random tracked coordinates from actual coordinates tracked from a user's ball toss. This allows correct path calculation to take place.

Impact Calculations:

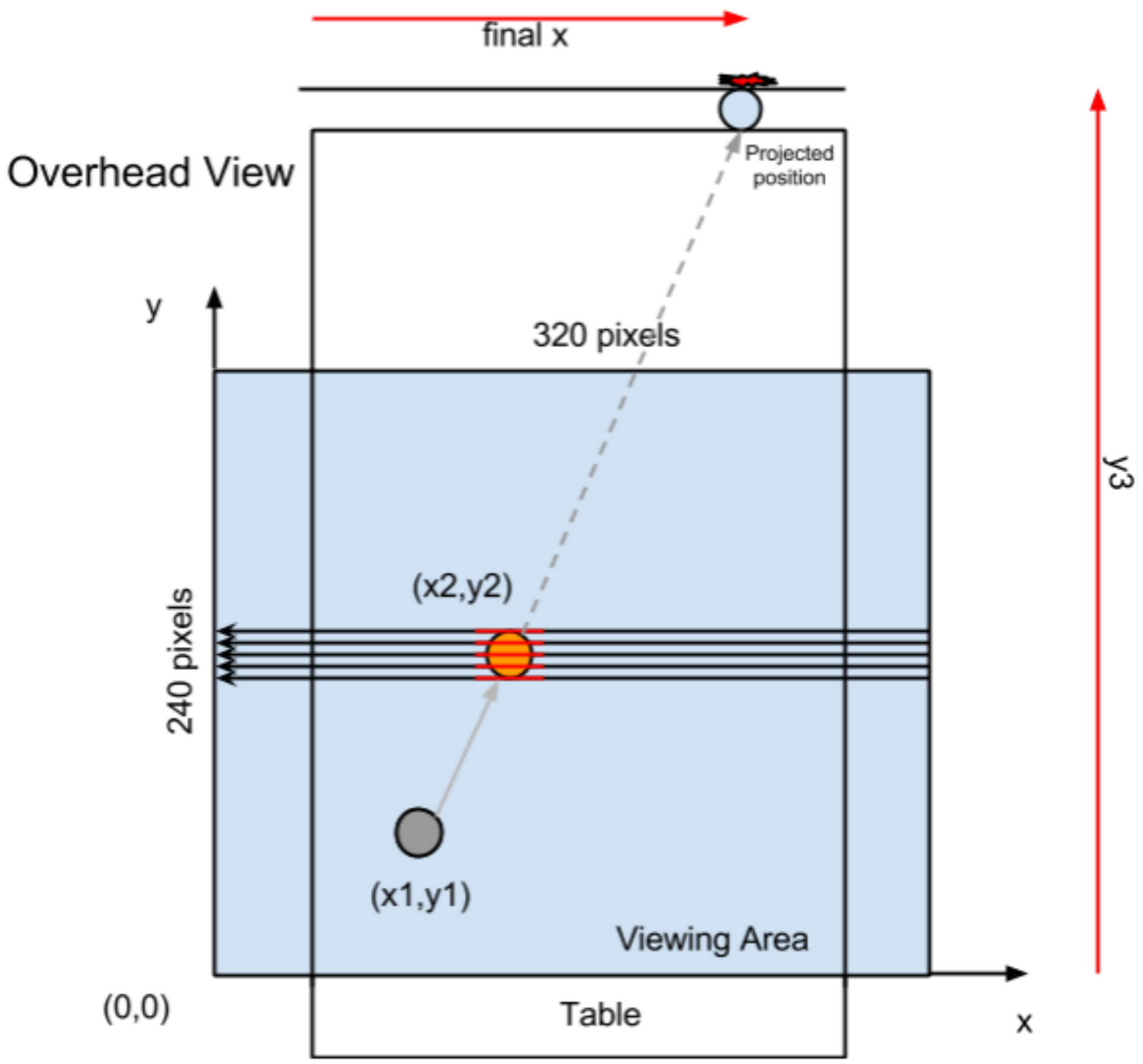

Figure 5: Path Prediction Calculations

Our software module is also responsible for employing the kinematics required to predict the impact point of our spherical object with a preset target. As described in the functional requirements section, we were unable to use the side camera to calculate 3D impact points. To calculate the x-coordinate of the impact point, we utilized the equation below.

$$x_{final} = \frac{x_2 - x_1}{y_2 - y_1}(y_3 - y_2) + x_2$$

This equation is implemented in the function, `predictX`, in `screen.c`. There are several considerations for this calculation. First of all, the hardware tracker performs tracking in 720 by 240 resolution. This is because the camera is outputting data in 720 by 480 resolution, but as mentioned in the design section, we only take the odd raster frames to reduce centroid coordinate jitter. Because of this, we need to size the received x-coordinate of the centroid down to the correct 320x240 resolution. This is done by a conversion to `double` for accurate division, and cast back to integer for use in the drawing functions responsible for the predictor mode animations. Secondly, because the ball tracker does not guarantee accurate centroid coordinates due to the camera's sensitivity to lighting changes, and because coordinates received in predictor mode can be extremely close to each other when a ball is thrown slowly, we need a method to increase accuracy. If two coordinates are received which are extremely close to each other, the calculated impact point based on their centroid may be very inaccurate due to flutter from the ball tracker. To increase accuracy, we take multiple centroid coordinates until the ball passes a threshold distance on the viewing area, at which point we compute the impact point. At this distance, the jitter becomes negligible, and accuracy is increased.

Test plan

Software Test Plan:

Image display to VGA
We tested the frame drawing capabilities of our software by initially drawing simple shapes such as circles and squares unto the VGA display. Simple animations were successfully performed to ensure the projected impacts points and the position of an object can be printed when they are available.

Calculation of projected path:
The coordinate samples taken by the CPU for trajectory calculation was tested by printing the coordinates of the sample to the console every time the CPU records the object's position. The results of the experiment showed us that two samples that are at an appropriate distance from one another were able to be successfully taken. The equations that would be used to calculate the future impact point is tested by printing the input coordinates and output results to the console. Calculation by hand verified that the software was performing the calculations correctly.

Hardware:
In order to have a successful project, we must ensure that all our components are working. The components that we need to test include the CMOS Camera, SDTV Video Decoder, the 512KB SRAM, and the VGA controller:

Component to Test	Test Method and Results
CMOS Camera	Connect the camera to a power source and connect the data wire of the camera to an oscilloscope. Move the camera around and look for any changes in oscilloscope screen
SRAM	We will run a customized memory test application through NIOS II. The provided memory test application tests the data bus, address bus, and whether each bit can store 1 and 0. We plan on customizing the code so that it tests half-word access since pixels would be stored as half-words. This test will be run on the entire span of the SRAM
TV Decoder	We will connect the decoder data bus to the CPU and plug in a functioning camera to the video port. In the CPU, we will poll the data lines and output to console while the camera is on and look for any changes in the data stream.
VGA	VGA was tested by running the DE2 demonstration which takes in a video input, converts it to RGB, and displays the image using the VGA.
ITU-R 656 Decoder + Resizer	We will connect the camera, the video decoder, the FPGA cores, and the CPU together. We will then poll the data stream lines from our decoder and resizer cores using the CPU to ensure that they are working properly
Object tracker	The 7-segment display on the DE2 board was used to display the recorded coordinates of an object.Some preliminary tests were performed to test the colour sensing and selection capability of the tracker by positioning object with different colours in front of the camera. This would tell us whether the pixel comparing algorithm is working or not. Once we confirmed that this is working, we will test the whole module using stationary objects, then slow moving objects, and then finally, fast moving objects to finish the test.

Integrated Testing:
After thoroughly testing the hardware and software components separately as described in the previous section, we will perform integration testing after combining the software and hardware

modules. Much calibration work is expected from the VGA output integration with the flight path calculator, to simulate the point of contact via VGA precisely where the projectile is expected to actually make contact with the wall. Also in need of calibration will be the placement of the cameras. We will find an optimal position for the cameras through trial and error. Integration testing is also where we will be testing the performance of the system as a whole. We may experiment with resolution of video input used to analyze the delay gain and the resulting path prediction accuracy. Final functional testing will follow to determine whether all functional requirements previously stated were met.

Testing Results

Tracking Mode

The system was able to successfully track an orange ball and display its position relative to the camera on a display screen. Using the tracking mode, we were able to set up the cameras to get an acceptable size for the sampling area. The following table shows the relevant measurements that will be sued for path prediction calculations

Unit conversion y-axis: 7.752 pixels/inch
Unit conversion x-axis: 6.803 pixels/inch

Measurements for Calculation		
Measurement	Value	
	Inches	Pixels
Sampling Area Width	47	320
Sampling Area Length	31	480
Sampling Area Height	47	480
Display to Sampling Area	47	480
Display Width	14	95
Display Height	11	85

Prediction Mode

The following table shows the results of running the collision point prediction function on our system. It is important to note that the coordinates that come from the overhead camera and the resulting position of the point of contact and are in pixel units. The result column displays where the point of impact will be displayed along the x-axis of the VGA display. Any projectile that yields a result below 0 or above 320 will be met with an out of bounds message because the display is unable to show the point of contact. Cases 1 to 3 shows cases where the projectile falls within the bounds of the display while cases 4 to 5 shows cases when the projectile fails to hit the display. Case 6 results from the ball being thrown away from the screen so an out of bounds message is displayed. The final case involves a slow moving projectile. As one can see from the result, the algorithm which prevents coordinate samples from being taken too close together was successful since the samples are 41 pixels apart along the y-axis. The system however cannot predict the collision point of projectiles that are moving too fast because the perceived color of fast projectiles are different from stationary or slow moving ones.

Prediction Mode Results						
Case Number	Coordinate 1		Coordinate 2		Result	Out of Bounds
	x	y	x	y		
1	153	251	137	15	135	No
2	92	196	196	15	204	No
3	244	223	68	15	55	No
4	108	177	27	102	-83	Yes
5	132	222	316	132	585	Yes
6	92	15	140	240	88	Yes
7	108	199	108	158	108	No

References

[1] Multiformat SDTV Video Decoder, Analog Devices Inc., Feb. 2013.
http://www.analog.com/static/imported-files/data_sheets/ADV7181B.pdf

[2] CMOS, 240 MHz Triple 10-Bit High Speed Video DAC, Analog Devices Inc. Feb. 2013.
http://www.eecg.toronto.edu/~tm4/ADV7123_a.pdf

[3] Billy Kozak, Jeff Theriault, "VHDL frame reader and VGA syncer," Univ. Alberta, Edmonton, AB, App Note, Feb. 2013.
http://www.ece.ualberta.ca/~elliott/ece492/appnotes/2012w/VGA_frameReader/g5_appnote.docx.pdf

[4] YUV Pixel Formats, FourCC, Feb. 2013.
www.fourcc.org/yuv.php

[5] TERASIC CYCLONE II EP2C35 Development & Education BOARD, Terasic, Hsinchu, Taiwan, Feb. 2013.
https://eclass.srv.ualberta.ca/mod/resource/view.php?id=232127

[6] Altera Embedded Design Handbook, Altera, San Jose, CA, Feb. 2013
https://eclass.srv.ualberta.ca/mod/resource/view.php?id=232123

[7] Altera Avalon Interface Specifications, Altera, San Jose, CA, Feb. 2013
https://eclass.srv.ualberta.ca/mod/resource/view.php?id=232125

[8] Jordan Tymburski and Rachita Bhatiat, "TV Decoder ADV7181B Chip," Univ. Alberta, Edmonton, AB, App Note, Mar. 2013.
http://www.ece.ualberta.ca/~elliott/ece492/appnotes/2012w/TV_Decoder_ADV7181B/

[9] N. Owens, C. Harris, and C. Stennett, "Hawk-Eye Tennis System," Proc. Int'l Conf. Visual Information Eng. (VIE '03), 2003

[10] R. Hosie, G. West, "Predicting Ball Trajectories in Uncontrolled Environments", Curtin University of Technology, 1994

[11] Billy Kozak, Nathan Sinnamon, Jeff Theriault, "Video Game with Wireless Accelerometer-Based Controls", Univ. Alberta, Edmonton, AB, April 2013.
http://www.ece.ualberta.ca/~elliott/ece492/projects/2012w/g5_video_game/g5_final_v2.docx.pdf

Initial Setup:

With the FPGA design flashed on the board, the setup of the project is fairly easy
1. Flash the design with project_top_level.vhd set as the top level to the first board and project_top_level_to_gpio.vhd to the second board.
2. Connect the boards to each other using a 40-pin rainbow cable.
3. Connect the top view camera to the board with the glowing red LED light using an RCA cable.
4. Connect the side view camera to the board with no glowing red light using an RCA cable.
5. Connect a VGA cable from the board to an external display.
6. Position the cameras and the display such that the cameras are orthogonal to each other and the display is perpendicular to the cameras.
7. Place an object to detect in front of the cameras and position the switches until the HEX display shows the correct coordinate of the object.

Controls:
1. Switches 17 to 9 are the colours to match.
2. Key 1 switches modes between tracking and path prediction.
3. Key 3 starts initializes the path prediction mode.
4. Key 0 is the hardware reset button.

APPENDIX B Future Work

We are keen on making the second camera functional in the prediction mode. We will look into increasing timing performance of the system to 100MHz from 50MHz, and test the timing accuracy. Another method we will investigate is to integrate an external timer via SOPC. By applying an interrupt ID to it, we may be able to gain important timing information for the projectile motion equation that is the source of our problems.

Furthermore, we want to implement a full, score-based dartboard game to complement our predictor mode implementation, rather than the simulation that we have right now. Not only will this be fun and challenging, it will make our system more complete functionally.

A final future endeavor would be the ability to function without highly contrast background. This will be extremely challenging as we will need to implement complex tracking algorithms such as the Kalman Filter, but it would allow our system to be used in actual real-life applications rather than set environments.

APPENDIX C Block Diagrams

Diagrams:

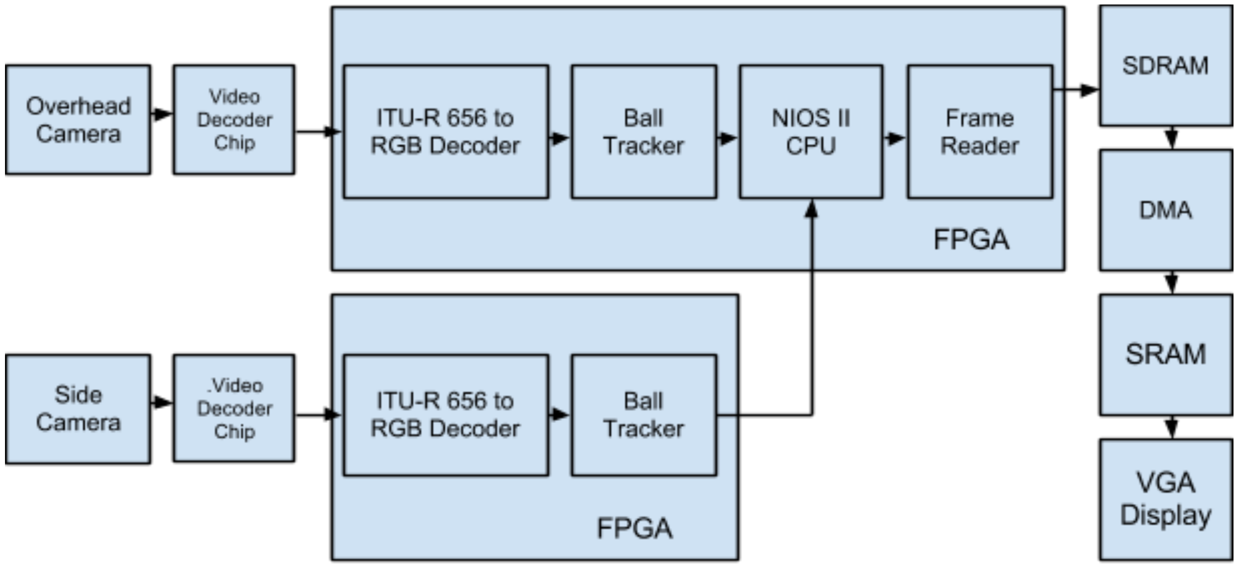

Figure 1: System Block Diagram

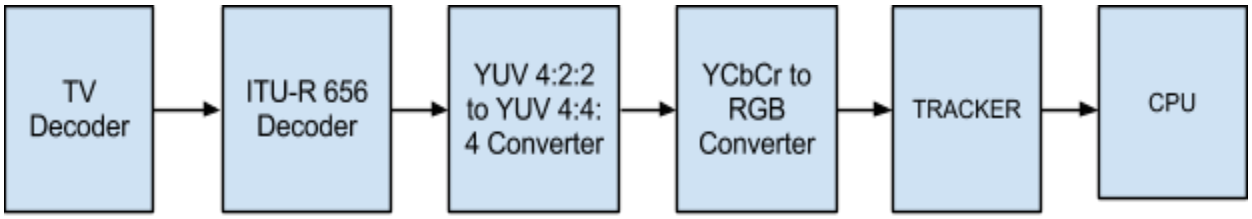

Figure 2: Data Flow for Path Prediction

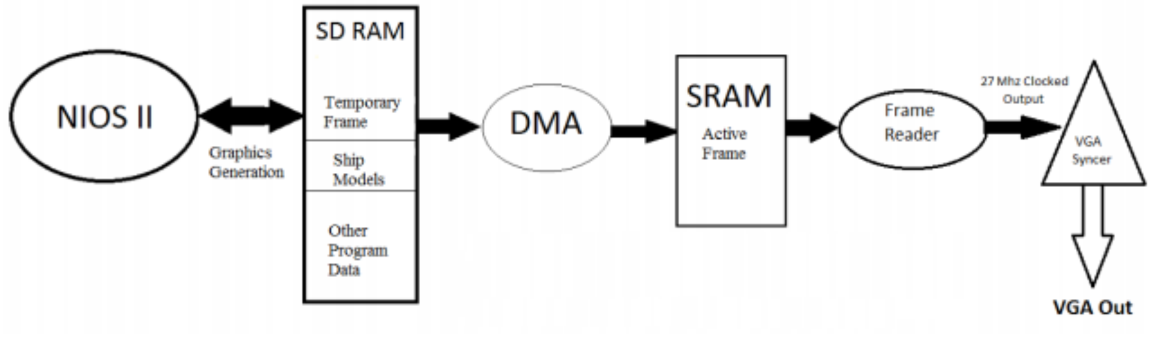

Figure 3: VGA Controller from CMPE 490 G5 Final Report [11]

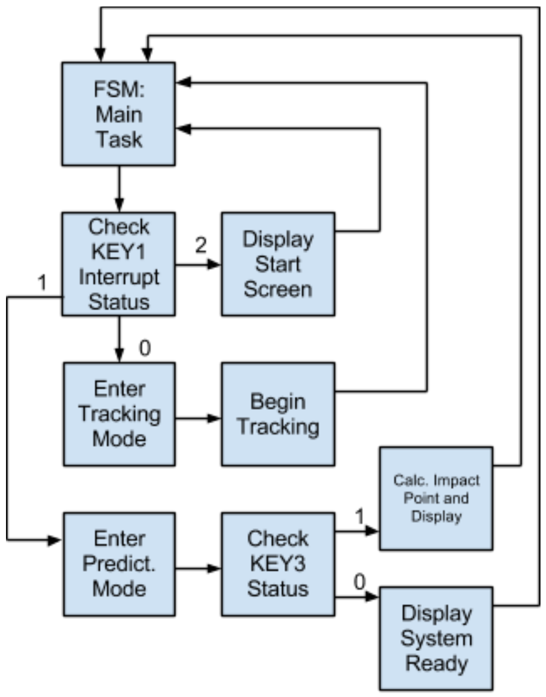

Figure 4: Software Operation

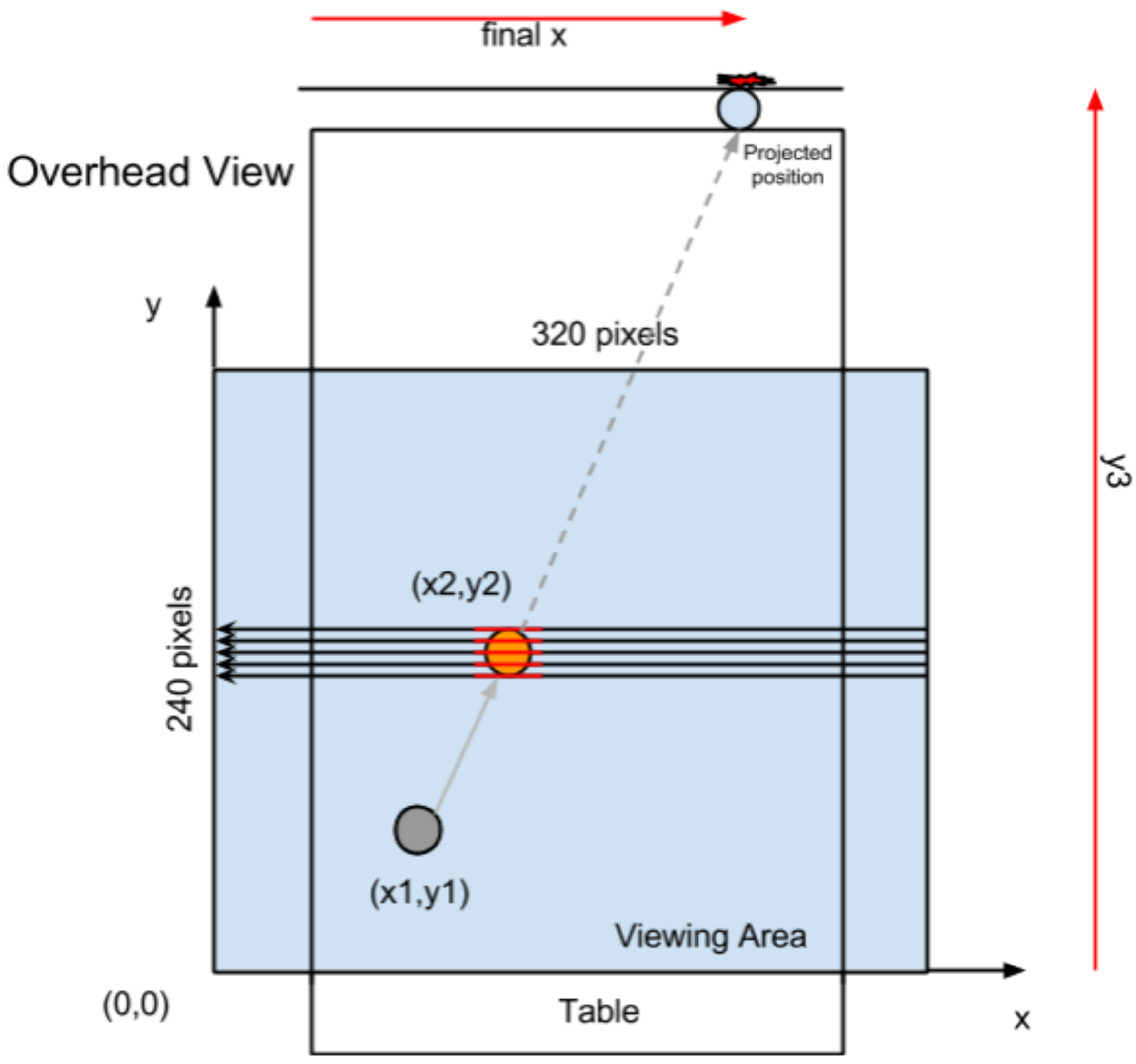

Figure 5: Path Prediction Calculations

APPENDIX D: Source Code

The code can be found tarballed on the server webpage. Here is an index to our source files:

Hardware:
Project_top_level_to_gpio.vhd
Hardware/I2c_av_config.v
Hardware/I2c_controller.v
Hardware/YUV422_to_444.v
Hardware/YCbCr_to_RGB.v
Hardware/Tracker.VHD
Hardware/TD_Detect.v
Hardware/SEG7_LUT_8.v
Hardware/SEG7_LUT.v
Hardware/Reset_Delay.v
Hardware/MAC_3.v
Hardware/ITU_656_Decoder.v
Hardware/DIV.v
Hardware/BALL_TRAJECTORY.v
Project_top_level.vhd
Onchip_memory2_0.vhd
niosII_system.vhd

Software:
Software/main.c
Software/screen.c
Software/screen.h
Software/fontGraphics.c
Software/fontGraphics.h