Air Defense System

Photon Phantoms

Nathan Woolf, Sam Cooper, Harrison Doll, Jake Neau, Cullen Krasselt

Team Website

Team GitHub Repository

University of Wisconsin-Madison

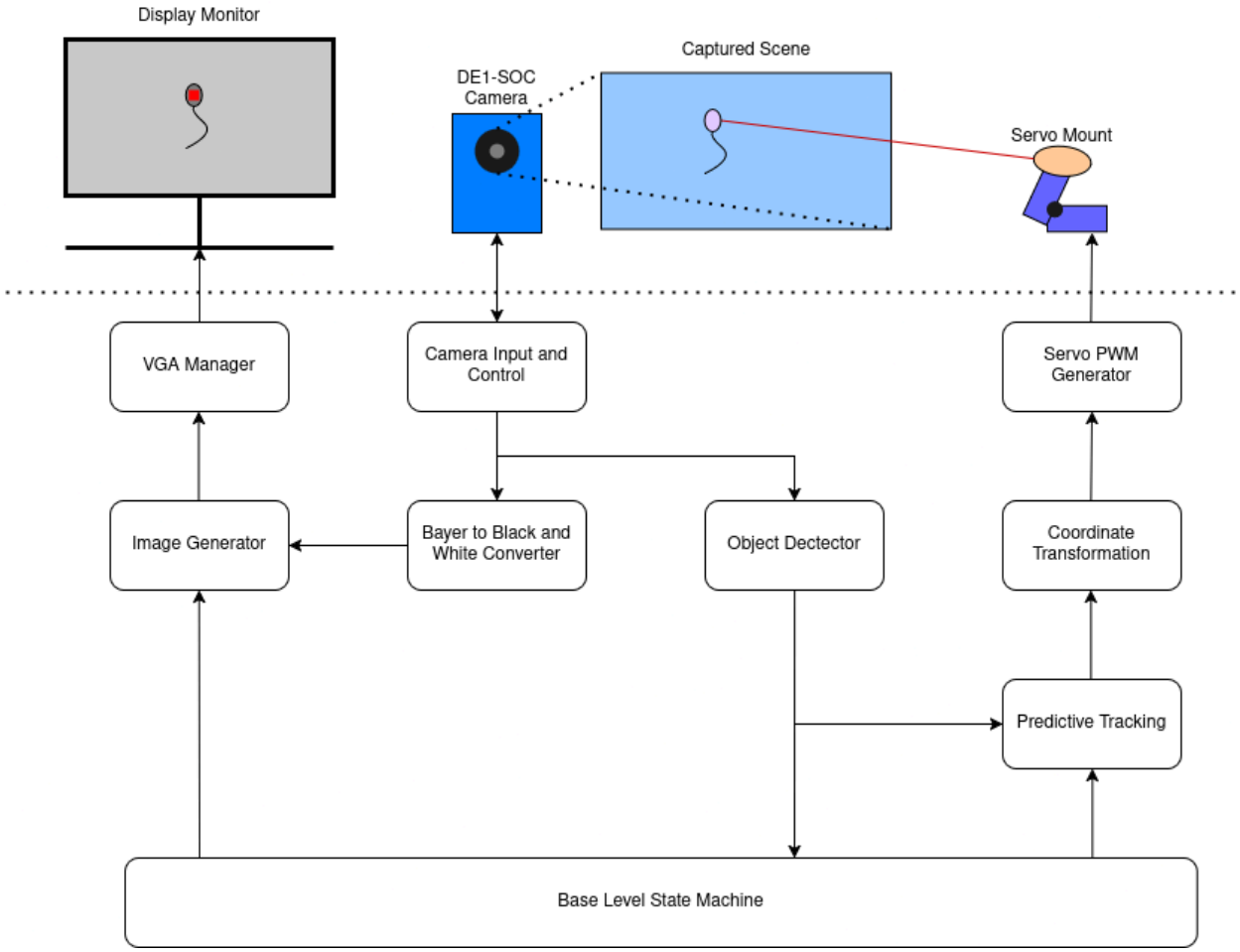
Project Final Report

Objective

At the start of this project, our team set out to prototype a budget mockup of an air defense system, similar to those seen in the defense industry. We began with general ideas for how to identify the object in a camera's frame and how to target it peripherally. Our team was inspired by this idea because each of our team members was challenged with different tasks we had to complete on our own initiative. Ultimately, we each took away valuable skills both in digital design and the engineering process we can deploy in the workforce.

Background

The project can be represented by a few basic blocks in the following diagram:



A detailed description of these blocks is included in the following section. The project required both designing FPGA units to perform logical calculations, as well as designing and tweaking hardware to

increase the repeatability of the system. The project uses the DE1-SOC, the DE1-SOC Camera, a laser module, a pan and tilt servo mechanism, and a 3.3V to 5V logic level converter as off the shelf parts. A 3D printed mount was designed to fix the camera in place and mount the servo with minimal parallax. The servo mount was lubricated to increase its precision.

Technical description

In the previous section, the project was broken down to a series of blocks. A more in depth description of each of these respective modules can be found below:

State Machine:

The state machine handles the high level control of the programmed system. The state machine receives coordinate information and a signal that indicates if an object was seen. There is a latch that records the object's last confirmed position when the ready signal was last active.

When an object is not on the screen, the state machine is in its idle state and the servo pipeline is given the coordinates for the middle of the screen. After an object is seen, the state machine transitions to the tracking state and the servo pipeline is given the coordinates of the last confirmed position indicated in the coordinate latch. When an object is no longer visible, the state machine waits half a second with a timer to confirm that the object is off the screen, then returns to the idle state.

Object Detection:

At the front of the project is the object detection system. All of the modules listed use the object detection output signals as drivers for their actions. The main output signals are a x position, y position and a signal indicating a valid object measurement. These signals represent the center mass of a moving object in the last read frame from the camera. The integrity of these three signals is central to the functionality of the project.

The first implementation of Object Detection functioned by storing all the red pixels from frame 0, and then comparing the red pixels of frame 1 to the pixels from frame 0. On a comparison of pixels the absolute value of the difference of the two 12 bit pixels is taken, if this difference is greater than a parameterizable threshold number then motion is confirmed for the pixel location being read by the camera. If motion is detected for a pixel its x and y value is saved. This difference formula happens for every red pixel in a frame, until the end of frame. Once a frame has completed there is a long window of clock cycles between the next frame. During this time, the mean x and y positions of the saved x and y values is calculated. If there is over a threshold number of values saved then we determine that the change is a valid object, and output a valid signal indicating that the mean x and y positions are a moving object's center mass. This cycle repeats for all the frames, with even frames being the "store" frame and odd frames being the "compare" frame.

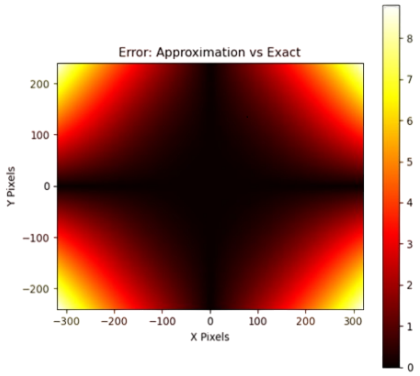
Predictive Tracking:

In our tests we noticed that the servo would fail to follow faster moving objects, likely due to the latency inherent to the design. We developed predictive tracking to allow the system to drive the servos ahead of where the target will be in hopes that it will be on target by the time the signal gets to the servo. To do this, we store pixel offset from the previous frame. We have this offset data for the current frame and the three previous frames. These offsets represent the movement of the object in the past four frames. We take a weighted average of these frames (current - 0.5, last - 0.25, 2nd to last - 0.125, 3rd to last - 0.125). We then have a final multiplier parameter to track those many differences ahead of the last fram. For example, if the multiplier is 2, we track 2 of the weighted average differences ahead of the current coordinates in both X and Y.

We noticed that parameter values above 1 tended to make the servo more unstable, so we stopped there. The slow tracking problem was reduced, but was not removed completely.

Coordinate Transformation for Laser Tracking:

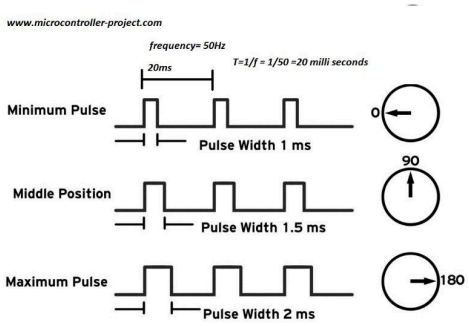
The project required converting Cartesian coordinates from the camera into spherical coordinates to drive the servos. Computing trigonometric functions directly would have required series expansions, which are both slow and resource intensive on an FPGA. Our initial solution involved using a lookup table with precomputed angles for each pixel coordinate. However, we found that by treating the x and y components independently, we could apply the small angle approximation $\tan(\theta) \approx \theta$, effectively eliminating the need for trigonometric functions and enabling real-time computation with minimal resource usage.



To validate this approximation, we generated this heatmap to visualize the resulting error. The results showed that most of the error was concentrated near the corners of the camera frame, while values along the axes and near the center closely matched the true angles. Based on this, we concluded that the approximation was sufficiently accurate for our application and that the performance gains outweighed the loss in precision.

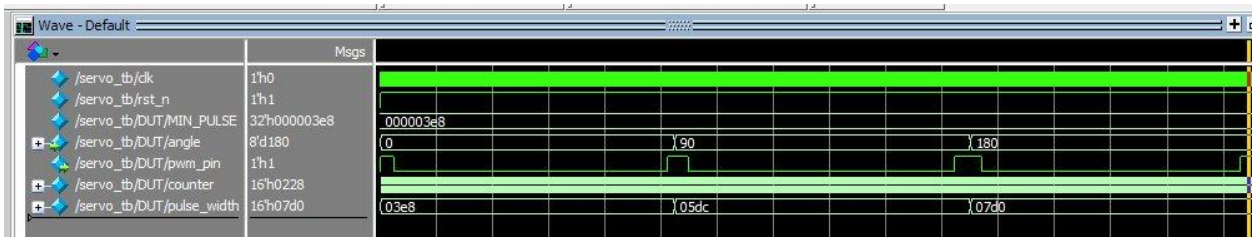
Servo Drivers:

To control the servos, the team decided to assemble servo driver modules to make their control simplified regardless of how position was communicated from image processing. The image to the right briefly describes how a servo motors PWM signal should be structured to achieve the desired servo horn position. The bounds of this specification were the motivating factoid to our servo driver. The formula relating servo horn position to duty cycle can be seen here:



Duty Cycle = 1000 + ((degree_val * 1000.00) / 180)

The resulting duty cycle has units of microseconds, this ultimately gives a greater degree of accuracy than if we used floating point or another means of data representation. Successful simulation waveforms can be seen below highlighting the duty cycles from the image above



Furthermore, the driver modules are parametrized to enable “overclocking” of the PWM signal frequency. Since the duty cycle, or the high pulse, of the period is constant (i.e 1ms for 0 degrees and 2ms for 180 degrees) we can shorten the low pulse of the period. This low period is in place to allow the internal comparator of the servo to correct its position. However, within our application, the servo is constantly receiving new positional commands and thus we can afford the sacrifice in accuracy of position in exchange for greater speeds. For reference, typical hobbyist servo motors operate at 50Hz (hence the 20ms period in the image above). In this application, we were able to push our servos closer to 300Hz with no apparent fallbacks on accuracy.

Evaluation highlights

Being able to present hard work that took us months to complete was an extremely satisfying experience. All of us worked incredibly hard to realize goals within the project, and it was especially valuable gaining practice with developing a project with no guidelines other than restraints. This of course relates well to industry, where your manager/employer expects you to do the best with the tools at your disposal, and use problem solving skills to get the job done. The best part of this project development was combining all of our separate modules into one product to present to the department. When it clicked together and worked for the first time, it’s the best part of problem solving. We also had great joy presenting the final product to faculty, alumni, and fellow students. It was exciting showing underclassmen what awaited them in the final portion of their engineering degree, answering questions about class options, and showing them how all of the skills we learned culminated in this project. It was invaluable showcasing our project to alumni, faculty, and industry professionals, given the opportunity to present our hard work, perseverance, and creativity to those that valued it the most. All of us thought the most exciting part of this experience was winning the instructor award for capstone design excellence. Being recognized for our hard work and knowledge on the subject is indescribable, and we all feel like ECE 554 was a fantastic ending to our undergraduate experience at UW-Madison.

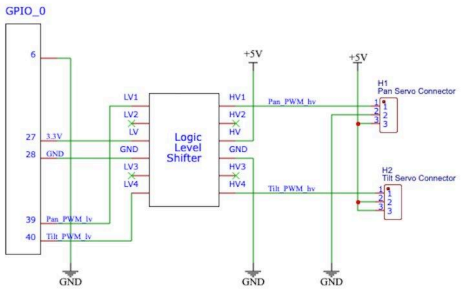
Challenges, workarounds, & lessons learned

FPGA GPIO pins voltage/current output rating:

When we were first planning our implementation, we were planning to use the GPIO pins alone to hook up the servos. Hence, we wanted to use all three signals of the servo: PWM, +5V, and GND to be driven from the GPIO pins. As soon we tested our setup on the FPGA, we realized we were having a major issue as the servos weren't really moving, they were just jittering around. As a means of testing, we hooked up an oscilloscope to the PWM signals to observe their behavior. This test setup showed us that we were getting both the correct duty cycles and frequencies. It didn't take long to see that the high duty cycle of the PWM signal was only being driven at +3.3V. This was our root cause of the servo drivers not working.

Our first prototype to resolve this problem was using an arduino. Our saving grace here was the high voltage threshold for a 5V arduino being around 3V. Our test setup was rather simple, feed +5V and GND from the arduino power pins, take the GPIO PWM signal output from the FPGA fed as a digital input to the arduino. From there, the servo gets the arduino's digital output, a one-to-one mapping of the respective input pin.

The arduino prototype worked perfectly, so we decided to implement it in the form of a logic level shifter ordered off Amazon. Understanding this board was simple, and easy to hookup. We made the decision to get +5V from an external power supply as we had worries the FPGA would not be able to source enough current for the collective loads of two servos and the laser. The high-level schematic of this additional circuitry can be seen to the right:



Pan-tilt servo mount binding:

One of the rather challenging problems to identify in our process of integration was the tilt component of our servos binding, preventing us from observing the full accuracy of our system. We initially attributed the lack of movement in the y-direction to our coordinate translation modules. However, after disassembling the servo mount and further observing the servos behavior, it became apparent that it was caused by the plastic in the mount giving the servo resistance, which it didn't have enough torque to overcome. By loosening the screws on the mount and adding some plastic grease in the mount, we were able to observe the expected servo behavior.

Low camera FPS:

The Terasic DR5B-D5M was specified to run up to 60 FPS over a VGA connection. However, the module we generated using the Terasic system builder was outputting ~15 FPS with severe ghosting. The smaller the FPS, the less frames we had to work with for using our object detection given that our implementation depended on buffering and comparing pixels within frames. With only 15 frames per

second, we could miss something moving in frame entirely. Reading the official Terasic manual, we realized that we configure the camera settings over I2C on the GPIO pins.

Table 1.7 Standard Resolutions

Resolution	Frame Rate	Sub-sampling Mode	Column_Size (R0x04)	Row_Size (R0x03)	Shutter_Width_Lower (R0x09)	Row_Bin (R0x22 [5:4])	Row_Skip (R0x22 [2:0])	Column_Bin (R0x23 [5:4])	Column_Skip (R0x23 [2:0])
2592 x 1944 (Full Resolution)	15.15	N/A	2591	1943	<1943	0	0	0	0
2,048 x 1,536 QXGA	23	N/A	2047	1535	<1535	0	0	0	0
1,600 x 1,200 UXGA	35.2	N/A	1599	1199	<1199	0	0	0	0
1,280 x 1,024 SXGA	48	N/A	1279	1023	<1023	0	0	0	0
	48	skipping	2559	2047		0	1	0	1
	40.1	binning	2559	2047		1	1	1	1
1,024 x 768 XGA	73.4	N/A	1023	767	<767	0	0	0	0
	73.4	skipping	2047	1535		0	1	0	1
	59.7	binning	2047	1535		1	1	1	1
800 x 600 SVGA	107.7	N/A	799	599	<599	0	0	0	0
	107.7	skipping	1599	1199		0	1	0	1
	85.2	binning	1599	1199		1	1	1	1
640 x 480 VGA	150	N/A	639	479	<479	0	0	0	0
	150	skipping	2559	1919		0	3	0	3
	77.4	binning	2559	1919		3	3	3	3

Using this table from the hardware specifications of the camera, we changed the I2C module from the system builder as follows:

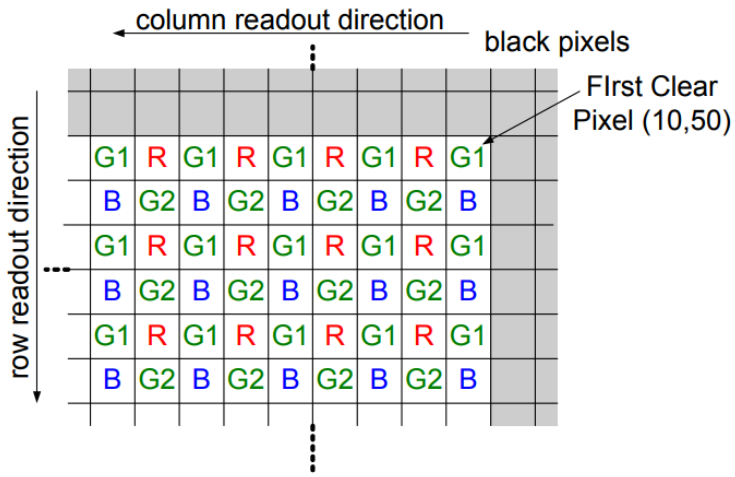
- Changed bits [5:4] of the registers Row_Bin & Column_Bin to 01
- Changed bits [2:0] of the register Row_Skip & Column_Skip to 001

The camera still captured at the full resolution of 2560 x 1920 pixels, but by changing both the row and column modes to implement 1x skipping & 1x binning, we reduced the resolution output to the VGA standard of 640 x 480 pixels, a 4x decrease. This gave us quite the boost in FPS as output to the monitor, but we still wanted to achieve the VGA maximum of 60 FPS. To do this, we lowered the exposure level in the I2C module as much as possible so that we could still see what was on the screen, finding a good balance of FPS and viewability. Due to these modifications, we achieved the specified maximum of 60 FPS on the camera, affording as much accuracy as possible for the object detection module (Terasic, 2008, p. 8).

Not enough on-chip memory to buffer frames:

In our object detection module, we wanted all of the calculations, data inputted, and data outputted to be as quick as possible. Due to this self-imposed restraint, we wanted to stick to using the fastest form of memory available to us - on-chip memory. However, due to the age of the board we were unable to buffer the entire frame despite reducing the captured resolution to 640 x 480 pixels. The total capacity

used from this was 640 * 480 * 12 (pixel size) = 0.4608 MB >> size of on-chip memory. We realized that we needed to optimize what data was buffered to compare against in every other frame. The bayer pattern output from the camera gave us some clues:



Due to the alternating patterns of the blue and red pixels, reducing the space needed to 320 x 240 pixels, we selected the red pixel to be buffered. This buffered data was 0.1152 MB, 4x less than the original idea. Using a FIFO as a storage device to implement this buffer, it fits perfectly on the on-chip memory. In this implementation we measured the X and Y coordinates of the pixel being outputted by the camera and if it was on an even row and odd column, it was detected as a red pixel and buffered into the FIFO. This buffering was originally done on even frames, and we used the odd frames as time to measure against the even frames, popping data off of the FIFO to read against. This implementation worked, detecting movement on the camera, but we noticed that the detection was still too slow, lagging behind objects if they were too fast (Terasic, 2008, p. 2).

Slow Object Detection:

The original implementation of object tracking outputs a new x and y position every other frame, which limits the speed of object tracking to (FPS/2). With a low FPS camera this isn't fast enough to track a quick moving object. To speed this up, we have two buffers, one for even frames and one for odd frames. This way, we can compare and store a frame's data in parallel outputting a new center mass position every frame. Saving every red pixel from a frame also requires a very large on chip FIFO. The boards provided did not have enough on chip memory to store two frames of red pixels in memory so we ended up comparing every other red pixel which reduced the memory used drastically. Sampling every other red pixel also did not affect tracking accuracy to any noticeable degree, which was the main tradeoff in doing this.

The final step in accelerating the object tracking was adding a small amount of predictive tracking. This is described in the predictive tracking section.

Contributions of individuals

- Camera Input
    - Sam and Harrison worked on manipulating the configuration of the camera to achieve 60 FPS, as well as buffer raw pixel data to be used in object detection.
- Object Detection
    - Sam and Harrison worked on developing frame buffers from FIFOs to collect pixel data, compare past frames to current frames, and output an objected_detected signal to the coordinate transformation module, with x and y coordinates.
- Base Level State Machine
    - Jake designed the base level state machine and worked on supplementary modules to interface it with object detection.
- Display Image generation
    - Jake worked on getting visual feedback of what pixel the servos were being driven to with a red dot on the monitor.
    - Jake worked on getting a stable display image without distortions.
- Predictive Tracking
    - Jake designed and implemented predictive tracking.
- Coordinate Transformation
    - Cullen assembled the module to translate coordinate positions within a frame to servo positions.
    - Cullen developed python models to understand the degrees of error expected from different approaches of coordinate transformation.
- Servo PWM Signals
    - Nathan built out servo drivers, driving GPIO pins for the correct duty cycle based on the inputs.
- Hardware
    - Nathan 3D modeled and printed the custom camera/servo mount as well as doing the necessary research on servos and motors for the additional hardware components of the project.
    - Nathan also prototyped and implemented the logic level shifter as a solution to our GPIO pins drive current challenge.

References

Terasic. (2009, June 10). *Terasic TRDB-D5M Hardware Specification.* www.terasic.com.

https://github.com/fuzzy41316/ECE554-Capstone-Project/blob/main/TRDB-D5M_Hardware%20specification_V0.2.pdf