digitEyez

Claire Traweek, Kendall Garner

Overview

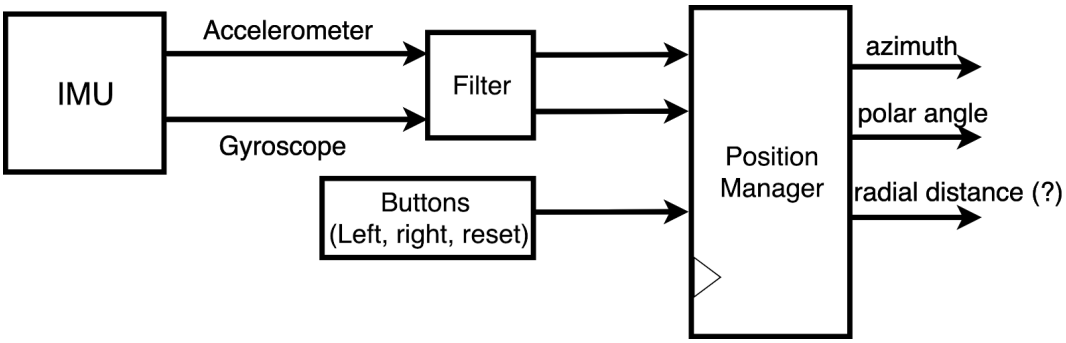
We plan on using the Nexys 4 DDR to make a responsive stereoscope. We will do this with an IMU, two small color OLED screens and the biconvex lenses commonly found in Google Cardboard headsets. We plan to design a scene that changes as the user tilts their head, giving the impression of being immersed in a virtual world. Ideally, the stretch goal for this project would be to generate the environment from music like Windows Media Player or similar visualization tools.

Hardware + Extras

- 3 LCD displays: used as the displays for the glasses. Because of how close they are to the user's face, we're opting for HD displays, which commonly come in 240x240 pixels.
- 1 IMU: used for getting acceleration and gyroscope data, to respond to user input.
- 1 "glasses" frame: something akin to the Google cardboard headset that is capable of holding two displays in place
- 1 FPGA: We intend to use the Nexys 4 DDR, but if the amount of resources needed to generate the images and process sensor data is not excessive, having a smaller FPGA would allow for more mobility
- 1 hat/FPGA mounting apparatus: in the event that we require the full processing power of the Nexys FPGA, we would need to construct some apparatus to place it relatively close to the glasses. We've been thinking about attaching it to a hat, but are concerned about breaking it. For development we'll accept a limited range of motion and keep the FPGA on the table, at least to start.

Modules

Position Module



The position module will be responsible for detecting changes to the user using spherical coordinates in two axes: the azimuthal axis (left and right, from 0 to 359 degrees), and the polar axis (vertical, from -90 to 90 degrees). At first, this calculation will be done through two input sources: the IMU accelerometer, and buttons. The IMU will be responsible for calculating the angle between the glasses and horizontal via the accelerometer. When the glasses are tilted towards the ceiling, it would report a positive angle, or a negative angle otherwise. To prevent sudden movement or natural shaking from resulting in an extremely shaky image, the IMU data would be passed into a filter. Similarly, pressing a button will control the azimuthal axis.

Targets:

- Minimal: IMU controls vertical movement, buttons control horizontal movement
- Goal: IMU accelerometer controls vertical movement, gyroscope controls horizontal movement. A risk associated with this task is that, while the gyroscope can detect movement, there is a risk of drift.
- Stretch: Position manager is able to detect movement (forwards/backwards). This requires more state, as well as determining rough forward movement.

Complexity and Debugging

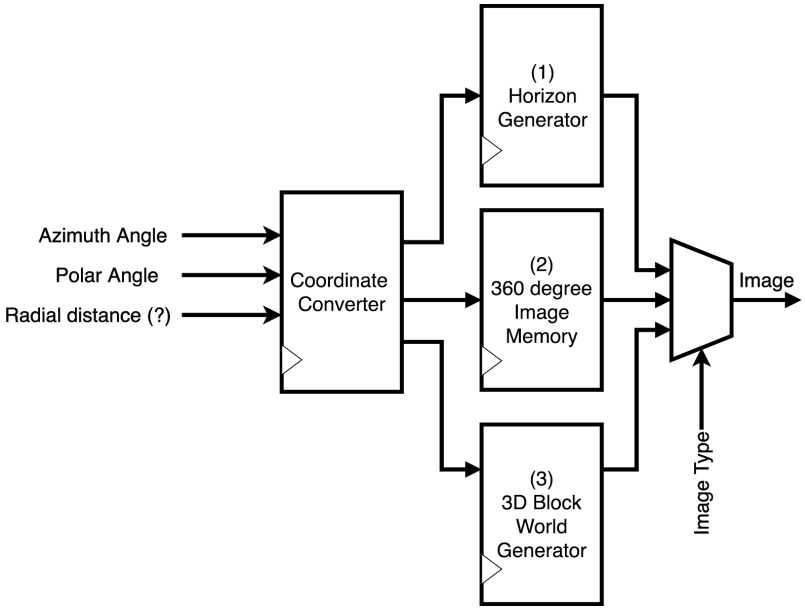
Since this module depends on input from an IMU or similar component, the calculations will be tied to the clock of the IMU and the rate at which data is received. For example, if we used components similar to that of lab 5B, then we would expect to update the position roughly every 4 ms (100Hz). Aside from multipliers for passing values into a filter or potentially scaling accelerometer/gyroscope readings, this

component only needs a few registers for state and adders for computation (on the order of 10s of btis, depending on the precision needed). Debugging the data receiver can be done using a combination of the ILA, hex display, and scope probes. Debugging the data itself will likely be done using the ILA or hex display, with serious challenges specced out in test benches.

Task Division

Due to lab 5 divisions, Kendall will take the lead on this task.

Image Representation (memory/generation)



Once the current coordinates have been calculated, they have to be converted into something meaningful before generating an image. We take images stored in jpegs, present them onto a hemisphere, and then determine how the image intersects with the user's screens. Once we have these new coordinates, they will be passed into other components to generate images. We have three possible approaches for generating images (ordered in increasing difficulty): a fixed/generated "horizon," a 180 or 360-degree image stored in memory, or an auto generated 3D figure. To reduce computational requirements, we would start by working with grayscale images, reducing the amount of data needed for a pixel. Each image generation module would be able to interface with the VGA output in lab for debugging, and also output data for smaller screens, likely communicating over SPI.

We have three different scenes we want to be able to display. The first, and simplest, would be a hard-coded/generated horizon. This “horizon” would consist of layers of colors that appear to remain horizontal to the viewer. Calculating the image pixels would be reasonably easy, as we could associate entire rows/columns with a single color. A (relatively significant) step up from this would be to use 180- and/or 360-degree images, akin to those of Google Maps street view. The challenge here would be on determining how to store the values in a memory (assuming memory can even fit such an image), and converting the spherical coordinates to the now-flattened images. A stretch goal would be to take this static image a step further, and generate three-dimensional figures (cubes, rectangular prisms, etc). This would require 3D graphics, and converting the spherical coordinates to something that can appropriately describe how to view a shape.

Sample source image format--the image will need to be converted to grayscale and resized before its uploaded to the Nexys. However, it’s already in equirectangular-panoramic format, which is what we plan to work with.

Targets:

- Minimal: be able to represent a fixed horizon (colors changing based off of vertical, horizontal axes)
- Goal: be able to represent and store a hemisphere or spherical image (akin to something from Google maps) using memory
- Stretch: be able to generate three-dimensional figures that change appearances based off of the angle viewed

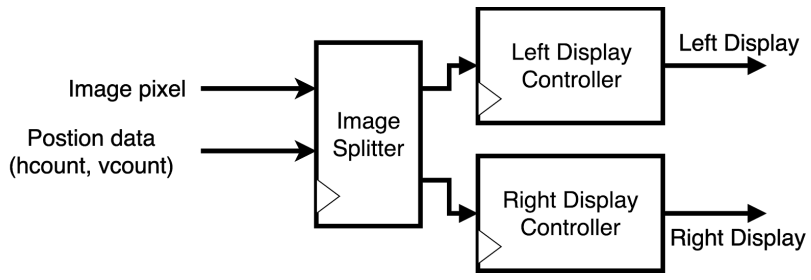
Complexity and Challenges

- Coordinate Converter: This module converts the x and y coordinates in the image what is eventually projected. More information on this process is available in the sources.
- Horizon Generator: This will be the simplest module and the least taxing. It will be a good test to benchmark what frame rate we are capable of, and how jittery the IMU is.
- 360 degree image display: Standard 360 degree images are actually just rectangular jpegs. A common size is 750x375. For memory conservation, we plan on using only grayscale eight bit. That means each image would take 2,250,000 bits, fitting comfortably within the Nexxys' capabilities. We will use COEs to improve the quality for each image
- Landscape generator: This is a stretch goal, but we anticipate the resource drain being the speed of the Nexys. Unfortunately, we don't have a great understanding of how we are limited, but some steps we can take to work with our computation limitations are to drop the fps, generate in monochrome or 4 bit color, and not to hold our generator to very strict standards as far as image wrapping is concerned. As an additional, very stretch add on, we think it would be interesting to generate the landscape based off of music, like the old Windows media player. However, we realize this will require a seperate music processing module and is likely outside of our abilities given our limited time frame. However, should we implement the rest of the project quickly, it is something we will consider.

Task Division

Kendall will take the lead on the angle related aspects of this module, and Claire will focus on the image display/projection aspects.

Image Splitting



The image splitter component will be responsible for taking a color value and coordinates in the initial image and calculating the right positions for the left and right displays given some horizontal offset between the two. This data will then be communicated via SPI to each LCD display. As a first pass, the goal will be to take the incoming image and and divide it vertically, with reasonable overlap to create the appearance of a single image. Further adjustments may be necessary based on screen size.

Targets:

- Minimal: project the same image to two screens.
- Stretch: Introduce binocular overlap, allow for two different images for stereoscopic viewing.

Complexity:

This module will require us to devise a module to communicate with the LCD displays. Because we do not yet have the displays, the specifics of this module are yet to be determined. In the interim, we plan to debug over VGA by displaying two squares the same pixel size as our anticipated screen. Our first pass will simply split the image in two, cutting squares the same pixel size as our screen that change based on the position from the position module. We will then calibrate for screen size, possibly scaling the image for a more natural feel. We will then introduce binocular overlap, widening the FOV. Eventually, for our generated module, we will allow for two separate images to display, to give the sensation of 3d.

Task division:

Claire will be responsible for this module.

Sources

https://vr-lens-lab.com/field-of-view-for-virtual-reality-headsets/
https://kei-studios.com/quick-guide-degrees-of-freedom-virtual-reality-vr/
https://www.roadtovr.com/understanding-binocular-overlap-and-why-its-important-for-vr-headsets/
https://developers.google.com/vr/discover/360-degree-media
https://web.archive.org/web/20180903203919/http://sensics.com/how-binocular-overlap-impacts-horizontal-field-of-view/