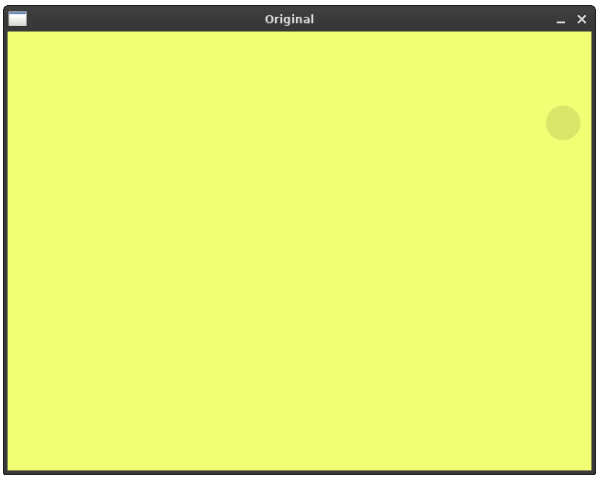
Week 5

Weekly Progress:

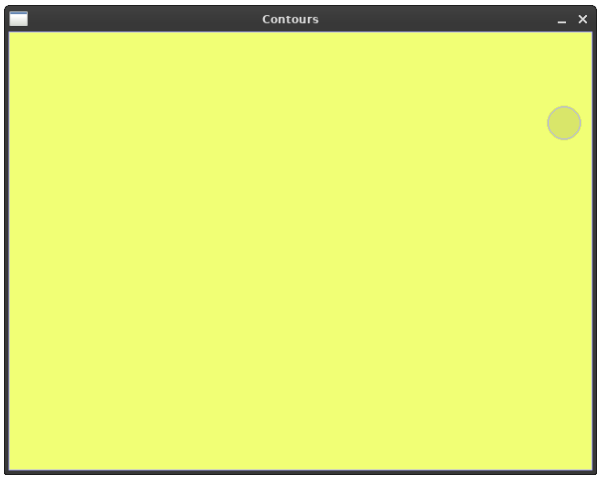
Ball Detection:

This week James and I dove into OpenCV's blob detection algorithm to test on the sample images I created in week 3. After looking at OpenCV's sparse documentation on blob detection, and reading a forum post, we decided to not use blob detection and instead try detecting the contours directly using findContours. Below is a function used to process an image:

```
int detectBlob(Mat im){
  //im: matrix representing photo
  Mat imgray, im_gauss, im_thresh, im_hierarchy;
  vector<vector<Point>> contours;
  cvtColor(im, imgray, COLOR_BGR2GRAY);
  GaussianBlur(imgray, im_gauss, Size(5,5), 0);
  threshold(im_gauss, im_thresh, 212, 255, THRESH_BINARY);
  imshow("Original", im);
  findContours(im_thresh, contours, im_hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE);
  drawContours(im, contours, -1, Scalar(255,172,172));
  imshow("Contours", im);
  waitKey();
  return EXIT_SUCCESS;
}
```



Input Test Image

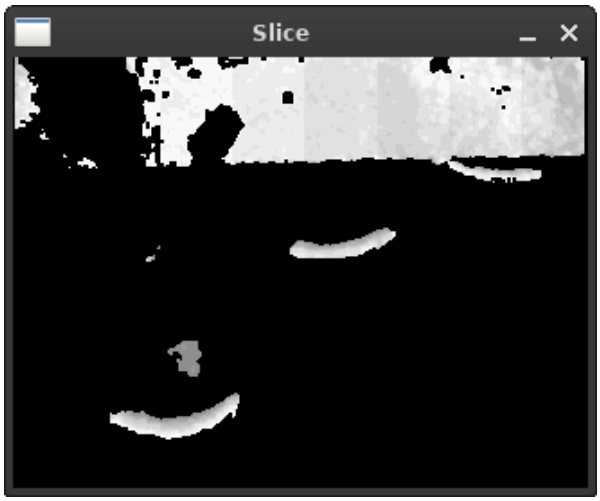


Output Test Image with Contour Drawn

The next step was feeding the OpenNI depth feed into the OpenCV contour algorithm. The OpenCV threshold function only takes in a matrix of 8 bit ints, while the OpenNI feed is a matrix of 16 bit ints. This meant that we would either have to scale the OpenNI feed and loose detail, or only use a 256 size slice of the depth matrix. We chose to use a slice as we only need the depth from right above the table, however, because the table is not perfectly flat, we end up with a difference in depth from one side of the table to the other. To account for this we plan to change our slice from a simple depth cutoff to a 3D plane: each corner of our image will have a depth value, and at each pixel that we will find a new acceptable range for the depth to fall in based of a linear interpolation between all four corners.

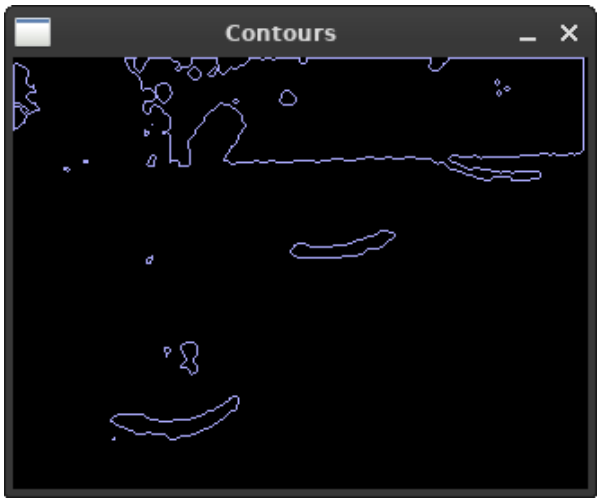
To just get objects on top of the table I adjusted the slice of depth that we sample to be much smaller and raised it to be just above the table. This meant that any object that obstructed the view of the table would not have it's depth recorded and be black surrounded by the white of the table. Below is a demonstration of this technique, the objects on the table and the 3 images that we have while processing the image.

Objects on the Table



Overhead Depth Slice of the Table

Threshold Filter to Clean Depth Image



Contours Drawn of Objects on the Table

The code is getting pretty large, so it is harder to include in this progress report. The code is available to view in these two files on github:

spectator/testBlob.cpp at thinSlice · purdue-RACHEL/spectator

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/purdue-RACHEL/spectator/blob/thinSlice/testBlob.cpp

purdue-RACHEL/
spectator

Code to:

2 Contributors 0 Issues 0 Stars 0 Forks

spectator/cameraBlob.cpp at thinSlice · purdue-RACHEL/spectator

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/purdue-RACHEL/spectator/blob/thinSlice/cameraBlob.cpp

Button Matrix:

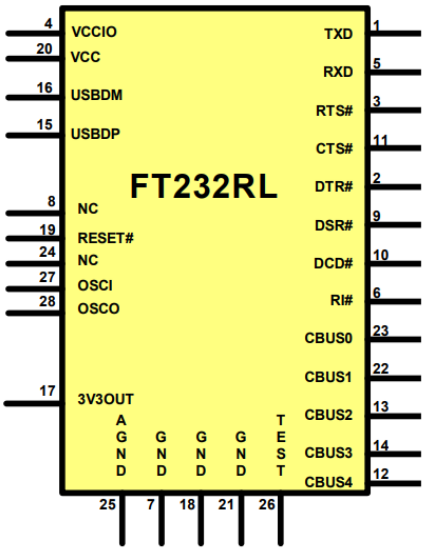
This week we explored the use of a Sparkfun 4x4 button matrix PCB that had pads for diodes to eliminate button ghosting. To test, we soldered the diodes to the PCB and then tested the lines using a power supply and a voltmeter.

However, to acquire the extra hardware to make this a user friendly button matrix (Contact pads, mounting brackets, standoffs) from Sparkfun, would be around 20 dollars, and require us to place an order. It might make more sense for our team to design a button matrix PCB, with diode pads, and through holes for tactile buttons.

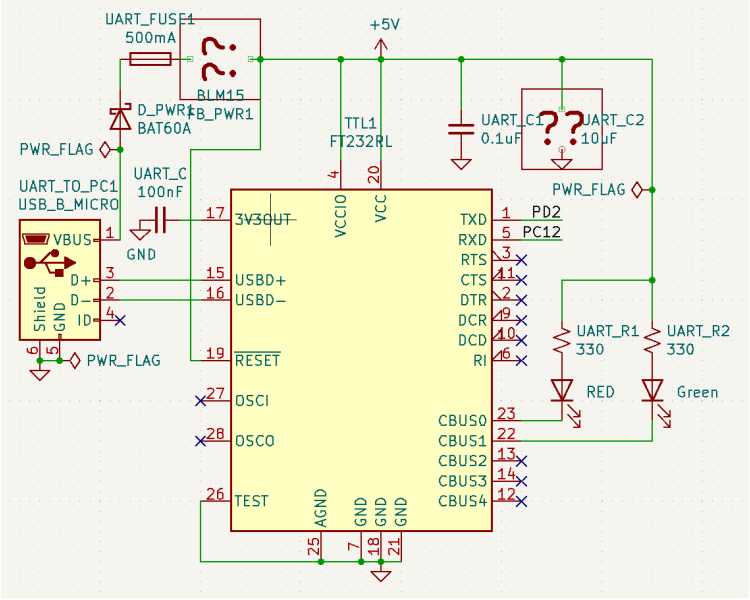
Diodes Soldered to Sparkfun PCB

F232RL PCB Integration:

It is necessary for us to incorporate the FT232RL IC directly into our PCB design. This week, in conjunction with Bartosz, we determined, from the F232RL datasheet how to appropriately interface the IC with the MCU and USB input.



FT232RL Pinout



Integrated FT232RL (kinda)

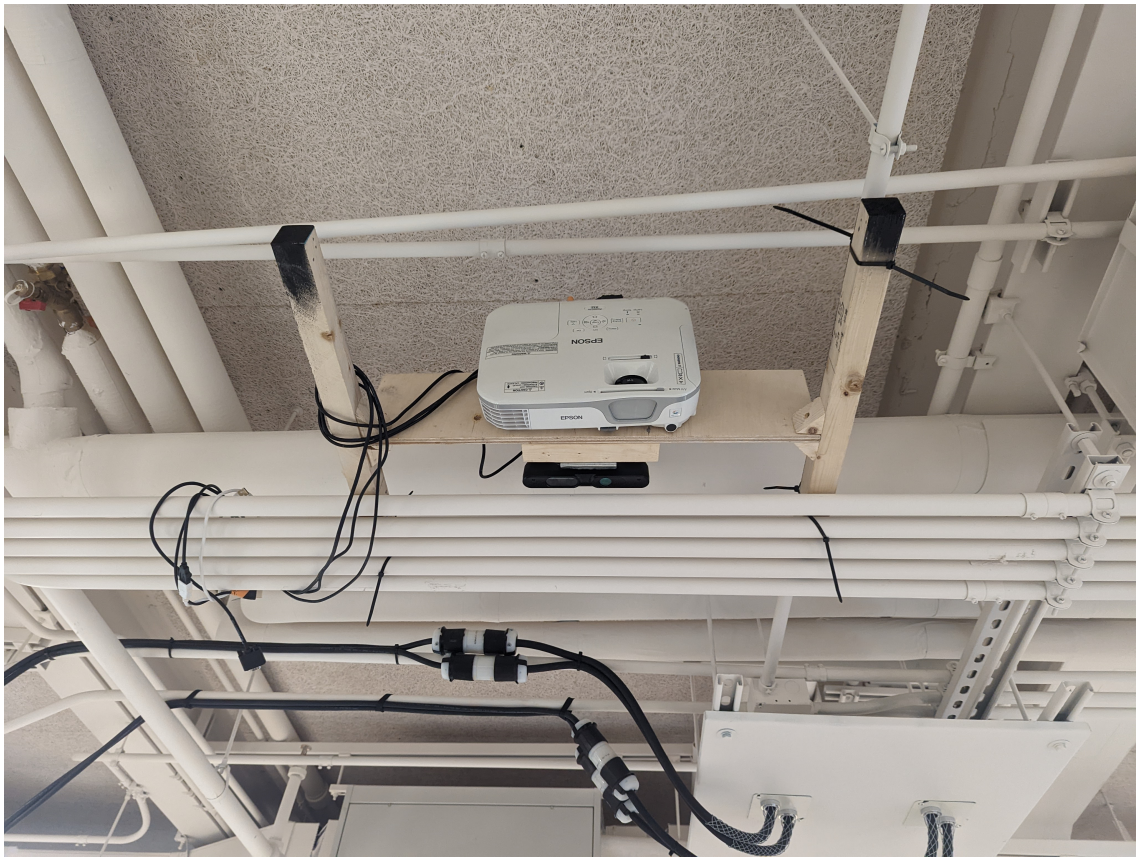
We also modified our PCCSs to not include a buck converter anymore, which leaves us the task of deciding how to power our MCU. As seen in the schematic, the F232RL takes

in a 5V source from USB. Using this source we were able to connect a voltage regulator, to create the 3.3V source needed by the MCU.

Hardware:

Camera and Projector Mount:

This week, we found a place to mount the projector and camera unit that James assembled.



Camera and Projector Mounted in the Ceiling

PCB Components:

I worked with Bartosz to order each of the components for our PCB. This became a challenge when it came to our USB to Serial IC. The FT232RL has no availability on vendors such as DigiKey and Mouser. This leaves us with two possible options:

- Finding an alternative IC and adapting the schematic/PCB design
- Cannibalizing one of our FT232RL breakout boards and yoinking the IC

Calculating View Distance:

To get an idea of the space we would need to display to the whole table I calculated the throw distance of the projector.

Throw ratio range: 1.48-1.77 $\frac{Distance\ from\ Projector}{Width\ of\ Image}$

Distance of projector with image width of 60”: $1.77 * 60 = 106.2$

Next Week:

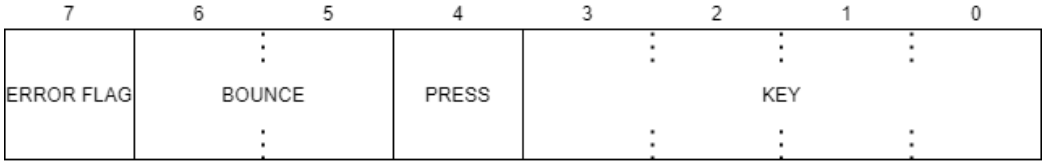
OpenCV:

Next week building of of this week, we can start to filter out contours based on their shapes to eliminate noise and hopefully narrow in on the correct blob to represent the ball. This will involve extracting more data about each contour. Then once this is done hopefully we can assign an x and y value for the current position of the ball.

James has a few ideas for sweeping the entire depth to get a more accurate reading of the ball’s position, so I will also possibly look into this with him.

UART:

Micah Revamped the structure of our UART message, so next week I will work on switching up my switch statement to fit this format.



0x00 - sent on startup / boot
0x01-0x0F - invalid (bit 4 should be set)
0x10-0x1F - key press data
0x20 - 0x3F - right bounce + key press data
0x40 - 0x5F - left bounce + key press data
0x60-0x7F - invalid (bit 7 should be set)
0x80-0xFF - errant reports...
    0b111XXXX - double bounce error
    0x9X - multiple press error (stores latest press)

UART Message Format

Additionally, I may look into incorporating a different USB to UART IC into our schematic due to the availability of the FT232RL

Game Logic:

The pieces are starting to come together to potentially get some meaningful data about the game. We could start piecing together some conditionals to represent the game logic and start building up the state machine that I detailed two weeks ago.

PCB:

Next week Bart will most likely order our first PCB, so I will help him finalize it and check it. Also I will be ordering the components needed this weekend.

Button Matrix:

The button matrix is not as important as the rest of this project so it will kind be on the back burner for a while. If time permits, next week I will look at designing a simple PCB to solder some buttons and diodes to, to create a matrix that doesn't have ghosting. Until then, we will use the button matrix from 362.