Final Project Report

Depth Perception

UCLA CS 152B, Fall 2017

TA: Babak Moatamed

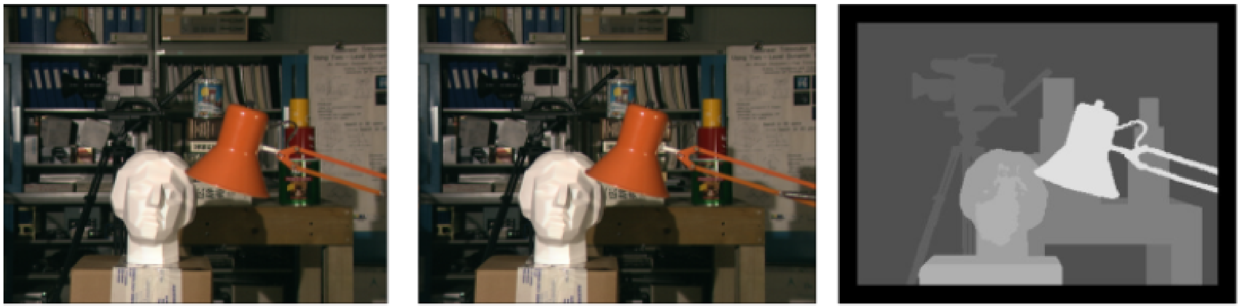
Group 1: Jeff Au-Yeung, Rahul Malavalli, Naing Min

1. Introduction

This report details the final project of the Digital Design Laboratory (CS 152B) at the University of California, Los Angeles (UCLA). The project was designed to give students the chance to design and implement their own choice of project that integrated hardware and software based on the information accumulated in the class. The project incorporated, at its core, a Field Programmable Gate Array (FPGA) and a corresponding Microblaze soft processor.

For our experiment, we attempted depth perception on an FPGA with a stereo camera peripheral. When two images are taken of the same scene from offset locations, as are taken in stereo cameras, the same object is also offset in both the images. By finding the pixel offset between the same object's location in both images, the actual distance of that object from the camera can be estimated. The larger the offset between the object in the two images, the closer the image is to the camera, and vice versa. You can test this phenomenon by looking at different objects with only one eye, then quickly switching to only the other eye without changing location; the objects furthest from you, such as the horizon, will "jump" the least between the two eyes, and the objects closest to you will "jump" the most.

To obtain these distance estimations, we compared the two images to create a disparity map, which visualizes how far objects are based on an arbitrary color or gray scale. The algorithm attempts to find how far objects have "jumped" from one image to another. An example can be seen in Figure 1, where the ground truth disparity map shows lighter images as closer to the picture and darker images as further away. The algorithm used in this project will be detailed in section 4.2 of the report.



Left Image

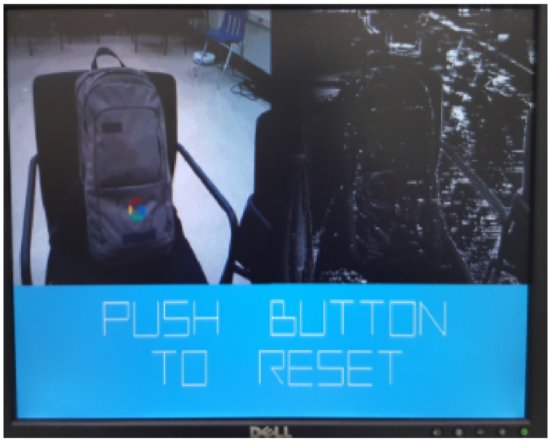
Right Image

Disparity map

Figure 1) The first two images show a picture of a room taken from slightly different angles. The disparity map created from the two images is shown the left and accurately gauges the lamp as the closest, followed by the bust, the table, and finally the camera.

To display the camera input, the disparity map, and an appropriate user interface, we utilized HDMI to output the information to a monitor. We also took input from push buttons and wrote letters to the monitor as part of the system. An example of the final system is displayed in Figure 2 below.

Figure 2) An example image of our project. The HDMI is connected to a monitor that is displaying a snapshot from the camera and a disparity map alongside it. The letters at the bottom are written at the bottom in our software.



2. Design Process Overview

To start the project, images needed to be retrieved from the camera. To do so, a VmodCam tutorial was used. The code provided configured and initialized the camera, which in turn wrote a live image stream into a specified location in DDR2 Random Access Memory (RAM). A Direct Video Memory Access (DVMA) module was then added and configured such that the HDMI would display on the monitor a specific segment of data stored in DDR2 RAM.

Next, computing a disparity map required a set of “rectified” images, or images that have been taken along the same axis. An unrectified set of images would need to be preprocessed to ensure that objects in one row in the first image appear somewhere in the same row of the second camera; this step is necessary to reduce the complexity of the disparity map calculation, which would otherwise have to search the entire image for matching objects, for every object. Thankfully, the physical configuration of the camera ensured that the input image stream would be rectified. As seen in Figure 2, the two lenses of the camera are fixed in specific positions, such that they are parallel to each other and therefore will ensure the images are pre-rectified.

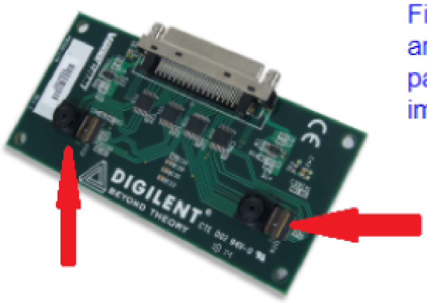


Figure 3) Picture shows the VmodCam hardware module. The red arrows show where the lenses are. Due to the two lenses being parallel to each other and soldered in place, it guarantees that images will have rows that correspond to each other.

In computing the disparity map, objects or patterns in one image have to recognized in the same row of the second image. To do so, “pixel blocks” from the first image are compared with all of the other pixel blocks in the same row of the second image. The pair of blocks that has the lowest Sum of Absolute Differences (SAD) between each corresponding pixel value is treated as a match, and the distance between the two blocks is taken to be the “disparity” between the two images for that set of pixels. The color displayed is determined by this distance, and the block is saved into a part of the DDR2 RAM. The disparity map algorithm itself will be covered in detail in section 4.2 later on.

Next, the DVMA is set up appropriately so that the HDMI reads values from the RAM and outputs them to a monitor. It should be noted that the disparity map takes approximately two minutes to generate, so a live disparity map calculation and display would be infeasible. Therefore, a simple user flow was determined to disparity map calculations on individual “snapshots”. In doing so, text was included on the monitor to indicate the current state and required user actions to continue the program. Specifically, a “LOADING” message is displayed when the camera is being configured; “PUSH BUTTON TO CALCULATE DEPTH” is displayed to prompt the user to take a snapshot; and “PUSH BUTTON TO RESET” is displayed after the disparity map has been calculated. A button on the FPGA controls the state transitions.

3. Set up

The following hardware is required:

- FPGA (Xilinx Virtex5)
    - DDR2 RAM
    - Push Button (GPIO)
    - HDMI connection
    - Microblaze soft processor
- Stereo Camera (VmodCAM)
- Monitor, connected by HDMI

All the above components need to be connected together. The appropriate block diagram is shown in Figure 4. The camera is connected to the DDR2 RAM, where it stores the RGB values of the pixels it reads in. This data in RAM is connected to the HDMI through the DVMA and gets sent to the monitor. Within the FPGA, Microblaze also has access to the DDR2 RAM so that the images can be pulled and processed. Lastly, the push button is connected to this soft processor via the GPIO bus so that we can control which state we are in.

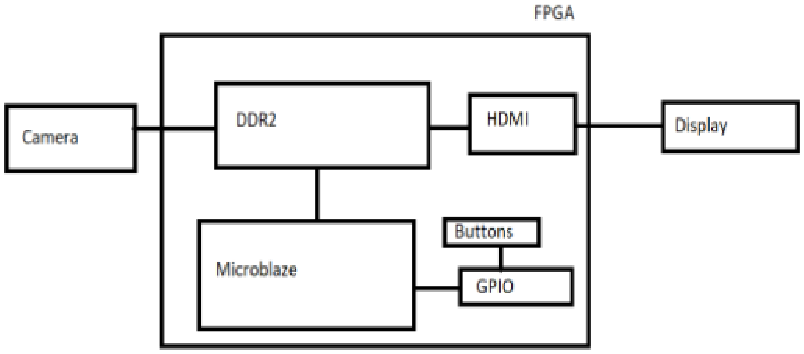


Figure 4) Block Diagram of our project.

4. Implementation Details and Algorithms

For our design, we implemented it so that it acts as a finite state machine with four states: the initialization, live feed mode, calculation and disparity map display. The state diagram is shown below in Figure 5.

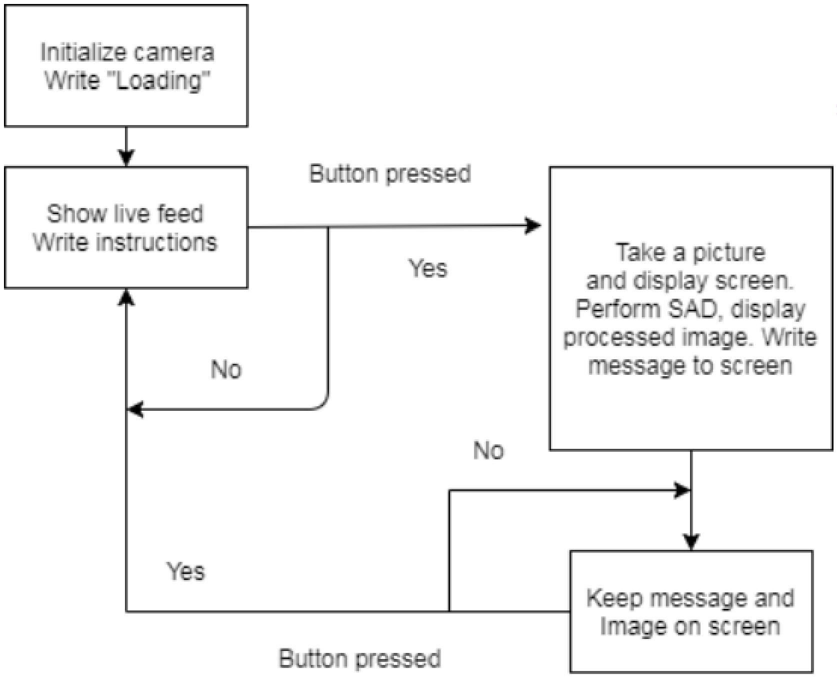


Figure 5) State diagram of our project. Goes between one initialization state, and two states which are switched between via button press.

The code for the initialization and loading stage is given to us in the tutorial. From our experiment, it takes approximately five minutes to set up the camera to display an image on the screen at the beginning. Then, the instructions for showing the live feed and calculating the disparity map were written by us and will be explained below.

4.1 Live Feed Display

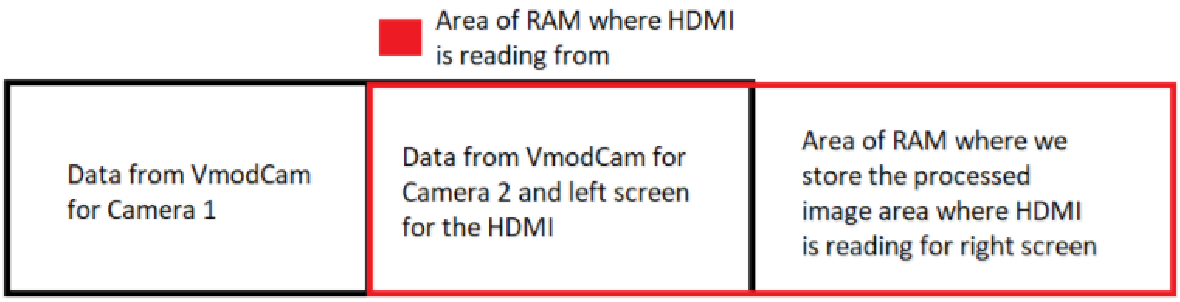

🟥 Area of RAM where HDMI is reading from
Data from VmodCam for Camera 1
Data from VmodCam for Camera 2 and left screen for the HDMI
Area of RAM where we store the processed image area where HDMI is reading for right screen

Figure 6) Example of DDR2 segmenting for the required data that will be stored and read.

Displaying the live feed is already set up by the tutorial. However, for our purposes, we only wanted to displayed one of the images the camera was capturing and displaying it to the screen. This required a small modification to the given code, namely changing the following line:

```
XIo_Out32(lDvmaBaseAddress + blDvmaFBAR, HDMI_LOC_START); // frame base addr
```

This change sets the location from which the HDMI starts reading. Setting the location to the beginning of the second image from the VmodCAM allows for the appropriate effect. This can be seen in Figure 6, where the red box starts at the camera 2 data rather than camera 1.

4.2 Disparity Map Calculation

When we wish to calculate the disparity map, we first capture a snapshot of the image we will be performing our calculations on. We initially calculated the image that was showing on the screen but this required holding the camera for the entirety of the calculation process which could be upwards of 5 min. By taking a snapshot and saving the data onto a part of the RAM, we only need to hold the camera still for a couple of seconds and then set it down while the calculation is being performed.

For this purpose, we begin by transforming our captured image into a grayscale one, using the formula :

```
grayscale =(0.3*R + 0.6*G + 0.1*B), for red (R), green (G), and blue (B)
values
```

While grayscale is usually calculated as:

```
grayscale =(0.33*R + 0.33*G + 0.33*B), for red (R), green (G), and blue (B)
values
```

This equally weighs the three colors, the first one gives a much better output due to the wavelength of each color being taken into consideration. Turning the image into grayscale helps to improve the quality of the disparity map as it reduces the noise that comes from the camera and lessens a lot of discoloration that happens. This discoloration will be discussed in a later section.

Then we use Sum of Absolute DIfferences (SAD) to find the block in the other grayscale image that best matches the one in our current one. SAD simply looks at the block and calculates the absolute difference between the color values of each pixel within the block. The one with the lowest difference is returned and the distance between the two pixels is saved.

Finally, the distance is used to create a ratio between it and the maximum distance it can be, in our case 640 pixels as that it the length of one row in our images. Initially, the distance was divided by 40 to ensure 640 stays within the 16 values the color could be. Instead, this value was lowered to 10 which gave a much better output.

The high level algorithm is shown below. One thing to note is that we eventually used a BLOCK_WIDTH of size 1, which means that every row of only a single pixel height.

```
For each column incrementing by BLOCK_WIDTH:
    Mindiff = MAX_NUM
    Distance = 0
    For each row incrementing by BLOCK_WIDTH
    Create a block of BLOCK_WIDTH size which will be compared to other
    blocks to the left of it in that row
    Calculate SAD for each block with each block the left of it.
    If SAD < Mindiff
        Mindiff = SAD
        Distance = distance between two blocks
    Output = distance/10
Print pixel to monitor with color value R,G,B = output
```

As a baseline, we ran the algorithm against a set of stock pictures displaying two Clorox bottles and a wine bottle, resting on a table. From visual inspection, we can see that the rightmost bottle is the closest, the wine bottle in the middle, and the last Clorox bottle is the furthest away. In Figure 7, we see the output which shows the brightest colors for the closest Clorox bottles and subsequently lighter colors for the other two bottles. It should be noted that the image resolution is very similar to the VmodCAM's but the quality and consistency is much higher.

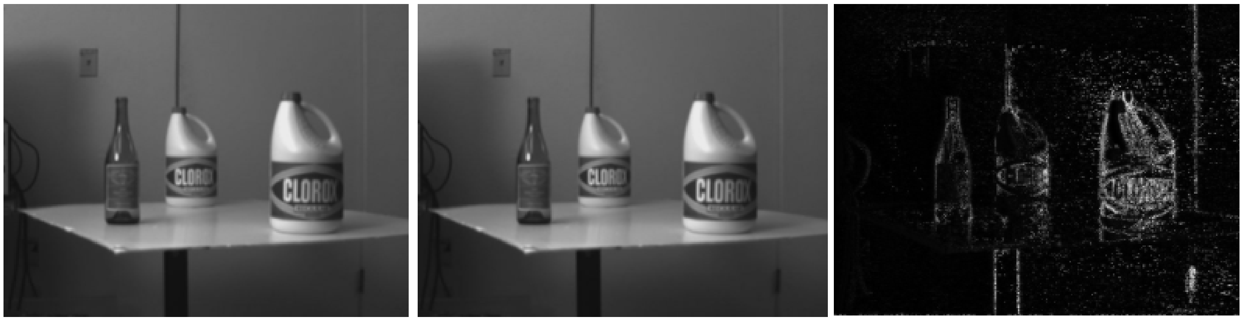


Figure 7) Image for a couple of bottles sitting on top of a table. The left image on the left, the right one in the middle and disparity map from running our algorithm on the right.

4.3 Text on Display

For all of our states, text is written out that explains the current state or the action required to proceed to the next state. Implementation of this is very similar to how we light up the 7 seven segment LED display on the FPGA boards from CSM152A. To create letters, 16 segments were used and writing specific ones to the HDMI screen would write out a specific letter. An example is given in Figure 9.

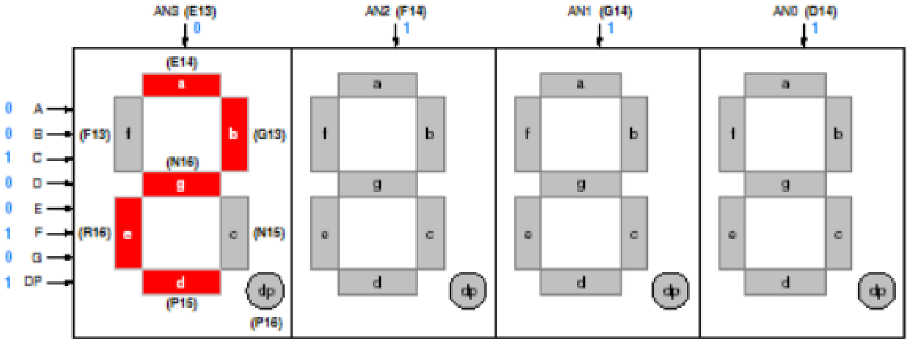


Figure 8) Example of the 7 seven LED display. Here we see that there are parts that are capable of lighting up and lighting up certain ones will display a number out.

http://ollintec.com/SistemasDigitales/libros/FPGA%20Piano%20in%20VHDL_archivos/7-seg_1.png

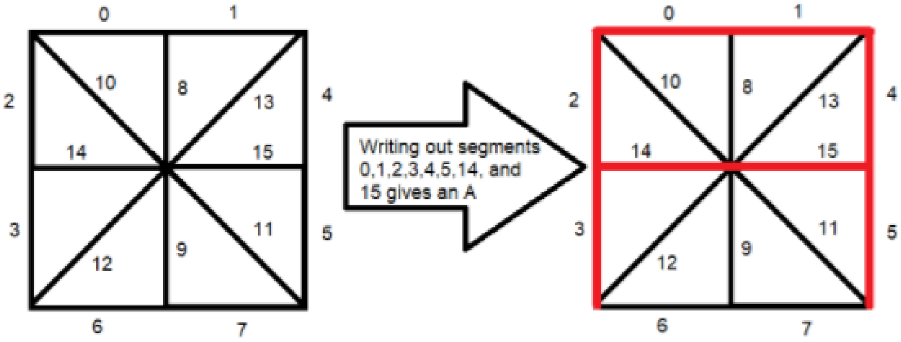


Figure 9) Layout of the 16 segment display used to write out our letters. Example shown writes an A.

4.4 Push Button and state implementation

As mentioned earlier, we chose to make our design a state machine with the different states shown in Figure 5. To rotate between the states, we used the press of a push button as trigger to switch. To do this, the button input is read from the GPIO bus and a seperate function is used to indicate whether to stay in the current state or move into the next one. To account for button debouncing, we used a busy for-loop that does nothing for a set period of time after a button press is detected.

5. Real-World Viability

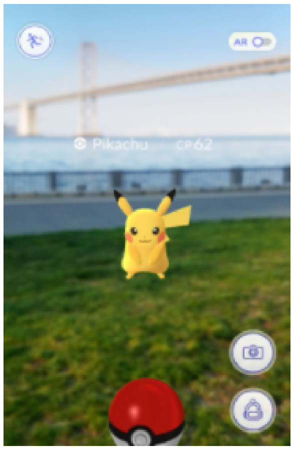


Figure 10) Pikachu seeming to float in the air since distance is not being calculated.

Here we will discuss the viability of creating disparity maps in the real world. Currently there are some interesting applications that, such as equipping drones with depth sensing capabilities and having a better experience in augmented reality. This can lead to drones being much better at interacting with the environment, i.e. less collisions and Pokemon Go having pokemon standing on the ground instead of seemingly floating in the air. This method of depth perception works better than using an IR sensor due to the specific application that it could be used in. IR sensors use reflected light in order to tell how far away an object is. Since Pokemon Go and drones are for outdoor use, IR sensors would suffer from ambient light sources like the sun or reflections interfering with the captured image.

5.1 Costs

When discussing introducing something to a market, many investors main concern is how profitable the product will be. For this project we used the computing power of an FPGA board alongside a 20 dollar outdated camera that has very poor image quality as shown in Figure 11.

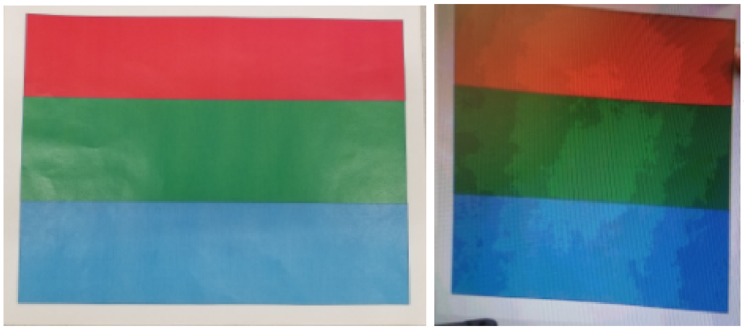


Figure 11) Two images of the picture. On the left, we have the picture taken by a smartphone and on the right with the VmodCam. The right image clearly shows a poorer image quality with the green line showing dark and light green.

We can upgrade our hardware to have a more specialized CPU like an ASIC do the software computation. A better camera can be used that takes better image quality such that there is less discoloration and noise from surrounding pixels.

The great thing is the technology is already implemented in today's technology. Recently smartphones have been released with dual lens camera which can be used to provide a stereoscopic image. The interiors of a smartphone already contain a powerful CPU which can be used to compute a disparity map meaning the initial cost of implementation is kept relatively low. There are also alternatives to stereoscopic images, done by using a single lens and motion detection.

5.2 Industry Standards

There is a problem regarding standards that must be conformed to. For cars, this means making sure it has the relevant safety measures to ensure humans can operate it with their lives at minimal risk. For our project we must look at the standard for capturing images due to the camera taking pictures as well as computer-human interactions that are required for the disparity map calculation.

When looking at the visual standards that must be applied, the VmodCam falls far below. Looking back to Figure 11, we can see that the color quality is pretty low. The camera supports a 12 bit color spectrum, 4 bits for the red, blue and green colors. When compared to cameras and smartphones that are released today, it majorly falls behind the color spectrum that more high-end devices can output. The refresh rate is also laggy, running at 15 FPS which a lot people believe is below what is necessary for a video to be acceptable.

Lastly, there is the delay that comes from all of the slow components. Initialization of the camera requires upwards of 5 minutes for it to be active. When a smartphone's camera app is opened up, it instantly opens up and an image can be immediately obtained. Computing the disparity map then requires an additional 2 minutes to get through calculations. When looking at the execution time, most people would be unhappy with the wait times.

6. Conclusion

We found our project is the most challenging at the beginning of it due to implementation issues. We have spent a lot of time trying to get the camera to work properly. Our initial implementations of our algorithms caused the program to run painfully slowly. When computing the SAD, we would need to compute each block with another block in the same row. At first, we would use the XIo_In16 instruction to read a block. But we would read individually read every block for every iteration causing a lot of overlap data to be read. So an optimization is done to read in the entire row and be cached so that it won't have to perform as many IO instructions. This gives a massive 3 times speedup.

Overall, this was an interesting and challenging project to work on. If we had more time, we would try to implement more features like binning the values of the disparity map so that it is much clearer, and actually calculate a value for the depth or optimize the program even more so that it could use hardware to perform parallel calculations.

References

Birchfield, S., & Tomasi, C. (1999, January 28). Depth Discontinuities by Pixel-to-Pixel Stereo. Retrieved December 15, 2017, from http://robotics.stanford.edu/~birch/p2p/