Tennis Ball Detection PDR

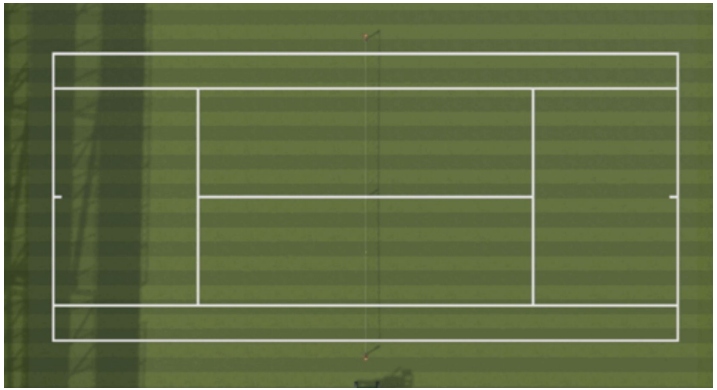
Group 2 - Detect Inc.

Joe Demaria
Andrew Sealing
Connor Ott
Isaac Southwell
Blizzard Finnegan
Wesley Madden

Purpose

During tennis matches, some serves or returns land near the boundary of the field-of-play. The high velocity of the tennis ball makes accurate human determinations of tennis ball position during bounce impossible.

This PDR will detail how we intend to design and implement a computer-vision object detection system to determine the in-bounds validity of a tennis ball bounce on a court.



Approach

Unity

- Capture still images of the tennis ball
- Produce motion of the tennis ball within frame from simulated or black-box function

FPGA

- Process images of the tennis ball using image subtraction

Algorithm (Tennis Ball Calculations)

- Using processed images, determine centroid of the tennis ball
- Determine physical location in 3D space that corresponds with pixel value

Communications Layer

- Create safe data transfer layers for image transfer and data display

Data Display

- Model X, Y, and Z coordinates of each serve on a 3D graph, determine in bounds or out of bounds

Risks

Camera specifications

- 3840 x 2160 resolution could create images too large to quickly process
- Distance from the court diminishes resolution of the tennis ball
- Camera angle creates distortion around the edges of the tennis court

Processing speed

- Large-format images will take longer to process
- Snickerdoodle may not be the best hardware

Major delays

- Issues with workflow or implementation
- Meeting due dates

Team Members and PDR Authority

PDR Authority: Dr. Kaputa

Project Team

Team Member	Name	Primary Role	Secondary Role
1.	Joe Demaria	Algorithm (Tennis Ball Calculations)	FPGA (Image Processing)
2.	Connor Ott	FPGA (Image Processing)	Algorithm (Tennis Ball Calculations)
3.	Andrew Sealing	Unity (World Server)	FPGA (Image Processing)
4.	Blizzard Finnegan	Communications Layer / Data Visualization	Business
5.	Isaac Southwell	Business	Communications Layer / Data Visualization
6.	Wesley Madden	Unity (World Server)	Float

Unity Tasks for PDR

Unity

Task	Est. hrs	Task Ref. ID
Find suitable *physical* camera for virtualization.	2	1.1
Document each of the *physical* camera parameters -- focal length, resolution of output image, color space of output image, maximum frame rate, etc.	1	1.2
Implement *physical* camera attributes in Unity for each of the two virtual cameras.	1	1.3
Determine most reasonable locations for each of the two cameras.	4	1.4
Document virtual camera position and angle relative to the center of the court and pass off to algorithm team.	0.5	1.5
Create a realistic test function that receives an input *t*, and moves the ball across the court by *f(t) = t * some-x-value*, where *f(t)* should be the entire length of the court when t = 2.	2	1.6
Verify test function can receive an input from MATLAB and ball follows motion. Verify images can be captured and saved with the tennis ball in different positions for initial PDR requirement.	1	1.7

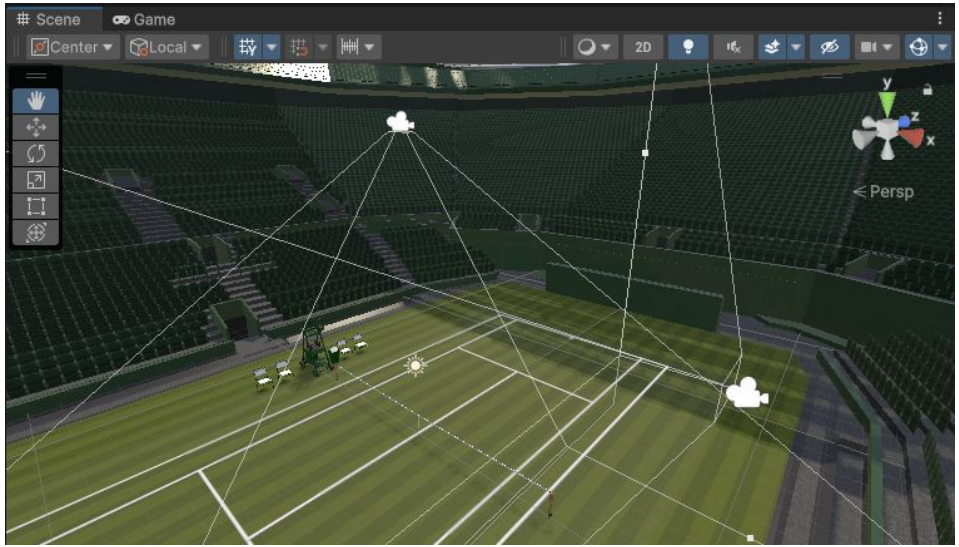
Affordable 4K Resolution

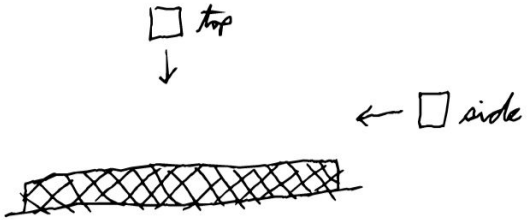
WIFI Transmission	Support
Video Resolution	4K 30FPS; 2.7K 30FPS 1080P 120FPS; 1080P 60FPS 1080P 30FPS; 720P 240FPS 720P 60FPS; 720P 30FPS
Focal Distance	f=7.36mm

$149.99

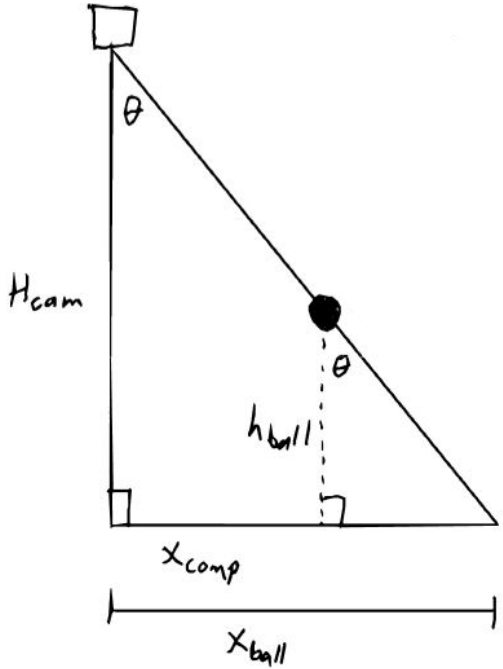
Unity Camera Setup

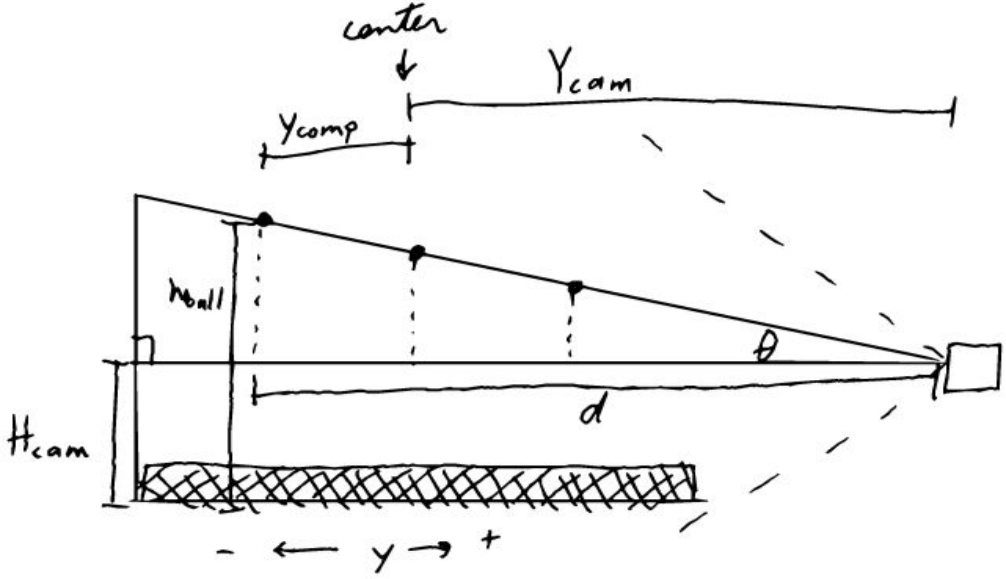
- Decided against stereo vision because of large pixel size at farther distances, which would make height imprecise.
- Briefly considered symmetrical setup with cameras angled in from farther apart, but perspective would have made it unnecessarily difficult.
- Now pretty much decided on top & side setup as the most viable option to have the most precision in all three directions.





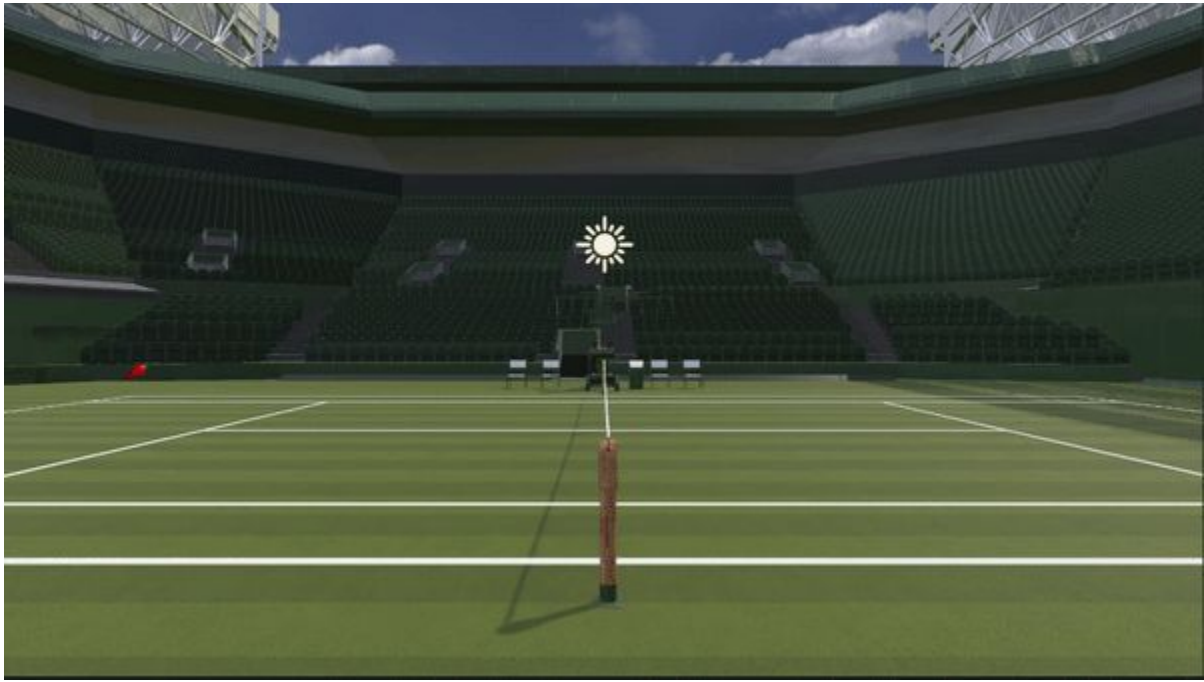
Top & Side Cameras Concept





Simulated Ball Movement

- Scripted Ball movement in Unity
- Including Physics to account for drag, bounce, and friction
- Can record vectors and position before sending to algorithm to cross check
- Can use random vectors to simulate back and forth match

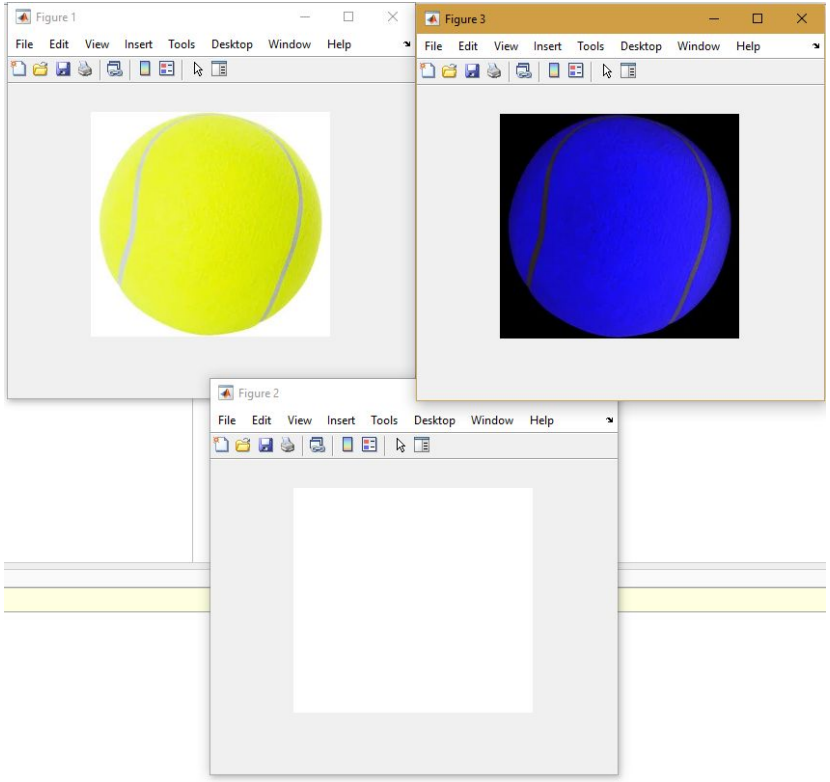


FPGA Tasks for PDR

FPGA

Task	Est. hrs	Task Ref. ID
Feed a saved image into the FPGA using VDMA buffer.	4	3.1
Create VHDL IP for processing an image from RGB to grayscale.	4	3.2
Create VHDL IP for processing an image by XORing with a second frame.	4	3.3
Record the delta(t) of each of the possible image processing paths: img -> rgb_to_grayscale -> XOR -> processed img img -> XOR -> processed img	4	3.4
Showcase delta(t) values for each process as part of the initial PDR requirement. Showcase the final centroid pixel (or closest value).		3.5

FPGA Demonstration



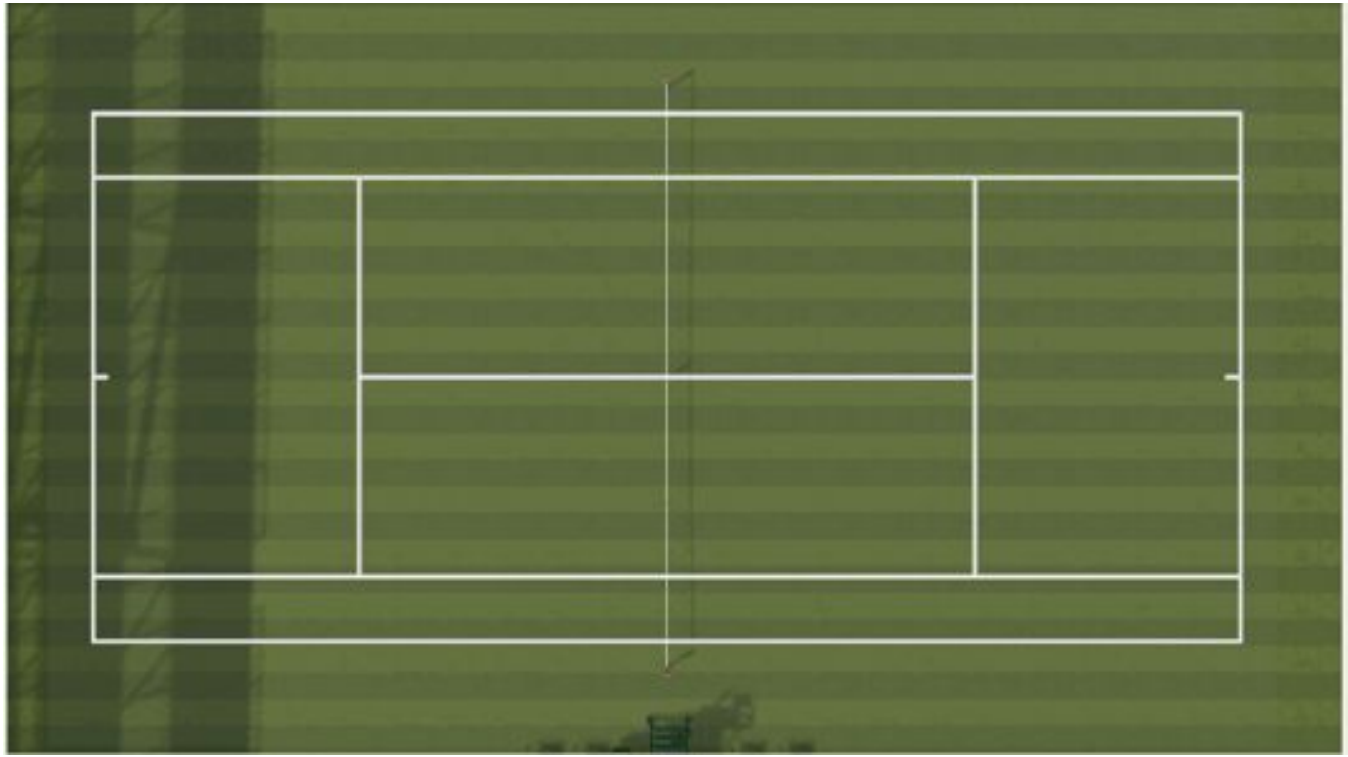
Algorithm Tasks for PDR

Algorithm

Task	Est. hrs	Task Ref. ID
Create test data that mimics what the FPGA will send out; i.e. X and Y positions across each of the two images, as well as diameter of the ball in pixels. Test data should be manually created with saved images from Unity team.	2	4.1
From the left and right image TEST positional data, determine position of the tennis ball. Use the formulas from Lab2/3 .	2	4.2
From the left and right image positional data from the FPGA team, determine position of the tennis ball. Use the formulas from Lab2/3, and possibly using Lab4.	2	4.3
For PDR: Showcase the test calculation for generating X and Y positional data from the pixel values. Showcase the real calculation using positional output from the FPGA.		4.4

Algorithm Demonstration

Left -> Red



Right -> Blue

Communications Layer Tasks for PDR

Communication Layer		
Task	Est. hrs	Task Ref. ID
Prototype the transfer layer between Unity frame capture and the SoC.	8	2.1
Prototype loading image data into the VDMA buffer for FPGA access.	4	2.2
Prototype the REST API between the processor's calculated position data and the web visualization.	8	2.3
For PDR: Showcase some form of data transfer between Unity and the SoC		

Communications Layer Demonstration

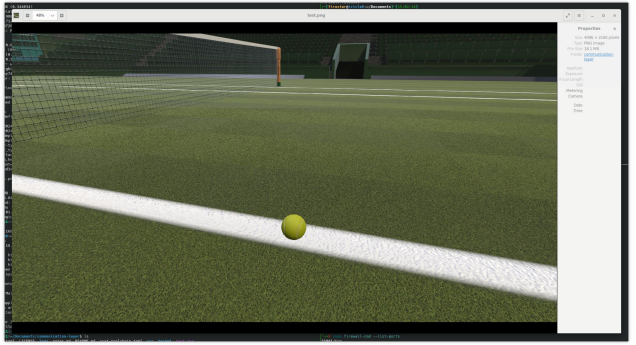
```
bluestar@jetson-tx1:~/Documents/communication-layer$ neofetch
                                           bluestar@jetson-tx1
                                           -------------------
                                           OS: Ubuntu 18.04.6 LTS aarch64
                                           Host: Jetson-TX1
                                           Kernel: 4.9.337-tegra
                                           Uptime: 4 days, 21 hours, 7 mins
                                           Packages: 2362
                                           Shell: bash 4.4.20
                                           Terminal: /dev/pts/0
                                           CPU: (4) @ 1.734GHz
                                           Memory: 454MiB / 3962MiB

bluestar@jetson-tx1:~/Documents/communication-layer$ tree -I target
.
├── Cargo.lock
├── Cargo.toml
├── LICENSE
├── notes.md
├── README.md
├── rust-toolchain.toml
└── src
    └── main.rs

1 directory, 7 files
bluestar@jetson-tx1:~/Documents/communication-layer$ cargo run --release
    Finished release [optimized] target(s) in 0.09s
     Running `target/release/communication-layer`
2024-03-31T10:48:08.497641719-04:00 - [DEBUG, nelink::socket] - Message sent:
NlmsghdrNetlinkMessage { nl_len: 36, nl_type: GetRoute, nl_flags: NlmFFlags(FlagBuffer(1, PhantomData<nelink::consts::NlmF>)), nl_seq: 0, nl_pid: 0, nl_payload: Payload(RtmsgRtmsg { rtm_family: Inet, rtm_dst_len: 0, rtm_src_len: 0, rtm_tos: 0, rtm_table: Unspec, rtm_protocol: Unspec, rtm_scope: Universe, rtm_type: Unspec, rtm_flags: RtmFFlags(FlagBuffer(1, 4096, PhantomData<nelink::consts::RtmF>)), rtattrs: RtBuffer([RtattrRtattr { rta_len: 8, rta_type: Dst, rta_payload: Buffer [] }]) }) }
2024-03-31T10:48:08.499403147-04:00 - [DEBUG, nelink::socket] - Message received: NlmsghdrNetlinkMessage { nl_len: 112, nl_type: Newroute, nl_flags: NlmFFlags(FlagBuffer(0, PhantomData<nelink::consts::NlmF>)), nl_seq: 0, nl_pid: 31189, nl_payload: Payload(RtmsgRtmsg { rtm_family: Inet, rtm_dst_len: 32, rtm_src_len: 0, rtm_tos: 0, rtm_table: Main, rtm_protocol: Unspec, rtm_scope: Universe, rtm_type: Unicast, rtm_flags: RtmFFlags(FlagBuffer(512, PhantomData<nelink::consts::RtmF>)), rtattrs: RtBuffer([RtattrRtattr { rta_len: 8, rta_type: Table, rta_payload: Buffer }, RtattrRtattr { rta_len: 8, rta_type: Dst, rta_payload: Buffer }, RtattrRtattr { rta_len: 8, rta_type: Oif, rta_payload: Buffer }, RtattrRtattr { rta_len: 8, rta_type: Prefsrc, rta_payload: Buffer }, RtattrRtattr { rta_len: 8, rta_type: Gateway, rta_payload: Buffer }, RtattrRtattr { rta_len: 8, rta_type: Uid, rta_payload: Buffer }, RtattrRtattr { rta_len: 36, rta_type: Cacheinfo, rta_payload: Buffer }]) }) }
2024-03-31T10:48:08.499497469-04:00 - [INFO, communication_layer] - aggregating all IPs... This may take several minutes...
2024-03-31T10:48:13.528554672-04:00 - [INFO, communication_layer] - Stream connected to address: Ok(192.168.0.228:55081)
2024-03-31T10:48:13.528682951-04:00 - [DEBUG, communication_layer] - Begin handshake; sending timestamp float...
2024-03-31T10:48:13.528835108-04:00 - [DEBUG, communication_layer] - Timestamp sent!
2024-03-31T10:48:13.529224408-04:00 - [DEBUG, communication_layer] - Continue handshake, read back image packet length...
2024-03-31T10:48:13.530970985-04:00 - [DEBUG, communication_layer] - Image packet length read as 26542080!
2024-03-31T10:48:13.531058359-04:00 - [DEBUG, communication_layer] - Continue handshake, read back image width...
2024-03-31T10:48:13.531332958-04:00 - [DEBUG, communication_layer] - Image width read as 4096!
2024-03-31T10:48:13.531391811-04:00 - [DEBUG, communication_layer] - Continue handshake, read back image height...
2024-03-31T10:48:13.531421603-04:00 - [DEBUG, communication_layer] - Image height read as 2160!
2024-03-31T10:48:13.531472331-04:00 - [DEBUG, communication_layer] - Image bitdepth calculated as 3!
2024-03-31T10:48:13.531551965-04:00 - [INFO, communication_layer] - Start reading...
2024-03-31T10:48:14.032072059-04:00 - [INFO, communication_layer] - Image successfully recieved
2024-03-31T10:48:14.032192526-04:00 - [INFO, communication_layer] - Building image...
2024-03-31T10:48:14.903933191-04:00 - [INFO, communication_layer] - Image built, saving to file...
2024-03-31T10:48:15.057251265-04:00 - [INFO, communication_layer] - image saved!
bluestar@jetson-tx1:~/Documents/communication-layer$ tree -I target
.
├── Cargo.lock
├── Cargo.toml
├── LICENSE
├── logs
│   └── 2024-03-31_10.48.log
├── notes.md
├── README.md
├── rust-toolchain.toml
├── src
│   └── main.rs
└── test.png

2 directories, 9 files
bluestar@jetson-tx1:~/Documents/communication-layer$
```

```
bluestar@jetson-tx1:~/Documents/communication-layer$ identify -verbose test.png
Image: test.png
  Format: PNG (Portable Network Graphics)
  Mime type: image/png
  Class: DirectClass
  Geometry: 4096x2160+0+0
  Units: Undefined
  Type: TrueColor
  Endianess: Undefined
  Colorspace: sRGB
  Depth: 8-bit
  Channel depth:
    red: 8-bit
    green: 8-bit
    blue: 8-bit
  Channel statistics:
    Pixels: 8847360
    Red:
      min: 5 (0.0196078)
      max: 255 (1)
      mean: 93.021 (0.364788)
      standard deviation: 49.2711 (0.19322)
      kurtosis: 2.55342
      skewness: 1.55209
      entropy: 0.889555
    Green:
      min: 6 (0.0235294)
      max: 255 (1)
      mean: 99.7625 (0.391225)
      standard deviation: 47.555 (0.18649)
      kurtosis: 2.07596
      skewness: 1.37919
      entropy: 0.89616
    Blue:
      min: 3 (0.0117647)
      max: 255 (1)
      mean: 70.4825 (0.276088)
      standard deviation: 51.5967 (0.20234)
      kurtosis: 4.39133
      skewness: 2.26241
      entropy: 0.832237
  Image statistics:
    Overall:
      min: 3 (0.0117647)
      max: 255 (1)
      mean: 87.7286 (0.344034)
      standard deviation: 49.502 (0.194125)
      kurtosis: 2.98899
      skewness: 1.71264
      entropy: 0.872651
  Rendering intent: Perceptual
  Gamma: 0.454545
  Chromaticity:
    red primary: (0.64,0.33)
    green primary: (0.3,0.6)
    blue primary: (0.15,0.06)
    white point: (0.3127,0.329)
  Background color: white
  Border color: srgb(223,223,223)
  Matte color: grey74
  Transparent color: black
  Interlace: None
  Intensity: Undefined
  Compose: Over
  Page geometry: 4096x2160+0+0
  Dispose: Undefined
  Iterations: 0
  Compression: Zip
  Orientation: Undefined
  Properties:
    date:create: 2024-03-31T11:07:27-04:00
    date:modify: 2024-03-31T11:07:27-04:00
    png:IHDR.bit-depth-orig: 8
    png:IHDR.bit_depth: 8
    png:IHDR.color-type-orig: 2
    png:IHDR.color_type: 2 (Truecolor)
    png:IHDR.interlace_method: 0 (Not interlaced)
    png:IHDR.width,height: 4096, 2160
    png:sRGB: intent=0 (Perceptual Intent)
```

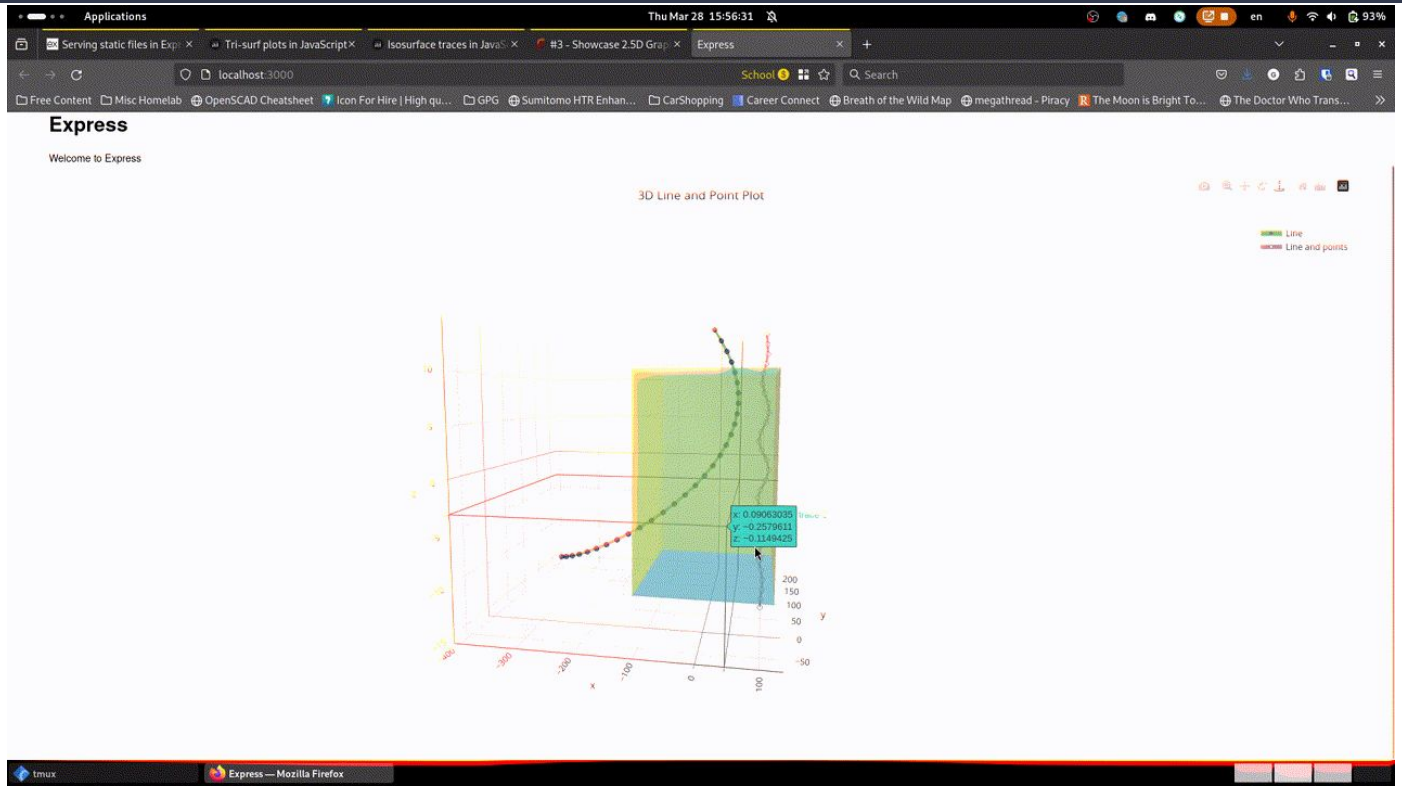


Data Visualization Tasks for PDR

Data Visualization

Task	Est. hrs	Task Ref. ID
Create test data for five samples -- each with an X, Y, and Z position. This can be done by running a t-value through the Unity team's test function.	1	5.1
Create a simple webserver GUI that takes in arguments for X, Y, and Z position.	4	5.2
For PDR: Showcase a graph of each sample's X,Y, and Z position on a 2.5D plot.		5.3

Data Visualization Demonstration



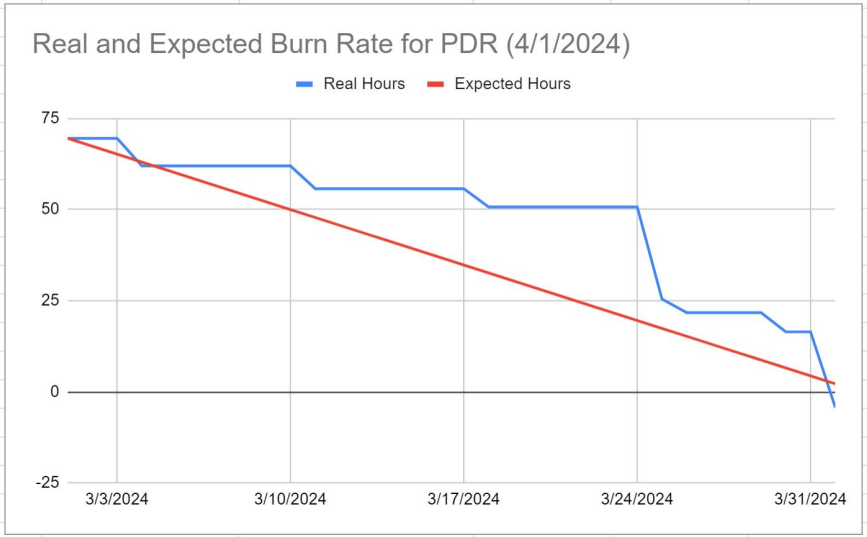
Business Tasks for PDR

Business

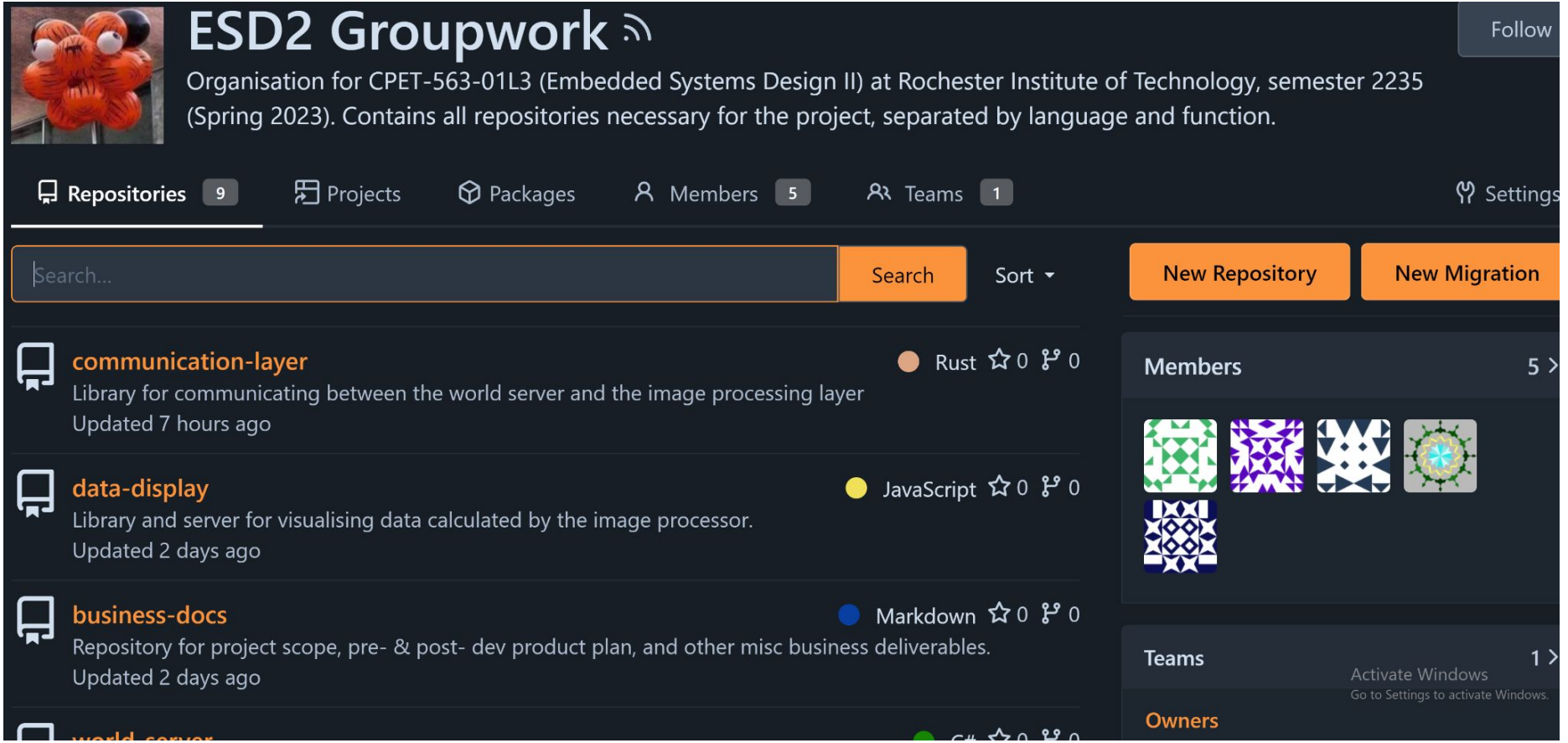
Task	Est. hrs	Task Ref. ID
Create Gantt chart to track critical path and additional slack times.	3	6.1
Keep track of all hours and send weekly updates to all stakeholders.	2	6.2
Create a presentation to highlight each of the completion for PDR in each of the work categories.	5	6.3
For PDR: Create a presentation to highlight each of the PDR requirements and how they were met from each category.		6.4

Individual	Cumulative hours worked
Isaac Southwell	12.25
Blizzard Finnegan	13.07
Joe DeMaria	13.25
Connor Ott	16.5
Andrew Sealing	11.5
Wesley Madden	8.75
Total Hours	75.32

Hours to PDR:	Hours Completed
69.5	75.32



Labor Produced for PDR



Organized Repositories

Executive Summary

Final cost to meet PDR: $7,532 Estimated cost to meet PDR: $6,950

Hours: Week 1	Hours: Week 2	Spring Break	Hours: Week 3	Hours: Week 4
0.75	1.25	0	4	7.25
0.75	1.25	0	2.5	7
0.75	1.25	5	5.5	4
2.25	1.25	0	3	5.75
3.07	1.25	0	6.75	2
-	-	1.5	3.5	3.75
$757.00	$625.00	$650.00	$2,525.00	$2,975.00

From our own development, labor required from PDR to CDR is estimated at 113 hours.

Estimated cost: $11,300

Total for PDR and CDR = $18,532

Hardware and Software decisions

- Use of the Snickerdoodle over Jetson TX1
- Discussions on FPGA implementation
- Google Sheets over Gantt chart
- Plotly for data visualization
- Camera positions

Immediate future for Group 2: Detect Inc.

- Brainstorm and create CDR task list
- Meet to determine dependencies in tasks
- Record hours separate from PDR
- Begin implementation work!