

STREAM AI Team

Dr. Sun's Zoom Link: <https://zoom.us/j/3114058196>

Meetings Fridays @10:00 AM
Hongyue Sun: hongyuesun@uga.edu
Brian Przezdziecki: bprzezdz@buffalo.edu

I've identified the real challenge:

When I want to search for something in my textbook, I merely need to look at the table of contents. This “table of contents” works because:

- I understand the semantic meaning of all the keywords and how they relate to my query in mind.
- The text is rigorously made with an organization of ideas in mind.

On the initial forward pass in constructing the summarization tree, there is no problem. But at what points do branches need to be cut off and relocated?

Tree Cleanup: $O(\text{num_of_nodes}^2)$

Worst Case Num of Nodes: $O(\log(\text{num_of_chunks}))$

Tree Cleanup with respect to num_of_chunks :

Ultimate Unit of Cost is num_api_calls

9/29/23-10/6/23

Langchain docs

Open Interpreter group

GPT3.5 disability

9/22/23-9/29/23

- Implement first rendition of knowledge graph construction on GCP cloud functions
- Create test dataset, and run KG construction on it
- Read Langchain docs and open-interpreter code

Topk & Topp links for Brian

<https://peterchng.com/blog/2023/05/02/token-selection-strategies-top-k-top-p-and-temperature/>

<https://huggingface.co/blog/how-to-generate>

<https://docs.cohere.com/docs/controlling-generation-with-top-k-top-p>

Local LLAMA

I downloaded LLAMA 70B chat and 7B chat locally. 70B chat takes up almost 140 GB. I used Andrej Kaparthy's [LLAMA pure C++ inference](#) repo. Anything 13+ billion parameters doesn't work because of integer flow in pointer arithmetic, which is yet to be fixed, and second, even if it were fixed, this repo is doing float32 inference right now, so it would be fairly unusably slow. The 7B LLAMA model runs at 30 tokens/sec on my Macbook M1 Pro. For this reason, I'll opt to work entirely in the cloud.

Performance metrics can be changed to by token usage and model throughput.

Claude2 vs GPT4

OpenAI <-> Microsoft

Deepmind <-> Google

Anthropic <-> Amazon

xAi <-> Tesla, X

MetaAI <-> Meta

As of now the closest competitor to ChatGPT appears to be Claude2 from Anthropic. It has a max context length of 100,000 tokens, but it hallucinates more.

GPT4 might be dethroned by Gemini, from Deepmind, said to come in December.

Another agent

This project has a much simpler codebase, and runs directly and interacts with your local terminal. It can interact with the chrome browser, file systems and more.

[Open Interpreter](#)

GPT-engineer is not good. The ones of interest so far are:

- Open Interpreter
- Langchain
- AutoGPT

Multimodality in ChatGPT in 2 weeks

GPT-4V (GPT-4 Vision)

This twitter thread shows a bunch of examples of it:

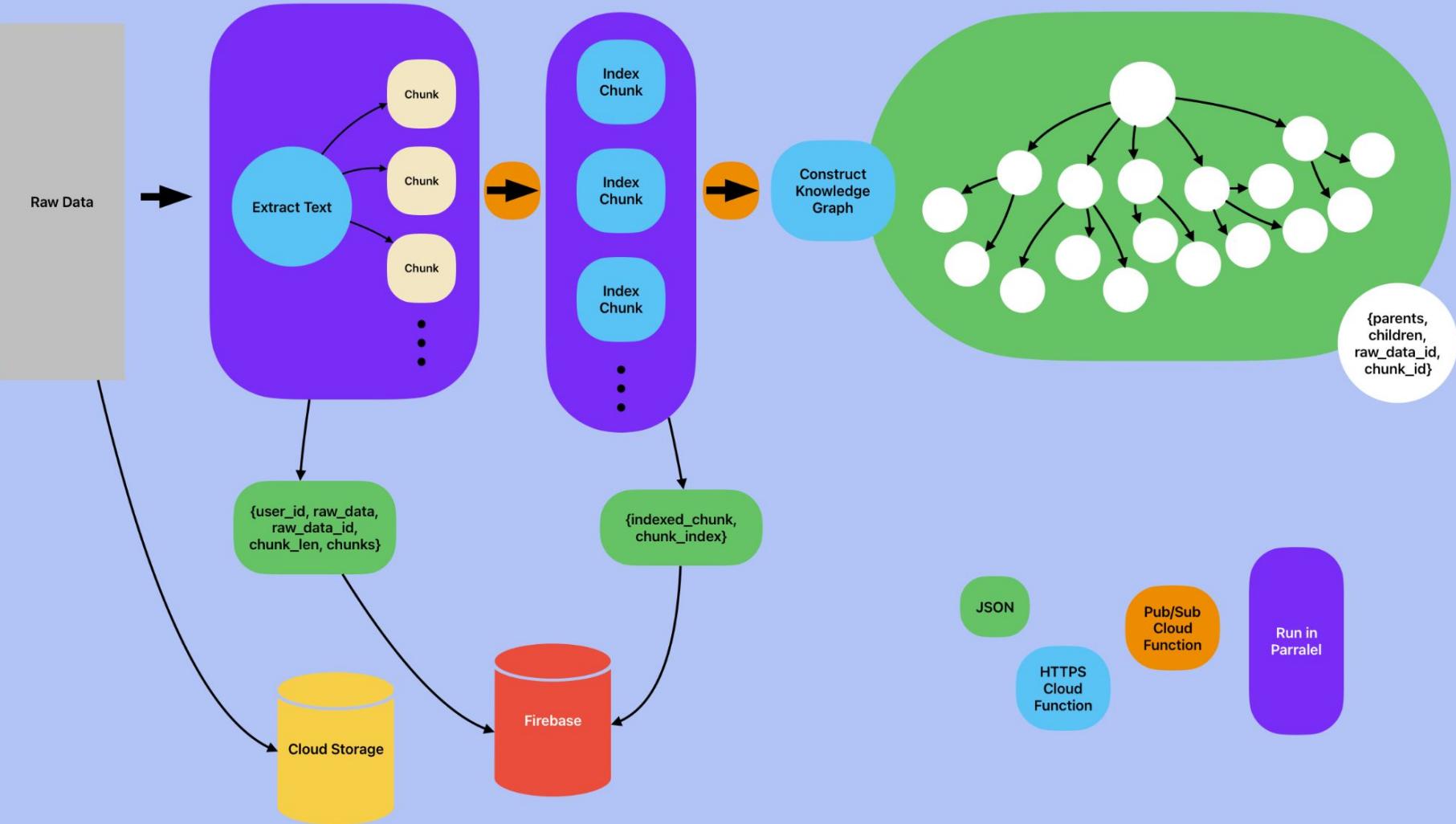
<https://twitter.com/youraimarketer/status/1706461715078975778>

Lots of problems, but cool and shows the direction we are moving in... so fast.

Data

Firestore for persistent storage of graphs as JSONs

Cloud Storage for storing objects and documents as key-value



9/15/23-9/22/23

- Read paper: ‘Consciousness in Artificial Intelligence: Insights from the Science of Consciousness’
- Explore open source GPT projects

Consciousness in Artificial Intelligence

“Our analysis suggests that no current AI systems are conscious, but also suggests that there are no obvious technical barriers to building AI systems which satisfy these indicators.”

“The main alternative to this approach is to use behavioural tests for consciousness, but this method is unreliable because AI systems can be trained to mimic human behaviours while working in very different ways.”

GPT-Engineer

Have GPT create an entire codebase for you using chain of thought, different GPTs and more.

AutoGPT

AutoGPT is an open source project that takes GPT and allows it to act as an agent. Give it a goal, however abstract, and AutoGPT will break the goal into steps, plan, think, reason, criticize and share everything with you. It can go browse the internet, try different solutions and get up to date, and always seeks your approval. It will even argue with you if something is wrong. It can analyze websites, save the data and analyze the results.

[Youtube video talking about what AutoGPT and GPT-Engineer can do.](#)

[It's all open source](#)

BookGPT

[Give GPT a book and have a conversation with it](#)

[The open source code for Book-GPT](#)

Uses Pinecone for vector embeddings and OpenAI for the language model. Both are paid and you need to use your own API key

There are open source versions of these. Although GPT-4 and Pinecone are more powerful

Future Plans

There's been a lot built with GPT already. Before I build anything myself, I want to dive into these open source projects, research papers and learn more about how others are building, what works and what's already been made. This is super cool stuff.

A good thing for me is, given how much time I've spent using the base foundational models, I have a very good understanding of the limitations and capabilities of these models. So time to dive in and see how these people made these GPT agents!

Unfortunately, I am going to need to spend my own money to use the OpenAI API. I hope it won't be too expensive.

Using LLM projects

From Andrej Karpathy, the famous FSD lead at Tesla and currently at OpenAI:

<https://github.com/karpathy/nanoGPT> GPT-2

<https://github.com/karpathy/llama2.c> LLAMA2 in C no dependencies

<https://lambdalabs.com/> - Good priced Nvidia H100 GPUs on cloud

<https://github.com/abetlen/llama-cpp-python> LLAMA2 python no dependencies

<https://github.com/shawwn/llama-dl> LLAMA weight download

<https://github.com/facebookresearch/llama-recipes> LLAMA examples

<https://github.com/cocktailpeanut/dalai> LLAMA/ALPACA Run

<https://github.com/Paitesanshi/LLM-Agent-Survey>

<https://lilianweng.github.io/posts/2023-06-23-agent/>

9/8/23-9/15/23

- KG, Tree of Thoughts, Reasoning, Vector Embeddings
- ~~Migrate pytorch to ONNX~~
- Write onnx run scripts for MobileNet
- Make RaspberryPi branch with new venv and ncnn
- Migrate onnx to ncnn
- ~~Download openev from source on raspberry pi~~
- ~~Download nenn from source~~
- Run system on raspberry pi

Raspberry Pi Optimizations

- Rather than using Pytorch, I will migrate all models to an ONNX format. Once saved as an ONNX model, we can import it into an ARM-friendly C++ framework such as ncnn or MNN. It will speed up our model considerably.
- This means we don't need Pytorch, Torch or Onnxruntime. This also allows us to save SSD and RAM.

Pt -> ONNX & MobileNet ONNX Run with ChatGPT

This could of took hours. I ran into so many errors. With chatgpt, it took 40 minutes:

- Load mobilenet model with changed final layer, add weights, convert to onnx with torch using dummy input with same transforms.
- To run new MobileNet onnx model create an `ort.InferenceSession`, apply transforms to images. In first index of outputs get logits, apply softmax and boom you get the predicted class. So easy with chatgpt.

In this scenario, whenever I ran into a bug I just copied and pasted it to ChatGPT. It's like an instant personal StackOverflow user. At one point, I did have to use my own reasoning to deduce that one bug was from the changed final layer, but once I mentioned it ChatGPT recognized it. Because ChatGPT didn't have full context of my project it didn't know I had changed the final layer.

[My Conversation](#)

OpenAI Stronger Control

LLMs produce unstructured output. False.

OpenAI Function calling allows us to direct the model to invoke certain functions in any manner you wish, and produce a structured output of any format.

Additionally, decrease temperature hyperparameter for consistent and high confidence output answers.

Ensemble methods.

These are techniques to get better control over the model when using it in our system, and remove unpredictable behavior.

New operations

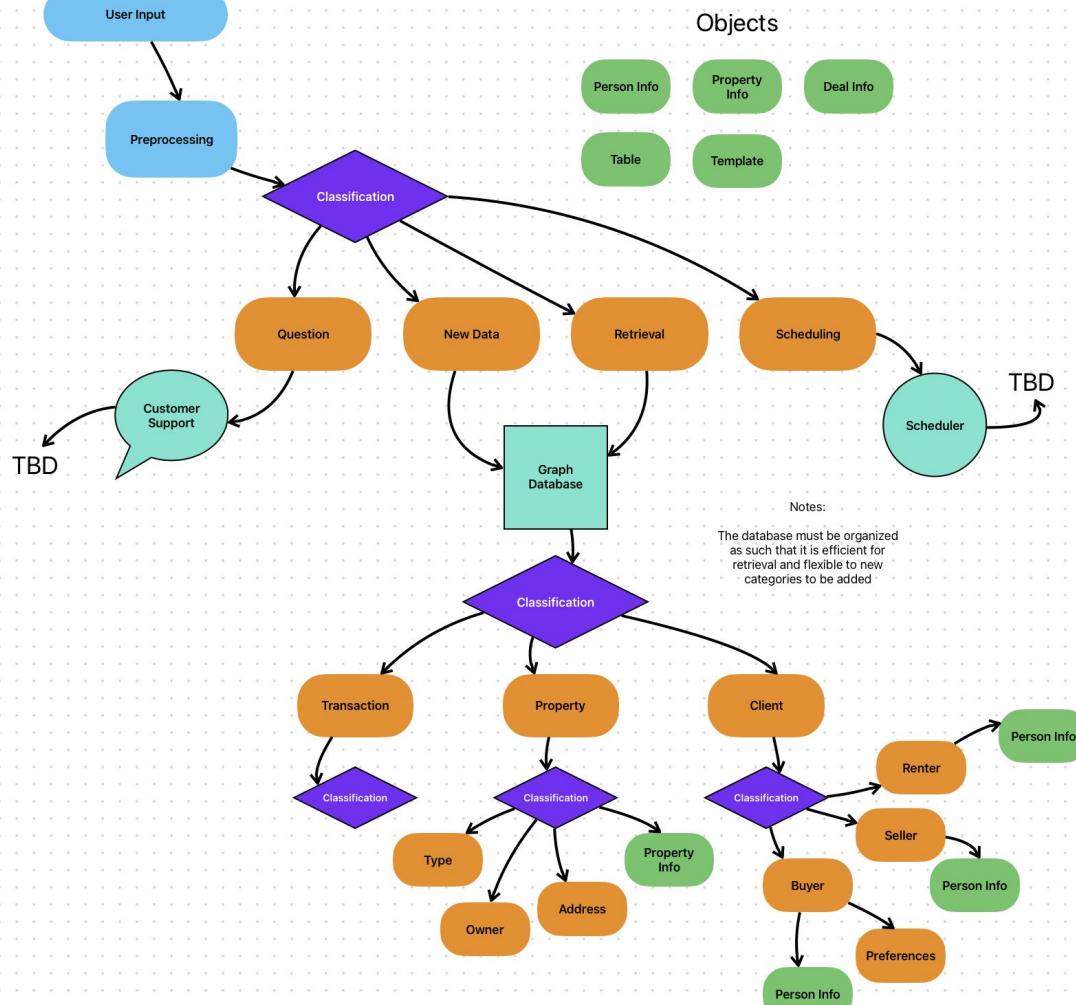
As coders there are things we can use to build systems. We have arithmetic, conditionals loops. We've developed functions and other things we can use like hashing, objects, etc.

LLMs provide an entire new suite of possible operations. Here's some examples:

Classification, comparisons, shallow reasoning, compression, extraction, self prompting and a variety of other simple tasks.

There are now specific things we can add to our system. We can create custom finetuned injection models made for certain tasks.

Example of a system



This is a **very rough** example of what I imagine. Very rough, there are many problems and issues with this that I need to resolve. (I made this a while ago)

But essentially you see the idea of using the 'classification' operation to organize data here. I just wanted to provide an example.

Don't look too deeply into this, this is mostly garbage.

Also, I organized the google drive better:

[LLMKB Project](#)

This stage of the project is very fun :)
(Planning and research)

9/1/23-9/8/23

- ~~Fix initial horizontal bug~~
- ~~Collect speed and RAM metrics~~
- ~~Create long form report~~
- Read LLM papers
- Download libraries from source on raspberry pi
- ~~Get streams on raspberry pi~~
- Run system on raspberry pi

Downloading libraries from source

Onnxruntime, opencv, torch and pytorch all have to be downloaded in a very arduous process on the raspberry pi.

Longer form report

I'm not done with the report yet, as I'll measure more metrics, and need to get the system on the raspberry pi. But here is what I have so far. If you think anything needs more clarification, or should be removed, or any other suggestions, let me know and I will change it.

[Report](#)

LLM Papers

I've gathered more papers of interest. I am in the process of reading all of these. These papers consist of knowledge graphs, reasoning graphs, vector embeddings, factuality, experiments, LLAMA, mathematical reasoning, new reinforcement learning techniques, self teaching and improvements, agents and applications to different financial, industrial and medical industries. I will read all of these, and in particular find which ones are most relevant to this project. I plan to spend lots of time catching up on the most recent and up to date existing work before beginning designing and engineering my own system.

[LLM Papers](#)

Notes

I will add all my notes while reading this paper into this doc. I will talk about the papers I read, my own ideas and rants and more in this doc. Upon reading all papers, learning more and defining the system clearly implementation may begin. However, I want to spend time really preparing for this. I care about this a lot, and really want to also have this for myself throughout my life.

[LLM Notes](#)

Software things I'll need to build

Document and web scrapers

Vector embedding spaces

Series of LLAMA2, GPT3.5turbo and GPT4 models

Database

Computer vision and Language project

The last step of the computer vision project is really to just deploy the system to the raspberry pi, and making some final optimizations to have it run in real time, possibly requiring network pruning, MobileNet fitting to raspberry pi or hardware specific dependency changes. Pytorch has libraries and functions optimized for raspberry pis.

For now, I'm going to begin working on both projects at once - closing and finishing up the computer vision project, and beginning and getting ready for the NLP project. Very exciting.

8/25/23-9/1/23

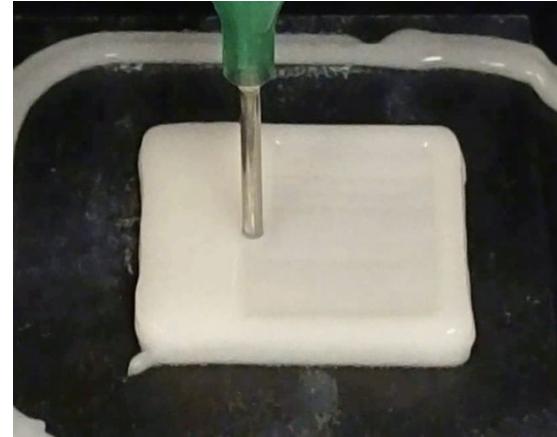
- ~~Refactor threads into modules~~
- ~~Skip impossible predictions (When tip is covering material)~~
- ~~Optimize frame buffer in initialization stage (Can cause zsh hardware errors when RAM overflows)~~
- ~~Define metrics for metric driven research~~
 - Measure diameter method
- ~~Build test runs to measure metrics~~
 - Collect metrics to measure metrics on different videos
- Another idea: LLM for diagnosis/process training, how to integrate text and images?

Skip impossible predictions

This is an example of an impossible prediction, when the tip is blocking the view so we cannot see the recently extruded material. In this scenario, we simply do not attempt to measure or classify the extrusion.

The implementation simply consists of looking at the angle (direction of movement)

If in the future, we can have another camera angle, this can be bypassed.



On the endoscope

I recall in the meeting it was mentioned that using the endoscope might be easier. However, the problem is the endoscopes shaking is unpredictable. The only way to locate the tip would be running YOLO more frequently which is more computationally expensive.

Unless of course, we simply mounted (physically in the real world) the endoscope so it doesn't shake. Shaking of around 10 pixels in any direction is acceptable for classification, but not for diameter measurement, at least with my current method.

So unless shaking can be addressed, the Instant 360 is better, because we don't need to always use YOLO, but can rather try to predict where the top is based off the gcode.

Optimize frame buffer

The frame buffer: When in the initialization phase we need two signals where the horizontal distance is greater than RATIO_INITIALIZATION_MIN_RANGE. I just set it to 5 mm for now. So we need to save the frames in a buffer when they stream in so we can look back on them for the initialization phase. When initialization is done, we can discard the buffer. However, if initialization takes too long, the buffer can take up too much memory.

Thus, simple solution: Keep 2 buffers. One buffer stores frames. Whenever a signal arrives, find the corresponding frame, save to the second buffer, and delete any previous frames. When YOLO is performed on that frame in the second buffer, we can once again discard that frame.

Umm, this might seem confusing, but it's not important to understand, just a specific optimization.

cursor.so said to refactor

Cursor is a code editor like vscode, but with chatgpt built into it. You can share projects of massive sizes with it and get feedback, and it can help you code much better than ChatGPT does because it has context and can see your entire project. On the left is a simple example where Cursor gave me some refactoring suggestions which I will do.

The screenshot shows the Cursor code editor interface. On the left, the file structure of a project named 'STREAM_AI' is visible, including files like '_conda', 'data', 'sidedata', 'src' (which contains 'YOLOv3', 'YOLOv8', 'constants.py', 'drawing_functions.py', 'helper_functions.py', 'inference.py', 'measure_diameter.py', 'preprocessing.py', and 'threads.py'), 'gignore', 'environment.yml', 'main.py', 'my_constants.py', 'notes.txt', 'README.md', and 'test.py'. The 'threads.py' file is currently open in the main editor area, showing a large block of Python code. To the right of the code, a red box highlights the 'Using codebase context' section of the sidebar, which contains a summary of the project's purpose and a list of refactoring suggestions. The 'Summary' section describes the project as monitoring errors in real-time during a 3D printing process. The 'Code Refactoring Suggestions' section lists five points: 1. Modularize the Code, 2. Use Classes, 3. Avoid Global Variables, 4. Improve Naming, 5. Add Comments and Docstrings. Below this, another red box highlights a specific suggestion about refactoring the 'video_thread' function.

```
181 hf.print_text('Signal stream beginning', 'blue')
182 global signal_queue
183 global signals_in_seconds
184 global signals_in_frames
185 global global_signal_index
186
187 signal_list = hf.parse_file(signals_path)
188 video_start_event.wait()
189 for signal in signal_list:
190     signal_time = signal[0] / 1000.0 # Convert milliseconds to seconds
191     time_to_wait = signal_time - (time.time() - start_time)
192     if time_to_wait > 0:
193         # Wait until the appropriate time has passed
194         time.sleep(time_to_wait)
195     # Add the signal to the queue
196     if tracking: signal_queue.put((signal, global_signal_index))
197     signals_in_seconds.append(signal)
198     signal_frame = round((signals_in_seconds[-1][0] / 1000.0) * 30)
199
200     print(f'{global_frame_index}-{signal_frame}')
201     signals_in_frames.append((signal_frame, signal[1]))
202     global_signal_index += 1
203
204 hf.print_text('Signal stream ended', 'red')
205
206 # This thread receives signals and calls the appropriate functions
207 def signal_router():
208     hf.print_text('Signal Router activated', 'blue')
209     global signals_in_seconds
210
211     while True:
212         signal, signal_index = signal_queue.get()
213         signal_time_frame = hf.millis_to_frames(signal[0], 30)
214         threading.Thread(target=temporal_thread, args=(signal_time_frame,), daemon=True).start()
215
216         # This thread runs YOLO and calls the appropriate functions
217         def YOLO_thread(signal_time_frame):
218             if len(screen_predictions) <= signal_time_frame: return
219
220             while True:
221                 buffer = video_buffer.copy()
222                 buffer_start = buffer[0][0]
223                 buffer_end = buffer[-1][0]
224
225                 if signal_time_frame < buffer_start:
226                     hf.print_text(f'Frame not found in buffer', 'blue')
227                     print(f'{buffer[0][0]} to {buffer[-1][0]}\nSignal time frame: {signal_time_frame}')
228
229                 if signal_time_frame > buffer_end:
230                     hf.print_text(f'Frame not found in buffer', 'blue')
231                     print(f'{buffer[0][0]} to {buffer[-1][0]}\nSignal time frame: {signal_time_frame}'
```

Using codebase context
@Codebase so what's this project doing? A summary? How can the code in the threads.py file be made cleaner and easier to read and develop on

USED
Codebase

Summary

This project, named STREAM AI, aims to monitor and correct errors in real-time during a 3D printing process. It uses video, gcode, and sensor data to track the position of the extruder tip, identify printing errors, and make real-time adjustments based on the measured width of the extruded material. The main functionality is implemented in the threads.py file, which contains several threads that simulate video and signal streams, perform analytics, and handle various tasks related to tracking and error correction.

Code Refactoring Suggestions

The threads.py file is quite large and contains many global variables and functions. Here are some suggestions to make it cleaner and easier to read and develop:

- 1. Modularize the Code:** Break down the code into smaller, more manageable modules. Each module should have a specific purpose. For example, you could have separate modules for video processing, signal processing, and error correction.
- 2. Use Classes:** Convert the global variables and related functions into classes. This will encapsulate related data and functions together, making the code easier to understand and maintain. For example, you could create a VideoStream class that handles all the video-related tasks.
- 3. Avoid Global Variables:** Global variables can make the code hard to understand and debug. Try to pass variables as function parameters or class attributes instead.
- 4. Improve Naming:** Some function and variable names are not very descriptive (e.g., hf, g). Use more descriptive names to make the code self-explanatory.
- 5. Add Comments and Docstrings:** While there are some comments in the code, adding more detailed comments and docstrings can help others understand the purpose and functionality of each function and class.

Here's an example of how you could refactor the video_thread function into a class:

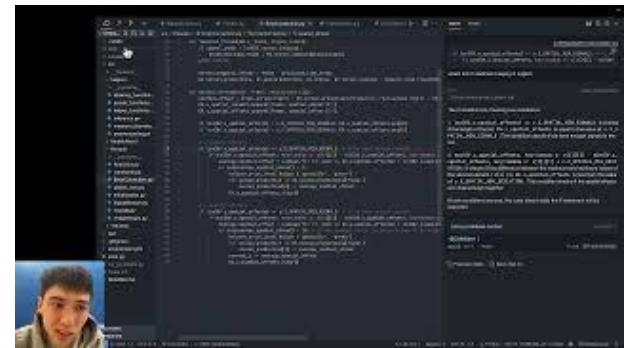
Ln 1, Col 1 Spaces: 4 UTF-8 LF ↵ Python 3.9.17 ('STREAM_AI':conda) Background

Code refactor into classes

As cursor noted, it will be better to break up the threads into each individual module.

Additionally, I want Sahil to work on his own separate branch and be able to merge my changes as they happen. However, for this to work smoothly, I'll need to give him his own file to work in. So I'll just create an additional thread module he can use which makes it easier for both of us, and allows him to develop locally and still get my updates.

By the way, the indexing into a compressed knowledge base performed by cursor.so is similar to what I referred to for the LLM project next.



Note to self: Warnings to fix in the future

Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at the same time. Both libraries are known to be incompatible and this can cause random crashes or deadlocks on Linux when loaded in the same Python program. Using threadpoolctl may cause crashes or deadlocks. For more information and possible workarounds, please see https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

warnings.warn(msg, RuntimeWarning)

[W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.

Need to download onnxruntime from source on raspberry pi:
<https://onnxruntime.ai/docs/tutorials/iot-edge/rasp-pi-cv.html>

Update

Summary of video: (At the end of the video I show how this looks in action)

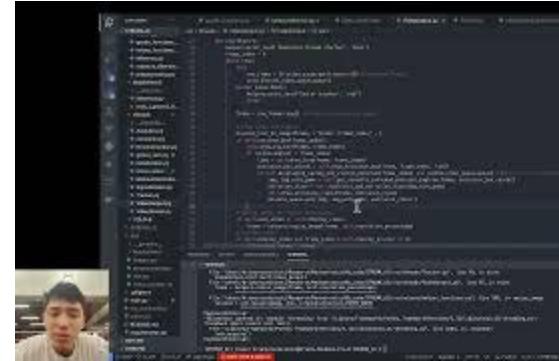
Every 1/30th of a second a frame enters the system. It gets added to the video_queue.

If the time it takes to process frames is longer than 1/30th of a second, a backlog of frames will begin building up. Since frames take up a lot of memory, this eventually leads to a zsh: hardware killed error.

In slide, Note to self: Warnings to fix in the future, there is an NNPack error. This error is causing MobileNet to run slower than it normally should. This causes a backlog.

Thus, the solution is that whenever there is a backlog to not run inference. Only when the video_queue is recent and up to date, will we run inference.

Anyway, in order to make real time adjustments to the tracker we want the extrusion class of the most recently extruded material.



Performance Metrics

- Measure Accuracy:

Label video manually, and compare systems labels.

- Speed:

See fps of classification.

- RAM:

Graph RAM usage throughout time

Time Measurements

Gmms: 0.01 s

Mobilenetvsmall run: 0.05713069438934326

Last time:

mobilenet_v3_small: 0.022805473804473876

LLM Research Papers I will read

FlexKBQA: A Flexible LLM-Powered Framework for Few-Shot Knowledge Base Question Answering

Galactica: A Large Language Model for Science

Opportunities and challenges of ChatGPT for design knowledge management

ChatGPT versus Traditional Question Answering for Knowledge Graphs: Current Status and Future Directions Towards Knowledge Graph Chatbots

ITERATIVE ZERO-SHOT LLM PROMPTING FOR KNOWLEDGE GRAPH CONSTRUCTION

LLM-assisted Knowledge Graph Engineering: Experiments with ChatGPT

Large Language Models Struggle to Learn Long-Tail Knowledge

Raspberry Pi 4 Issue

I wrote the code to receive signals and video from the Instant360. One issue however:

When I try to download the opencv-python package, because it needs to be installed from source it can take anywhere from 30 minutes to multiple hours. I began installation, but the raspberry pi eventually became burning to the touch. Upon googling, I found this could damage the Raspberry Pi.

Apparently, for high load tasks we should use a heat sink or cooling fan for the Raspberry Pi.

8/4-8/11

- Onboard Sahil

Repositories

<https://github.com/BrianP8701/STREAM.AI>

https://github.com/BrianP8701/Anomaly_Classification

Here are the repositories again in their current version. I added the short report and a video presentation (Same as the one from zoom) to the README.

I made setup easy with a conda environment and environment.yml.

Anomaly Classification Coding Interface

I made using the Anomaly Classification repository

Onboarding Sahil

I assigned Sahil his first task. To improve the MobileNet performance alongside the tip tracking algorithm (One of the metrics for the “metric driven research”). I shared with him all the google drives, and recorded a video for him detailing his task and how to use the code:

<https://youtu.be/L-UFGaSTrxw>

7/28-8/4

- Github documentation
- Complete report
- Finish deploying to Raspberry Pi
- Gather metrics and demo videos

7/21-7/28

- Why do small models perform better?
- Finish implementation!!!
- Deploy on Raspberry Pi

You mentioned last meeting my interpretation of the G-Code by the SKR board might be different than what I had implemented. I did further investigating online and it seems that my implementation is correct and the potential reason for the acceleration errors is the following:

Printer Firmware Overrides: The firmware of many 3D printers has built-in limits for acceleration, jerk, and max speed. Even if the G-code requests a certain speed or acceleration, the firmware might cap it to a pre-defined limit. This is especially true if the set values in the firmware are deemed safer or more reliable for the printer's hardware.

G-code Acceleration Commands: While most of the time acceleration is defined in the firmware, some G-code might contain commands to set acceleration for specific sections of the print. These commands can override the default acceleration values. Look out for M204 (set acceleration) and M205 (set jerk) commands in the G-code.

Physical Limitations: Sometimes, even if both the firmware and the G-code allow for a certain acceleration, the physical characteristics of the printer (like mass of moving parts, motor capabilities, belt tension, etc.) might prevent it from reaching those values. For example, a printer with a heavy tool head might not accelerate as quickly as a lighter one, even if the firmware settings are the same.

Segmented Moves: Some slicers break down curves and diagonal moves into many tiny straight-line segments. When the printer processes these segments, it might not achieve the desired acceleration due to the frequent direction changes.

Start and Stop Points: The printer might be decelerating as it approaches the end of a line segment or the start of a new one, especially if there's a change in direction. This can make the effective acceleration appear different from the set value.

Kinematics Problem

We use the 4 kinematic equations to predict the movement of the tip of the tracker solely based off the gcode, which provides coordinates and speeds. At each line of gcode, we predict the time it takes to finish that move. Without getting to deep into details, there is one scenario that cannot be solved analytically, but rather needs to be solved using numerical analysis or iterative methods.

The problem is as follows: Given a positive distance, acceleration, initial speed and final speed: Find the fastest time which you can travel the distance.

Since we know the distance, I thought of the following iterative method:

If curr_speed equals final_speed, we split the distance in half.

If curr_speed is less than final_speed, we accelerate for more than half the distance.

If curr_speed is greater than final_speed, we accelerate for less than half the distance.

What we can do is first try accelerating for the first half of the distance, then decelerating for the second half. Then we iterate the distance by which we accelerate to get closer to the final speed.

We iterate by 10% of the distance 10 times, then 1% of the distance 10 times, then 0.1% of the distance 10 times. With this method, every 10 iterations results in an order of magnitude improvement in accuracy. This is very computationally cheap.

Spatial Error Implemented

1. Run offset
2. When enough average
3. If condition spatial

I tested t



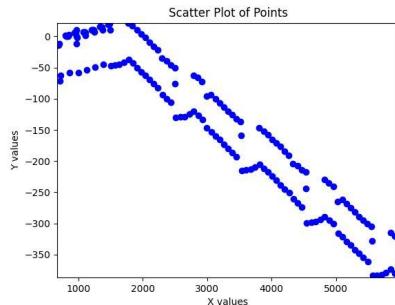
spatial
ose
the
k.

Temporal Graphs

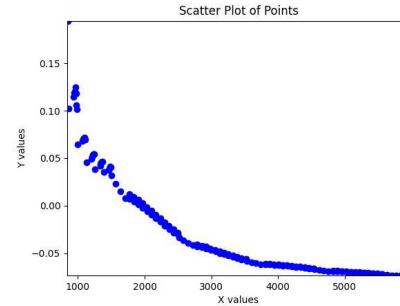
Why are there two lines in the temporal offsets graph? Why does it increase, and then decrease?

Okay, the two lines were an implementation error on my part. However, the “two lines”, or parts of the graph with different slopes is correct. The first phase is that part of the print where it is going around and making some curves, and the second is the making of the cube in the center. This makes it clear that, the acceleration at each of these times are different.

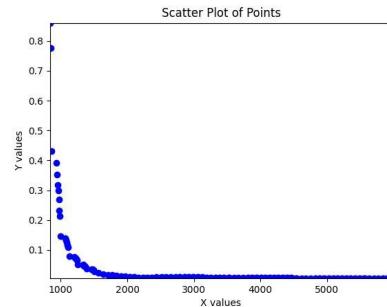
Temporal Offsets (frames)
vs. Time (frames)



Slopes of Temporal Offsets
vs. Time (frames)



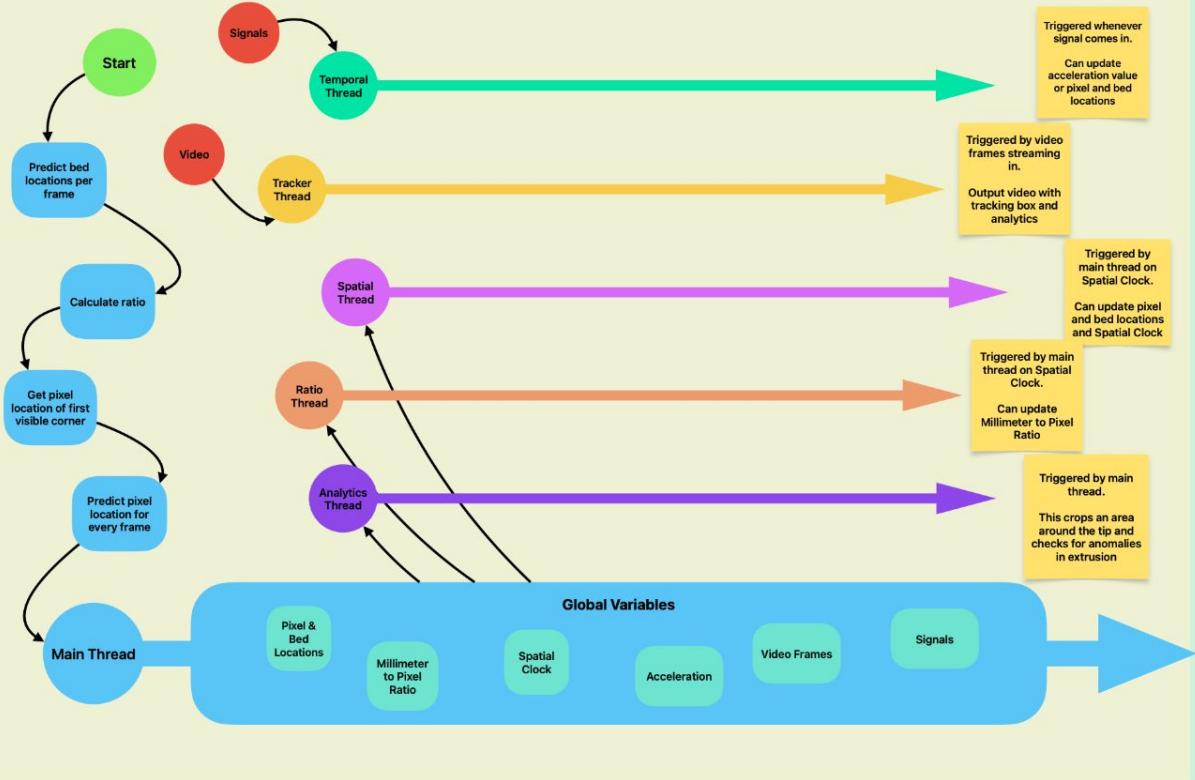
Stdevs of Slopes
vs. Time (frames)



Deploying to Edge

7/14-7/21

- Why do small models perform better?
- Achieve acceptable performance on bubble classification models
- Finish implementation
- Run on video locally
- Contact Dr. Zhou for real signal implementation
- Implement real video stream
- Deploy to edge



This is a more detailed view of how this system will be implemented in code. First the system will perform the initialization and synchronization (The 4 bubbles on the far left.) This gives us all the global variables we need to continue.

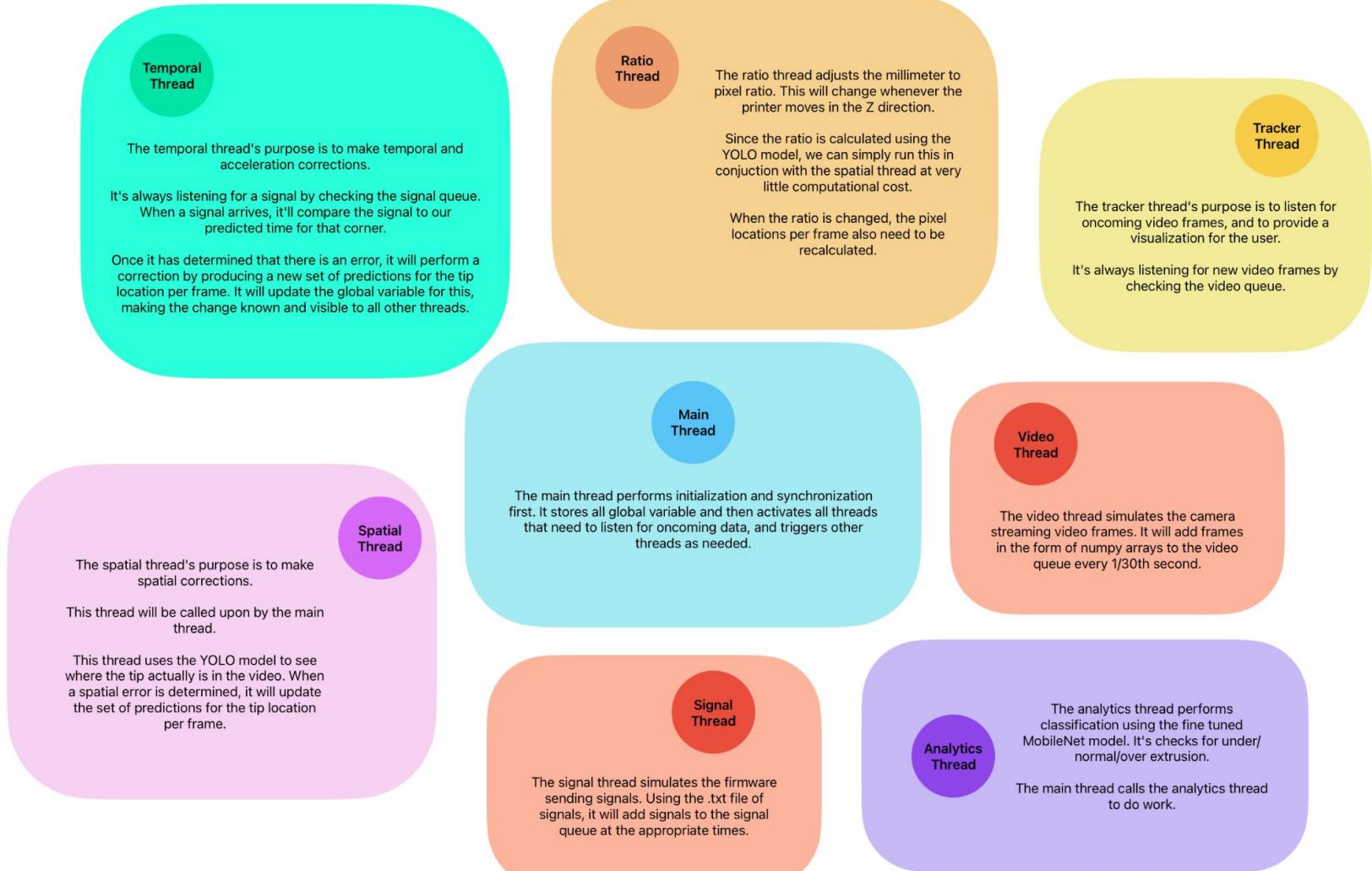
We have 2 different forms of data streaming in:

1. Video frames streaming in every 1/30th of a second.
2. Signals coming in at arbitrary times.

We do analytics on the video frames, use it for visualization, spatial correction and ratio adjustment. We use signals for temporal and acceleration corrections.

Each of these are separate tasks, that have to be run together. Additionally, there needs to be a thread listening for the data as it streams in, while other threads are triggered by conditionals. Thus, my approach is as follows:

We will have one main thread containing all global variables, shared amongst threads. Each thread can listen to their own signals or wait to be called upon by the main thread. When a correction is made, it is seen by all threads. Furthermore, this simplifies and abstracts the code.

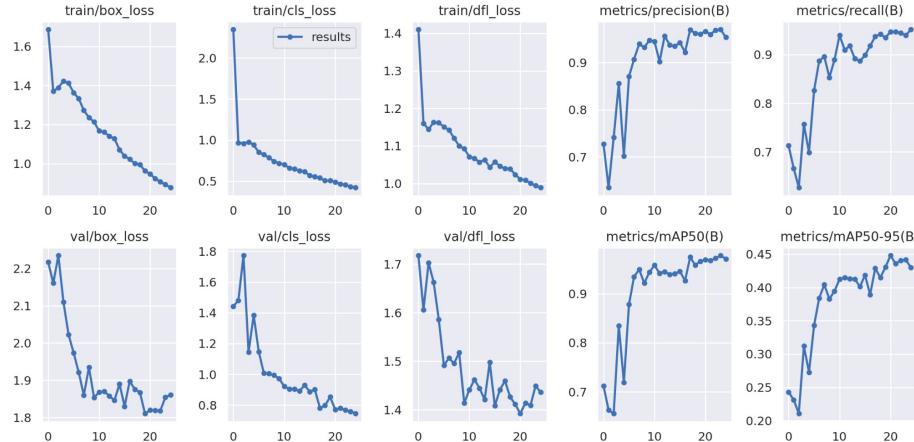


Interesting Details on Implementation

The main thread and all other threads run. I'm not achieving true parallelism, merely concurrency. Any processing of video frames must not hinder the rest of the process, meaning it needs to take less than 1/30th of a second. I found displaying and saving the image could be very computationally expensive, with displaying being worst. I found a simple workaround: reduce resolution or frame rate of display.

Yolo Update

I found that the YOLO model was not accurate enough. The previous tip detection dataset was trained on images from the endoscope camera. I've trained a new model, with 5,385 images (Including old data from previous models, and new ones). A problem was that it sometimes found a tip when there was no tip. So I made sure to add some null images in this dataset.



Bubble Model

Bubble: 184 images | No Bubble: 184 images

Images were cropped at **85x85**. I resized all images to **224x224**.

I also created an additional dataset, applying the **Gaussian Mixture Model + Sensitivity** filter.

For each class I created new images for each class using rotations, adjusting brightness, saturation, contrast, hue, horizontal and vertical flips. I did 40 augmentations per class.

I split each dataset into **75% training** set, **15% validation** set and **10% testing** set.

Training

Given that we have found what parameters worked best, I followed the same:

Batch size = 5

Resize to 224x224

Apply Gaussian Mixture Model Preprocessing

I fine tuned a couple MobileNetv3 models, and immediately got performances above 90%. My question now is, looking at the data, it looks like “bubble” is the same as under extrusion. Before moving forward, is this acceptable?

Yolo run: 0.22236160755157472

7/3-7/14

- Achieved acceptable performance on classification model
- Implemented real time, synchronized, self-correcting parser & tracker.
- Integrated classification model to check for over or under extrusion.
- Prepare to deploy to Raspberry Pi & use TinyYOLO

New Set of Models

Differences:

- More data
- Only did fine tuning
- Only used MobileNet
- Only used images 224x224

Dataset descriptions

Under: 436 images | Normal: 463 images | Over: 446 images

Images were cropped at **85x85**. I resized all images to **224x224**.

I also created an additional dataset, applying the **Gaussian Mixture Model + Sensitivity** filter.

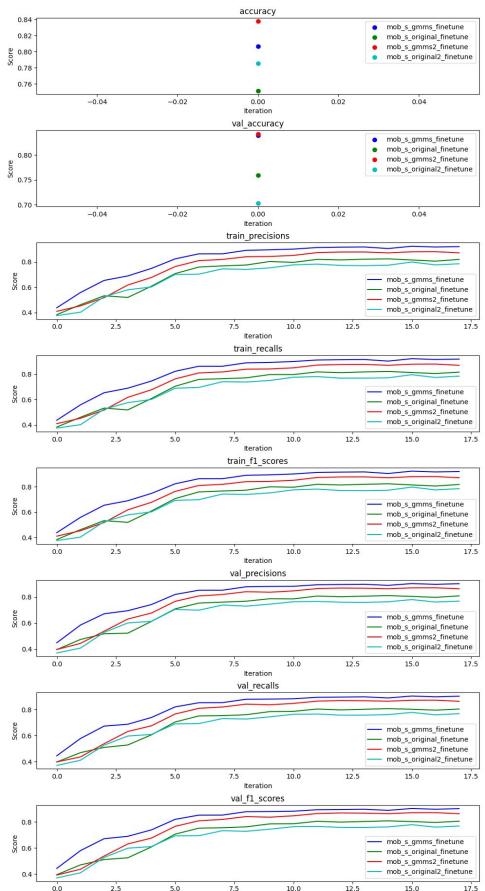
For each class I created new images for each class using rotations, adjusting brightness, saturation, contrast, hue, horizontal and vertical flips. I tried doing 60 augmentations and 150 augmentations per class

I split each dataset into **75% training** set, **15% validation** set and **10% testing** set.

Data for new models

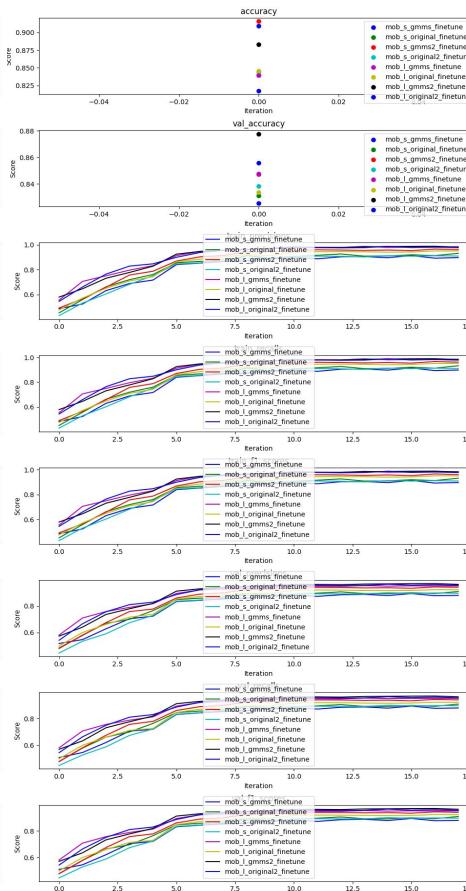
Batch size = 2

Augmentations per class = 60



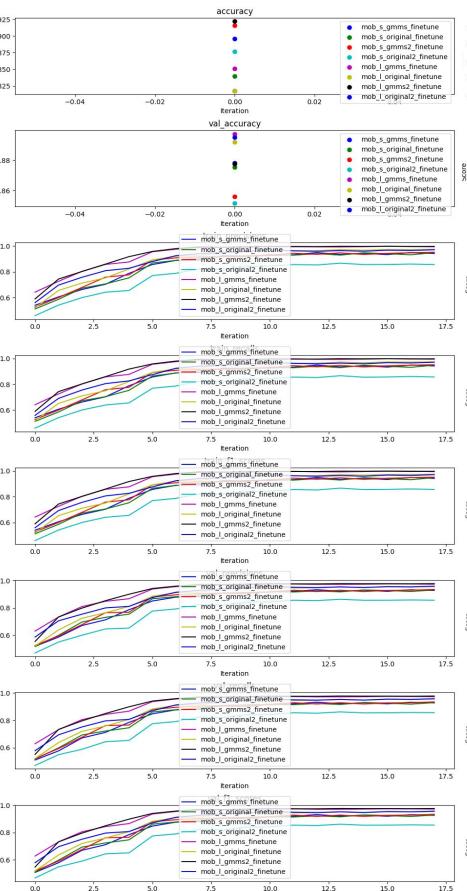
Batch size = 3

Augmentations per class = 150



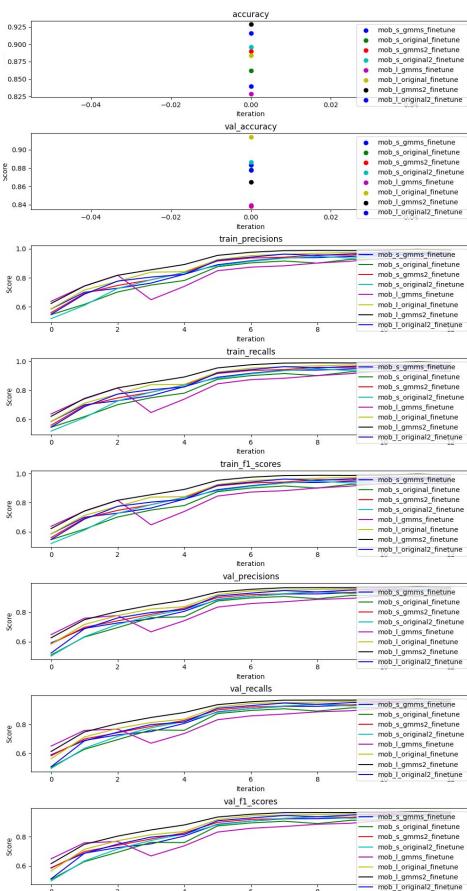
Batch size = 4

Augmentations per class = 60



Batch size = 5

Augmentations per class = 60



6/26-7/3

- Measure inference metrics for each model
- Train next batch of models. Aim for 85% accuracy on test set
- Update error detection and processing to work with corner updates
- ~~Collect all synchronization variables~~
- Automate synchronization
- Identify exact correction equation
- Implement error correction

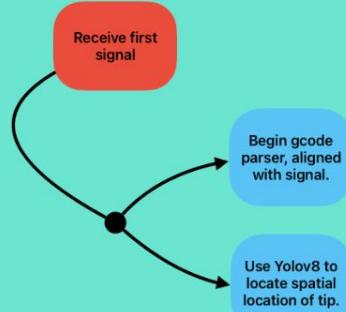
Model Inference Efficiency

I ran each model a 1000 times, and measured the average time in seconds:

- resnet18: 0.05358100652694702
- resnet152: 0.41191160917282105
- mobilenet_v3_small: 0.022805473804473876
- mobilenet_v3_large: 0.05056007623672485
- efficientnet_v2_s: 0.21759283781051636
- efficientnet_v2_l: 0.6030457210540772

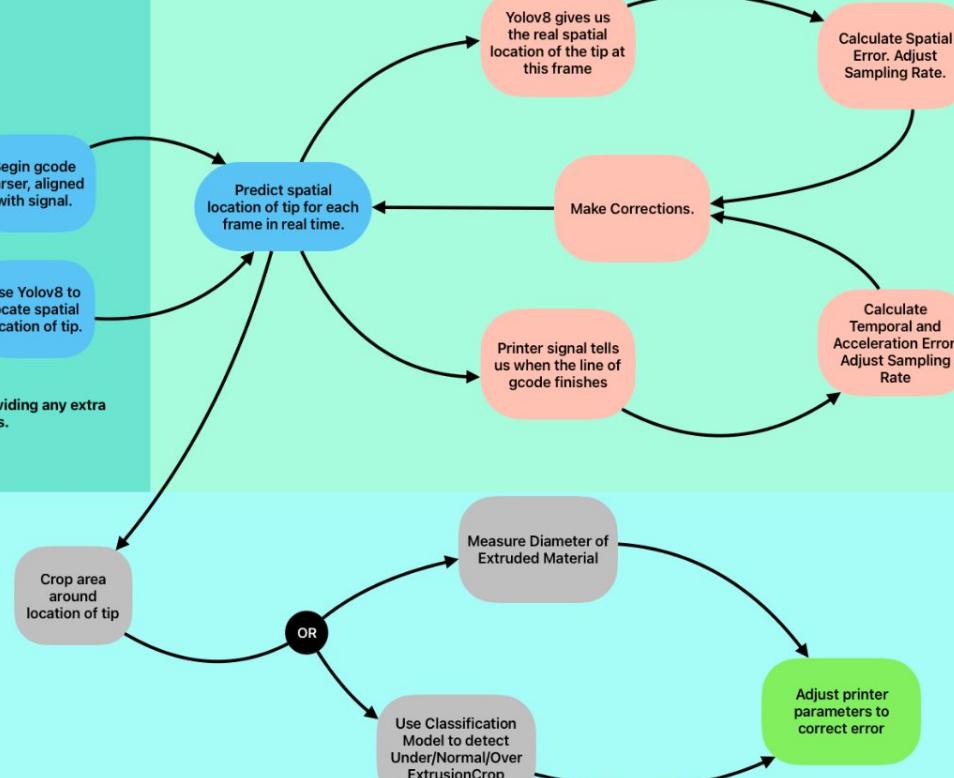
MobileNets are the fastest for inference, whereas EfficientNets take the longest.

Initialization and Synchronization Phase



This allows the user to avoid providing any extra initialization details.

Temporal & Spatial Feedback Loops



'Signals' refer to the signals Dr. Sun introduced last meeting.

Previously, the algorithm required you to specify the initial position of the tip and millimeter to pixel ratio and to manually align the video with the parser.

Now, that is all derived autonomously.

We still use Yolov8. This helps us derive those aforementioned values, and correct spatial errors.

When parsing the gcode, we are still prone to the mysterious acceleration error.

I'm not sure if it will be better to use the classification model or my measure diameter algorithm yet. I'll have to try to make a more robust and efficient measure diameter algorithm

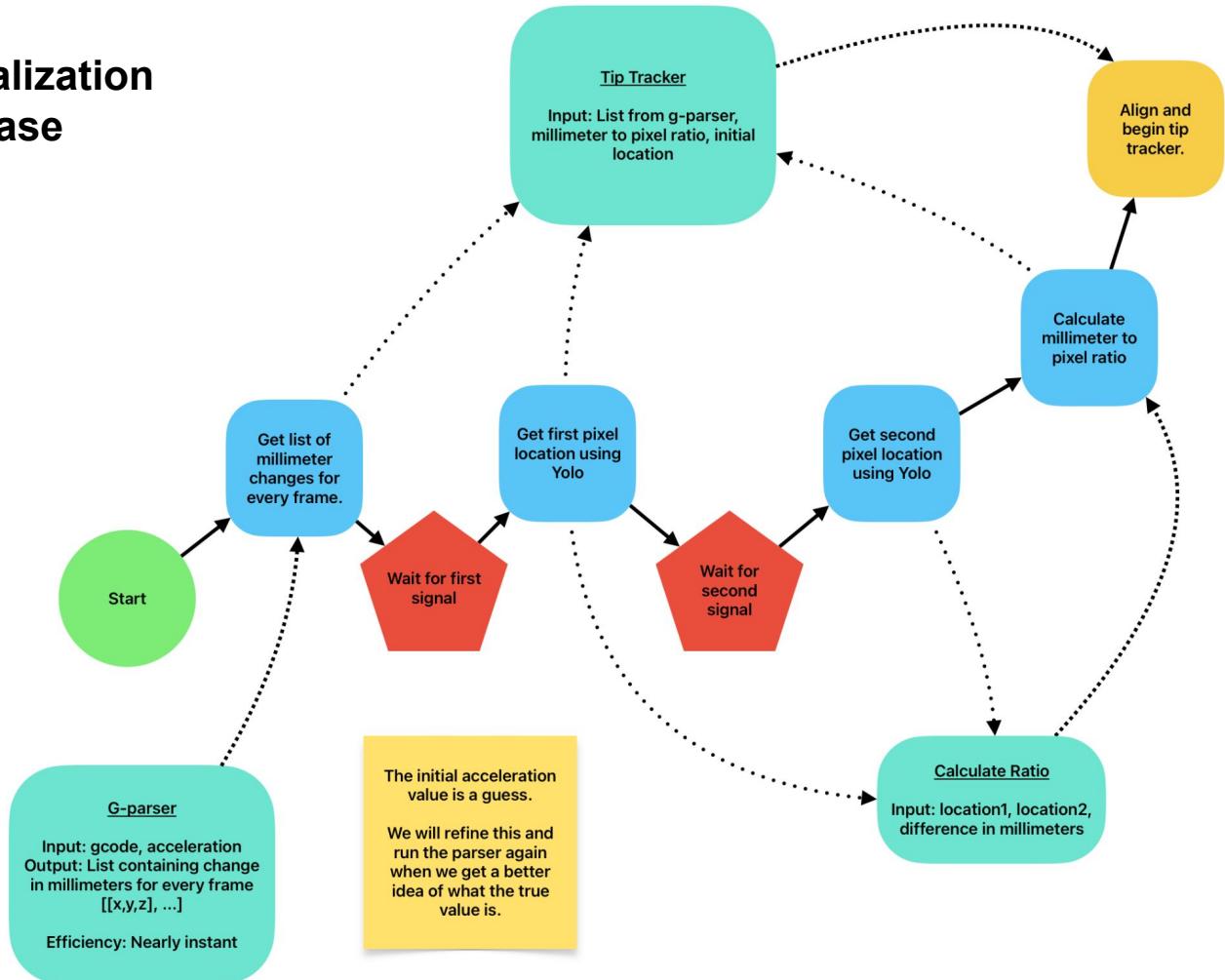
The signals only tell us about temporal and acceleration error. It tells us when the corners happen.

More Detailed View of Initialization & Synchronization Phase

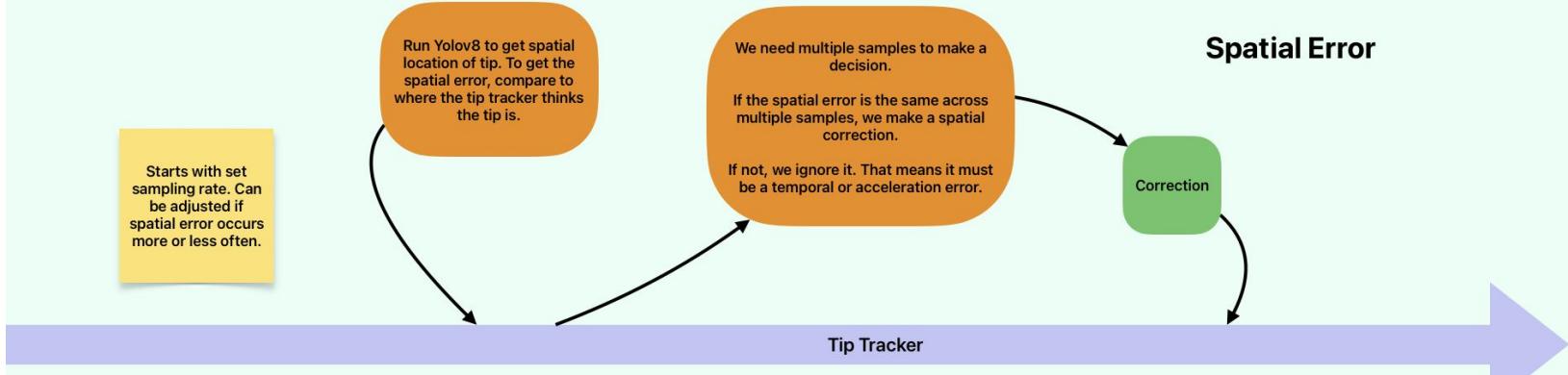
I've finished developing this phase. I've made it so that the video and signals get streamed in real time already to test locally. Some changes still need to be made however:

- Link to Instant360
- Configure to work with actual signal in real time

Once this is done (and I finish coding the next phase) this algorithm will be ready to deploy in real time.

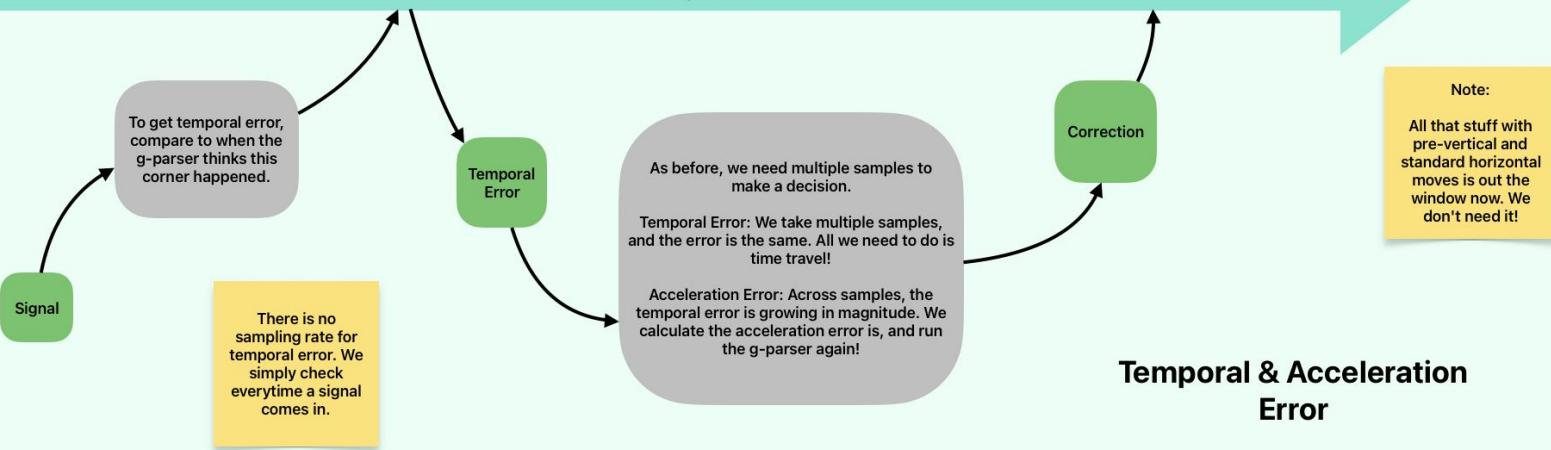


Spatial Error



Tip Tracker

G-parser



Temporal & Acceleration Error

Note:

All that stuff with pre-vertical and standard horizontal moves is out the window now. We don't need it!

New Set of Models

Differences:

- Added test set
- More data
- More data augmentations
- More epochs

See more detailed information on next slide.

Dataset descriptions

Under: 247 images | Normal: 255 images | Over: 246 images

Images were cropped at **50x50**. I made additional datasets with **224x224** images by **resizing**. I want to compare the performance of models trained on 50x50 vs 224x224 sized images.

I also created an additional dataset, taking the 50x50 images and 224x224 images and applying the **Gaussian Mixture Model + Sensitivity** filter.

For each class I created 30 new images for each class using rotations, and adjusting brightness, saturation, contrast and hue. I created 30 new images using horizontal and vertical flips.

I split each dataset into **70% training set, 20% validation set and 10% testing set**.

Training Comparisons

I measured the means for different attributes across my new set of trained models.

Comparing transfer learning to fine tuning:

test: ['**transfer**', 0.5094] vs ['**finetune**', 0.6043]
val: ['**transfer**', 0.5299] vs ['**finetune**', 0.6091]

Fine tuning is clearly better.

Comparing MobileV3Small to MobileV3Large:

test: ['**mob_s**', 0.5726] vs ['**mob_l**', 0.6047]
val: ['**mob_s**', 0.5866] vs ['**mob_l**', 0.6038]
The larger model seems to be better.

Comparing ResNet18 to ResNet152:

test: ['**res18**', 0.4972] vs ['**res152**', 0.5472]
val: ['**res18**', 0.5223] vs ['**res152**', 0.5605]
The larger model seems to be better.

Comparing MobileNet to ResNet:

test: ['**mob**', 0.5887] vs ['**res**', 0.5443]]
val: ['**mob**', 0.5952] vs ['**res**', 0.5658]]
MobileNet might be better.

Data Comparisons

I tried applying (gmms to images sized 50) and
(gmms to images sized 224, then resized to 50).

Comparing 50-50 to 224-50 :

test: ['**50-50**', 0.4800] vs ['**224-50**', 0.5153]

val: ['**50-50**', 0.5151] vs ['**224-50**', 0.4944]

When to apply gmms, does not seem to matter.

Comparing 50x50 to 224x224:

test: ['**50**', 0.5067] vs ['**224**', 0.6340]

val: ['**50**', 0.5104] vs ['**224**', 0.6616]

Images sized 224x224 are clearly better.

Comparing with and without GMMS (50x50):

test: ['**original**', 0.5248] vs ['**GMMS**', 0.4976]

val: ['**original**', 0.5215] vs ['**GMMS**', 0.5048]

GMMS seems to be bad.

Comparing with and without GMMS (224x224):

test: ['**resize**', 0.5997] vs ['**GMMS**', 0.6639]

val: ['**resize**', 0.6397] vs ['**GMMS**', 0.6807]

GMMS seems to be good.

Best Methods

Methods that are clearly better:

1. Use 224x224 images.
2. Use fine tuning.
3. Train for around 15 epochs.

I lost the metrics for this, but I trained a set of models earlier this week. I trained 42 models, each for 40 epochs. It was clear across each model that improvements stagnated at around 8-10 epochs.

Interesting Data Points

Best Model:

mob_s_resize_finetune: [test_accuracy: **0.7925**, val_accuracy: 0.8025]

Worst Model:

res18_gmms_actually50_finetune: [test_accuracy: **0.3868**, val_accuracy: 0.4268]

Moving forward:

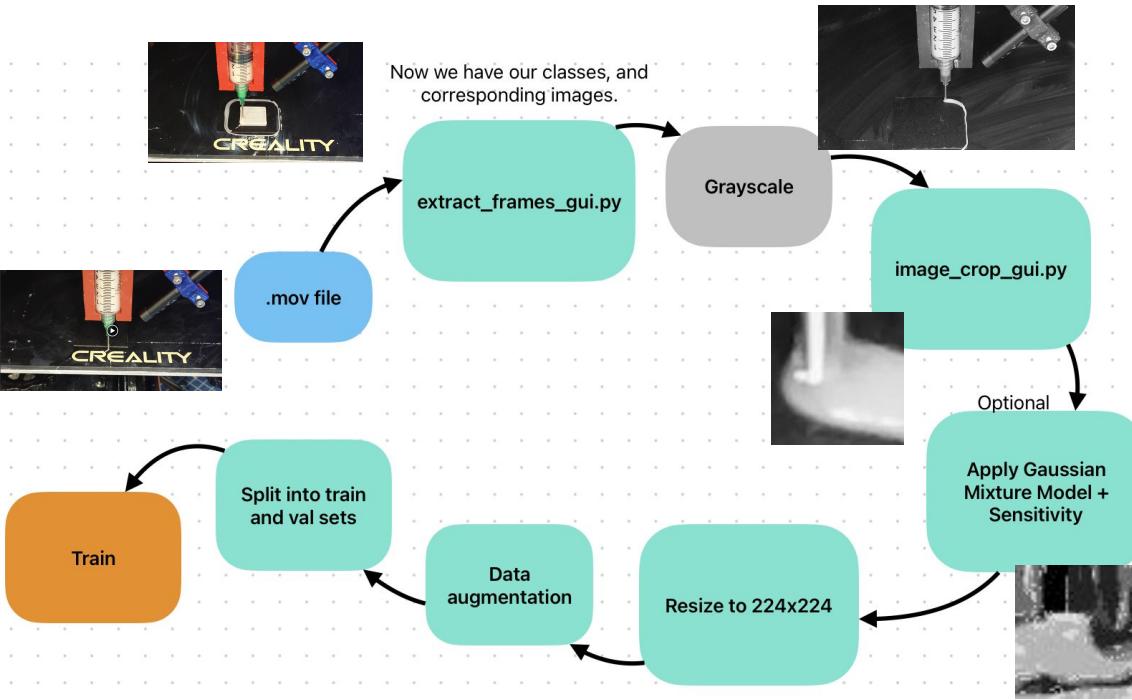
Using what we've found to be the best methods, let's train more models. I think if I just collect some more data we will be able to achieve above 85% accuracy.

See what images the models struggle on. Reshuffle data.

6/19-6/26

- Collect more data
- Make second round of datasets
- Analyze metrics of models
- Use GPU servers to train all models
- Write inference code
- Debug tracker
- Refactor error detection and error processing code

The process of collecting data



I faced some organizational challenges and my code overwrote several images, leading to confusion. As a result, I had to collect the data from scratch, a task that consumed most of the day. I've now more than doubled the entire dataset.

Last time I collected data, I saved every frame from a video as a .jpg file on my SSD, manually identified the frames relevant to each class, and looped through the folders to retrieve these frames. This method was time-consuming and inefficient.

To streamline the process, I developed a new graphical user interface (GUI). This GUI, implemented in the extract_frames_gui.py script, allows for a quicker review of a video and easier extraction of frames associated with each class. This has significantly reduced the time required for data collection.

I also added several new methods: recursively remove duplicate images from folder, reorder and format folder properly and combining multiple folders containing classes into one folder.

extract_frames_gui.py

Here is a screenshot of the description of the tool I implemented, that I mentioned last slide.

```
data_preprocessing > extract_frames_gui.py > ...
1   ...
2   Are you trying to look through a large video and save some individual frames?
3
4   This tool lets you move forwards, and backwards through a video at different speeds and save the frames you want.
5
6   Simply select the path to the video and the paths to the folders you want to save the images to.
7
8       1. Select your video
9
10      2. Select the folder you want to save to. If you have a different amount of folders:
11          Create a variable string for each one
12          Add that variable to classes list
13          In the series of if statements, add a new elif statement for each folder
14
15      3. Run the script
16      4. There are multiple keys you can press to move through the video:
17          'q' - Go forward 1 frame
18          'w' - Go forward 10 frames
19          'e' - Go forward 50 frames
20          'r' - Go forward 150 frames
21
22          'a' - Go backward 1 frame
23          's' - Go backward 10 frames
24          'd' - Go backward 50 frames
25          'f' - Go backward 150 frames
26
27
28      You can change these if you have more or less folders:
29
30          'z' - Save the frame to the first folder
31          'x' - Save the frame to the second folder
32          'c' - Save the frame to the third folder
33
34          'p' - Exit
35
36  import cv2
```

ResNet

ResNet models are deep convolutional neural networks known for "skip connections," allowing gradients to flow through the network directly, mitigating the problem of vanishing gradients, especially in very deep networks. This makes ResNet models capable of being trained deeply. The numbers associated with ResNet models indicate the depth of the network - ResNet18 has 18 layers, while ResNet152 has 152 layers. The deeper the network, the better it can learn complex patterns, but at the cost of increased computational cost and risk of overfitting.

ResNet18: 42.72 MB

ResNet152: 222.7 MB

MobileNet

The MobileNet architecture is made to be small and efficient, with a focus on mobile and embedded vision applications. It introduces depthwise separable convolutions to reduce computational complexity and model size.

With depthwise separable convolutions, rather than applying 3d filters (depth, width, length) to the image, we initially apply “depthwise” 2D matrices in each channel. Thereafter, a 1x1x3 “pointwise” filter combining the channels. This is significantly cheaper (This specific operation is now linear with respect to input size, as opposed to exponential).

MobileNetV3 is very cool, introducing additional enhancements: Neural Architecture Search and NetAdapt algorithm. Instead of the researchers tweaking hyperparameters to find the best architecture, they created an algorithm that would discover the best architecture for them. NetAdapt allows the model to “automatically adapt” to any mobile device, which seems really complicated, and I didn’t look into it (yet).

'Small' and 'Large' refer to versions of the model with fewer or more parameters respectively, again balancing between computational efficiency and model capacity.

MobileNetv3small: 5.93 MB

MobileNetv3large: 16.24 MB

EfficientNet

EfficientNets are made to provide a good trade-off between accuracy and computational efficiency. They introduce a new scaling method that uniformly scales all dimensions of depth(number of layers, width(number of neurons per layer), resolution(size of input) of the network. Instead of scaling these individually, they've found some way to scale them altogether in a better way.

The authors first developed a baseline model called EfficientNet-B0. Then they used their uniform scaling method to systematically increase the depth, width, and resolution of this model, resulting in a family of models from EfficientNet-B1 to EfficientNet-B7, each offering higher accuracy but also more complexity.

The versions like 'v2-small' and 'v2-large' have to do with the scaling factor applied to the base EfficientNet model. Larger versions have more layers or larger layers.

The specifics behind why this work are really interesting, and I want to get deeper into it.

EfficientNetV2small: 77.85 MB

EfficientNetV2large: 449.72 MB

Dataset descriptions

Bubble: 43 images | No Bubble: 50 images

Under: 60 images | Normal: 61 images | Over: 37 images

Images were cropped at **50x50**. I made additional datasets with **224x224** images by **resizing** and **padding**. I want to compare the performance of models trained on resized vs padded images.

I also created an additional dataset, taking the 50x50 images and applying the **Gaussian Mixture Model + Sensitivity** filter.

For each class I created 10 new images for each class using rotations, and adjusting brightness, saturation, contrast and hue.

I split each dataset into **20% validation set and 80% training set**.

Training

In **fine-tuning**, we continue to train all weights on our dataset, as we normally do.

In **transfer learning**, we also start with a pre-trained model. But this time, we keep most of the model exactly the same as it was. We only change the final part that makes the decisions (the "classification layer"). This is the only part of the model that gets to learn from our data. The rest of the model remains unchanged.

We split the training set into **3 batches** each epoch. After going through all the images (one epoch), we backpropagate the error. Then we check the model's performance using the validation set - this doesn't affect the model's learning but lets us see how it's doing. During the training, we keep a copy of the best-performing model according to its scores on the validation set. This approach, called **model checkpointing**, ensures we always have the best version of our model according to how it performs on the validation set.

Measured metrics

Accuracy is measured on the validation set. Errors from the validation set never actually get back propagated to the model. Rather, after each epoch we only choose to continue to train whichever model has performed best on the validation set so far.

The other metrics are measured every epoch, for the train and validation set. This lets us see if the model is overfitting to the training set, while not performing on the validation set.

The following slides follow this format:

1. Small models + Transfer Learning
2. Large models + Transfer Learning
3. Small models + Fine Tuning
4. Large models + Fine Tuning

Each slide will show metrics for all datasets.

Each trained model is labeled as:

model_dataset_finetune
or
model_dataset_transfer

Where model and dataset are abbreviated. I show these abbreviations in the next slide.

Model	Model abbreviation
ResNet18	res18
ResNet152	res152
EfficientNetv2small	eff_s
EfficientNetv2large	eff_l
MobileNetv3small	mob_s
MobileNetv3large	mob_l

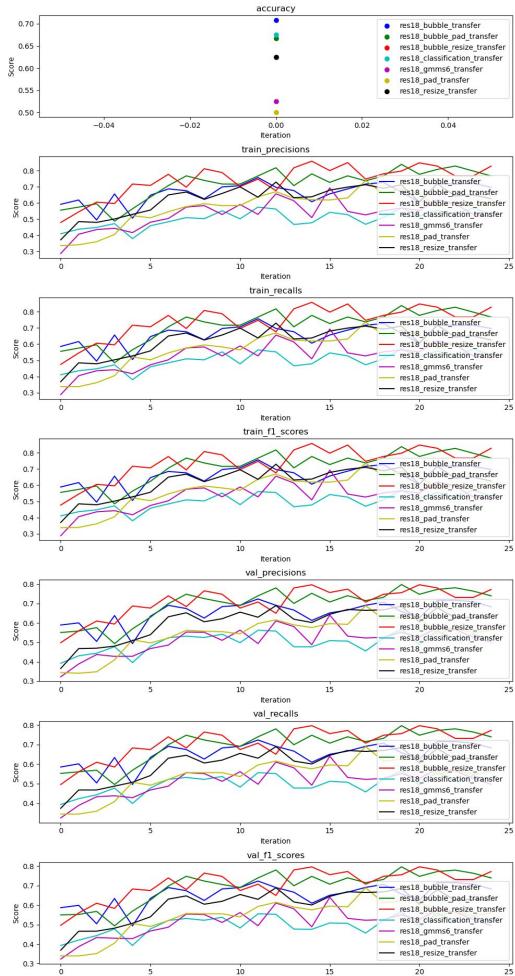
I trained each model with each dataset, once with fine tuning and once with transfer learning.

A total of 84 models.

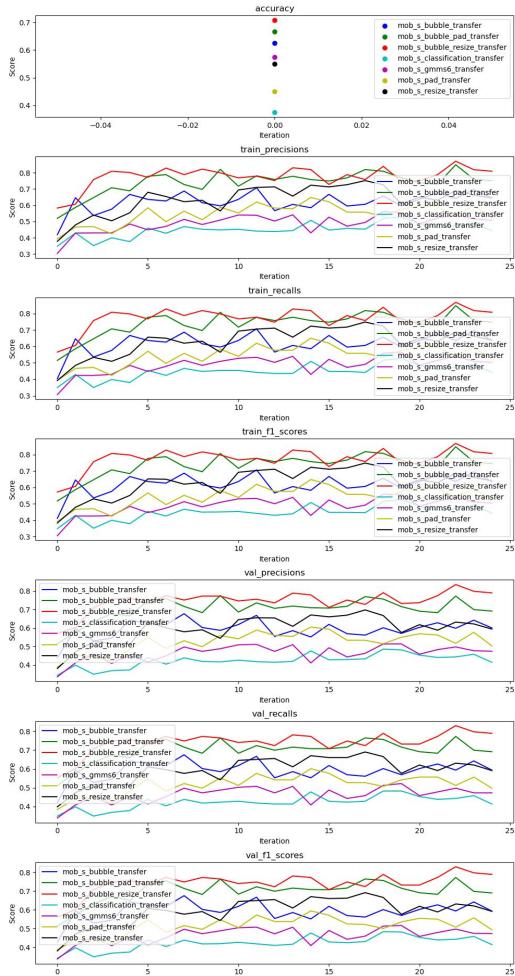
Dataset Abbreviation	Dataset Description
bubble	Bubble/No Bubble Dataset, in its original 50x50 size.
bubble_pad	Bubble/No Bubble Dataset, but with padding to make it 224x224
bubble_resize	Bubble/No Bubble Dataset, but resized to 224x224
classification	Under/Normal/Over Dataset, in its original 50x50 size.
pad	Under/Normal/Over Dataset, but with padding to make it 224x224
resize	Under/Normal/Over Dataset, but resized to 224x224
gmms6	Under/Normal/Over Dataset in its 50x50 size, but with Gaussian Mixture Model + Sensitivity filter.

Smaller sized models, trained using transfer learning

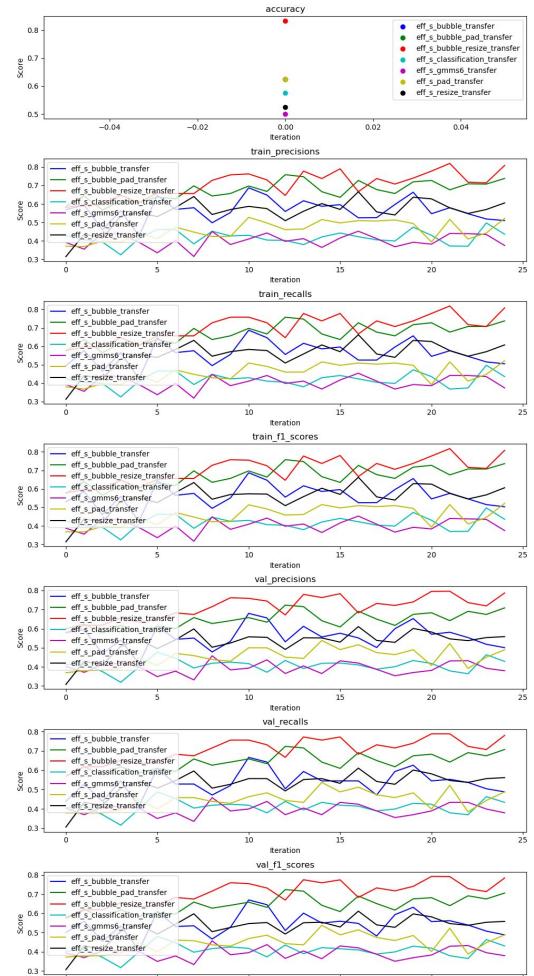
Resnet18



MobileNet Small

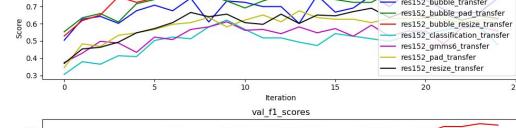
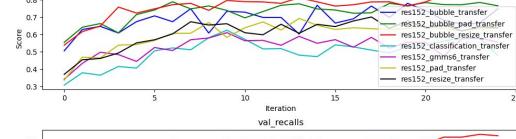
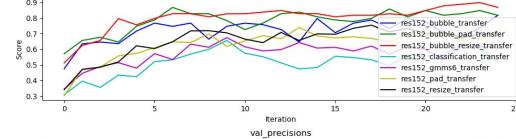
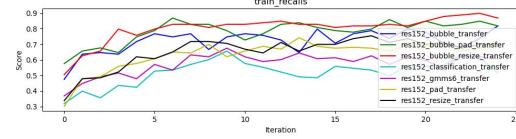
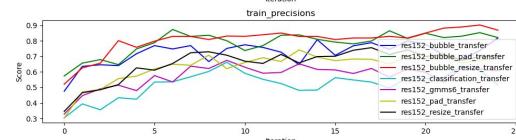
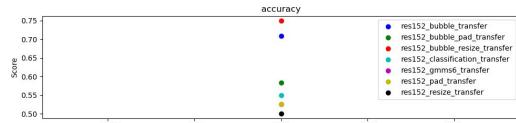


EfficientNet Small

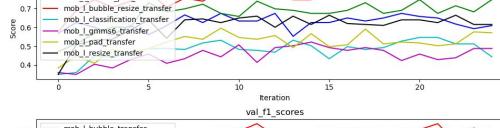
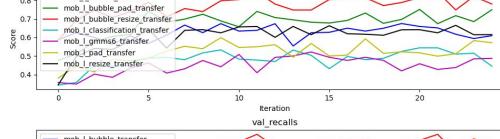
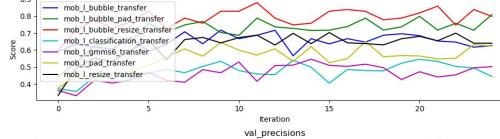
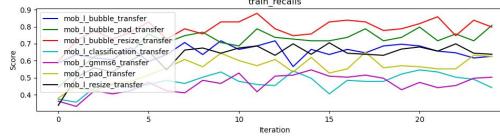
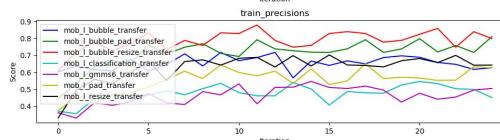
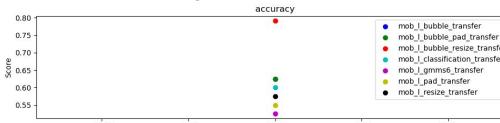


Largest sized models, trained using **transfer learning**

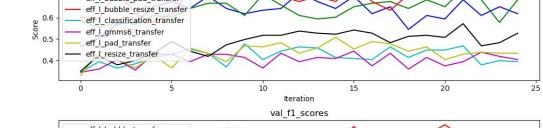
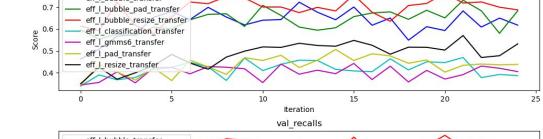
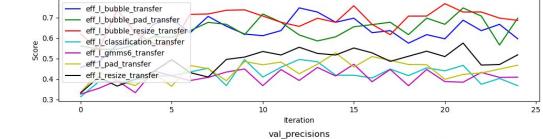
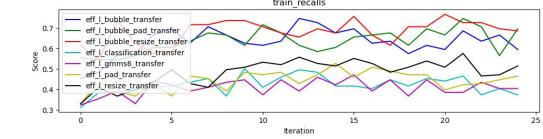
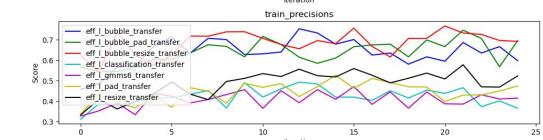
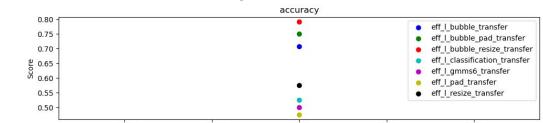
Resnet152



MobileNet Large

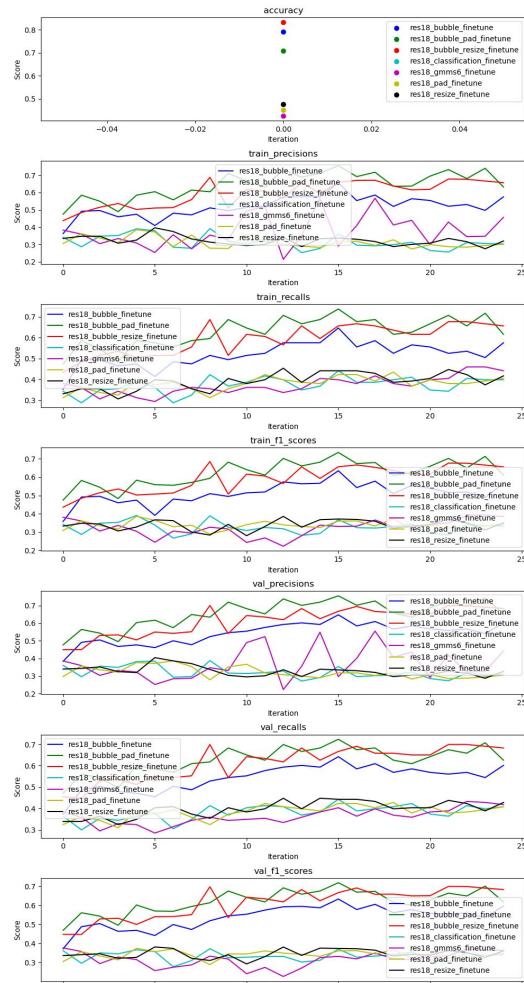


EfficientNet Large

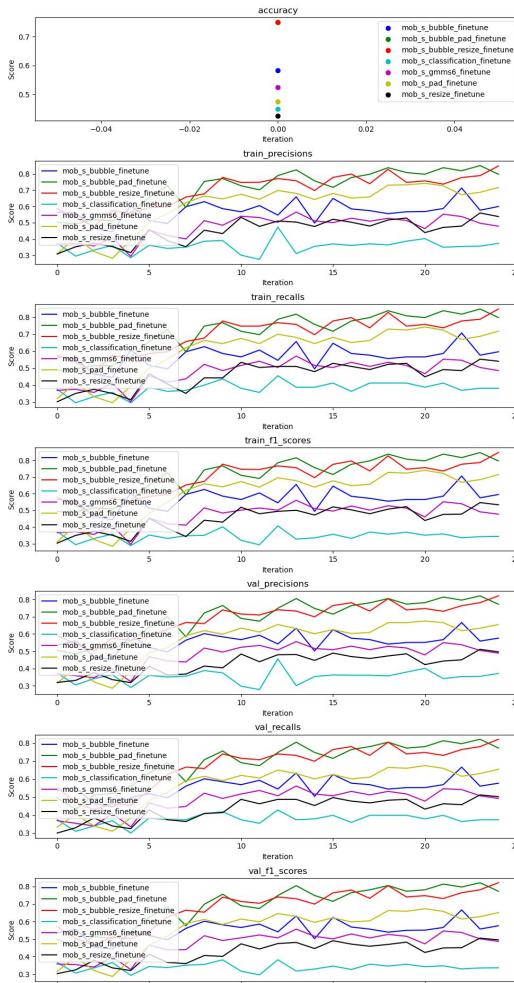


Smaller sized models, trained using fine tuning

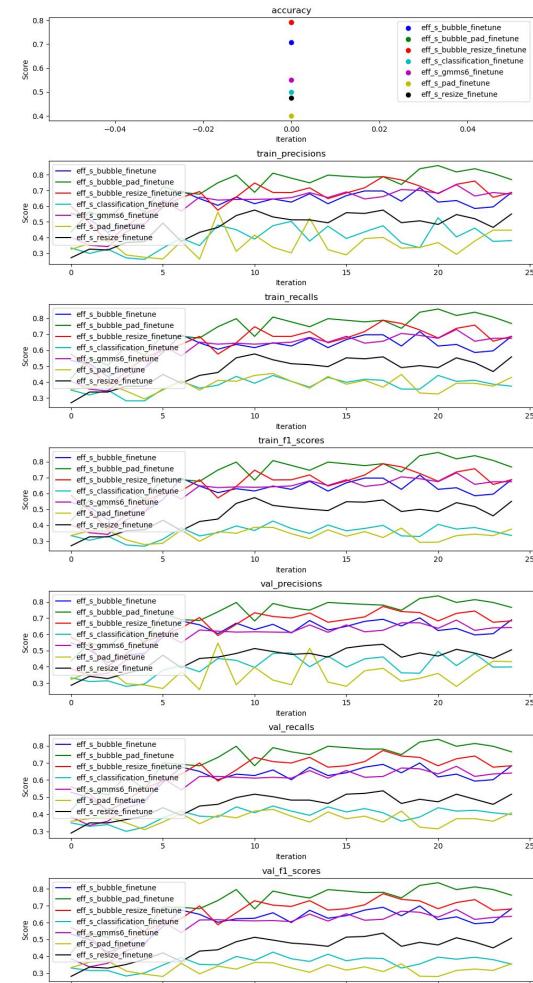
Resnet18



MobileNet Small

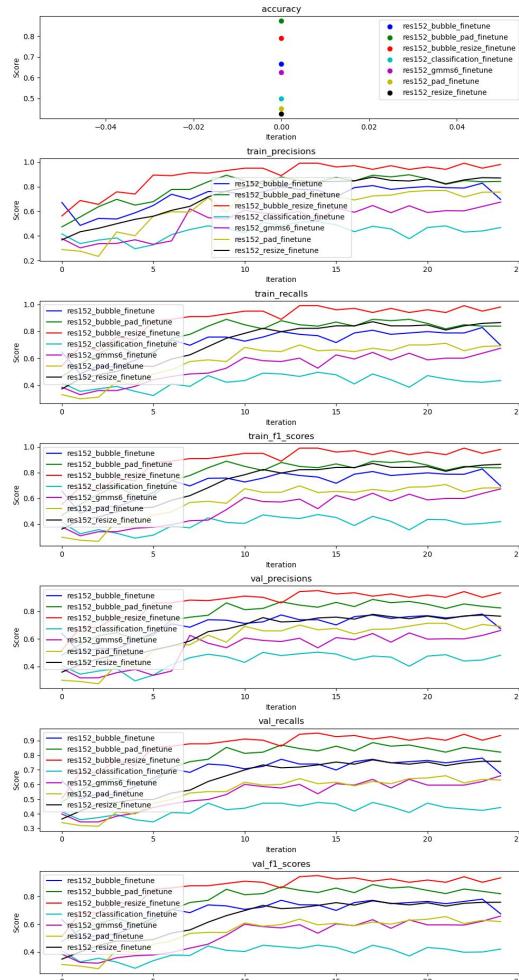


EfficientNet Small

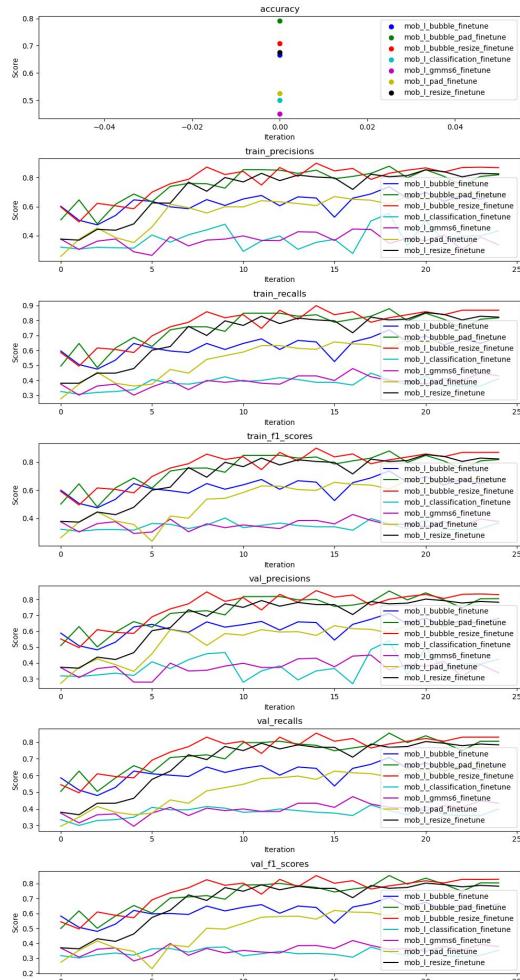


Largest sized models, trained using fine tuning

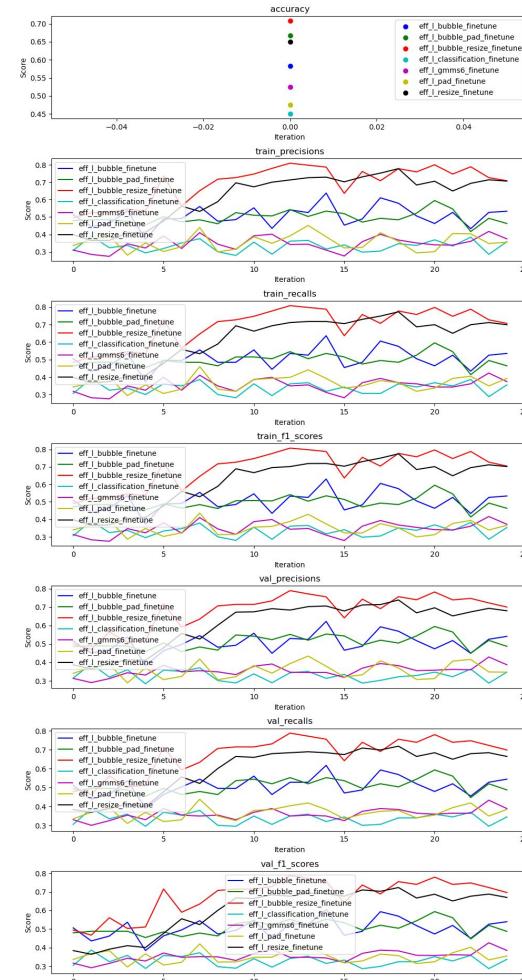
Resnet152



MobileNet Large



EfficientNet Large



Transfer vs finetune

Under/Normal/Over

Finetune	Accuracy	Transfer Learning	Accuracy
eff_s_classification_finetune	0.5	eff_s_classification_transfer	0.575
eff_s_gmms6_finetune	0.55	eff_s_gmms6_transfer	0.5
eff_s_pad_finetune	0.4	eff_s_pad_transfer	0.625
eff_s_resize_finetune	0.475	eff_s_resize_transfer	0.525
eff_l_classification_finetune	0.45	eff_l_classification_transfer	0.525
eff_l_gmms6_finetune	0.525	eff_l_gmms6_transfer	0.5
eff_l_pad_finetune	0.475	eff_l_pad_transfer	0.475
eff_l_resize_finetune	0.65	eff_l_resize_transfer	0.575
res18_classification_finetune	0.475	res18_classification_transfer	0.675
res18_gmms6_finetune	0.425	res18_gmms6_transfer	0.525
res18_pad_finetune	0.45	res18_pad_transfer	0.5
res18_resize_finetune	0.475	res18_resize_transfer	0.625
res152_classification_finetune	0.5	res152_classification_transfer	0.55
res152_gmms6_finetune	0.625	res152_gmms6_transfer	0.525
res152_pad_finetune	0.45	res152_pad_transfer	0.525
res152_resize_finetune	0.425	res152_resize_transfer	0.5
mob_s_classification_finetune	0.45	mob_s_classification_transfer	0.375
mob_s_gmms6_finetune	0.525	mob_s_gmms6_transfer	0.575
mob_s_pad_finetune	0.475	mob_s_pad_transfer	0.45
mob_s_resize_finetune	0.425	mob_s_resize_transfer	0.55
mob_l_classification_finetune	0.5	mob_l_classification_transfer	0.6
mob_l_gmms6_finetune	0.45	mob_l_gmms6_transfer	0.525
mob_l_pad_finetune	0.525	mob_l_pad_transfer	0.55
mob_l_resize_finetune	0.675	mob_l_resize_transfer	0.575
Average accuracy:	0.4947916666	Average accuracy:	0.5385416666

Bubble/No Bubble

Finetune	Accuracy	Transfer Learning	Accuracy
eff_s_bubble_finetune	0.7083	eff_s_bubble_transfer	0.625
eff_s_bubble_pad_finetune	0.79166	eff_s_bubble_pad_transfer	0.625
eff_s_bubble_resize_finetune	0.79166	eff_s_bubble_resize_transfer	0.8333
eff_l_bubble_finetune	0.583	eff_l_bubble_transfer	0.708
eff_l_bubble_pad_finetune	0.666	eff_l_bubble_pad_transfer	0.75
eff_l_bubble_resize_finetune	0.7083	eff_l_bubble_resize_transfer	0.7916
res18_bubble_finetune	0.79166	res18_bubble_transfer	0.7083
res18_bubble_pad_finetune	0.7083	res18_bubble_pad_transfer	0.666
res18_bubble_resize_finetune	0.833	res18_bubble_resize_transfer	0.625
res152_bubble_finetune	0.666	res152_bubble_transfer	0.7083
res152_bubble_pad_finetune	0.875	res152_bubble_pad_transfer	0.5833
res152_bubble_resize_finetune	0.79166	res152_bubble_resize_transfer	0.75
mob_s_bubble_finetune	0.5833	mob_s_bubble_transfer	0.625
mob_s_bubble_pad_finetune	0.75	mob_s_bubble_pad_transfer	0.6666
mob_s_bubble_resize_finetune	0.75	mob_s_bubble_resize_transfer	0.7083
mob_l_bubble_finetune	0.666	mob_l_bubble_transfer	0.625
mob_l_bubble_pad_finetune	0.79166	mob_l_bubble_pad_transfer	0.625
mob_l_bubble_resize_finetune	0.7083	mob_l_bubble_resize_transfer	0.7916
Average Accuracy:	0.73148	Average Accuracy	0.68981

Here I have collected the accuracies for each model I have trained. We can take a look to see if transfer learning vs fine tuning seem to be definitively better.

It would appear, that neither are better.

Metric Analysis

With the current accuracy of around 50%, it seems like we're hitting a roadblock. No specific preprocessing or fine-tuning method is proving to be superior. The stagnant performance metrics after 10 epochs might hint that our model has reached its capacity with the given data.

Things to try:

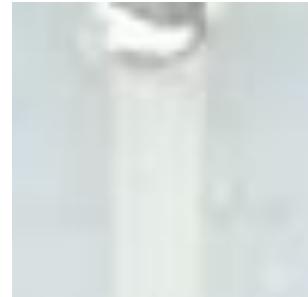
- Cross-validation
- Collect more data
- Refine dataset

I now have added around 50 new images to each class, so each class now has 104 images.

Refining my dataset

In my earlier work, I put images into 'over' or 'under' groups even if they only showed a tiny bit of over or under extrusion. Now, I'm going to change that. I'll only put an image in the 'over' group if the material is really getting wider. I'll only put it in the 'under' group if it's clearly getting thinner. Some pictures I had put in 'over' or 'under' before will now go into the 'normal' group.

I used to label these two pictures showed overextrusion, but I've changed my mind. I'm either going to move them to the 'normal' group or take them out completely, whichever keeps the groups balanced. I only had to get rid of 5 pictures in all, but I hope this will make our results more accurate.



6/12 - 6/19

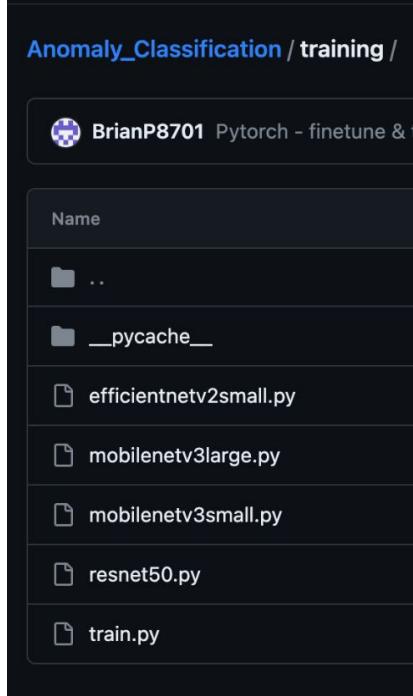
- Train all classification models and provide metrics
- Improve robustness of preprocessing step for low standard deviations
- Optimize preprocessing step
- Test virtual marker

Training modules done

I finished coding the training for models on Pytorch.

Besides making a model for each dataset I mentioned earlier, I'll also test fine tuning and transfer learning. Fine tuning starts with a pre-existing model and trains the whole thing. Transfer learning also starts with a pre-existing model, but only trains the last layer. Transfer learning should work better because it avoids overfitting, but we'll have to see.

Also, we need more data. I'll go through the new videos to get more.



The screenshot shows a file explorer window titled "Anomaly_Classification / training /". It displays a list of files and folders:

- Icon: BrianP8701 Pytorch - finetune &
- Name: ..
- Name: __pycache__
- Icon: efficientnetv2small.py
- Icon: mobilenetv3large.py
- Icon: mobilenetv3small.py
- Icon: resnet50.py
- Icon: train.py

Purpose of Preprocessing

Our preprocessing step does two things:

1. It reduces noise and sharpens edges so we can see the most recently extruded material's edges and measure its diameter.
2. It helps our trained models work better. Preprocessing can make a big difference if you're dealing with small, low-quality, or noisy datasets. If the images in the dataset have a lot of noise or if the objects are hard to see, reducing noise and sharpening edges can make the objects stand out more.

Refresher of the current simplify algorithm

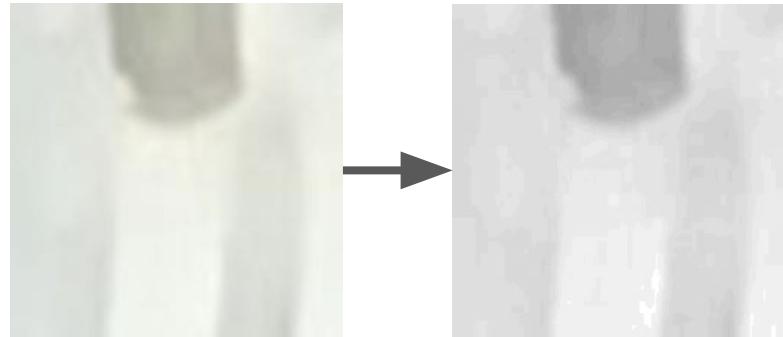
This slide is just to explain how the current preprocessing step works:

1. Figure out the standard deviation of all the pixel intensities in the image. This tells us how much the colors in the image vary.
2. Using the standard deviation, we calculate a 'threshold'. This threshold decides which pixels are similar enough to group together. If the colors in the image are very different, the threshold will be larger. If the colors are similar, the threshold will be smaller.
3. If the standard deviation is below a certain point, we set the threshold to a fixed value.
4. Find the brightest pixel in the image.
5. Turn all pixels that fall within the threshold of the brightest pixel to maximum intensity, making them white.
6. We adjust the range to include pixels that are slightly less bright, excluding the ones we just turned white. This is our new 'cutoff'.
7. We repeat this process, finding the brightest pixel below our cutoff and adjusting all pixels within its threshold to match its intensity. We do this until we've looked at and adjusted all pixels in the image.

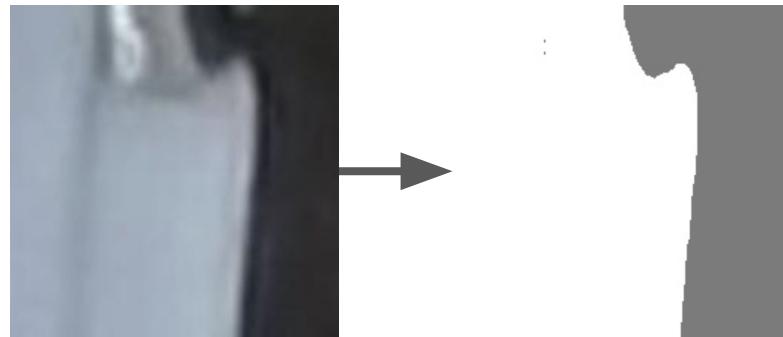
After all this, the image is simpler because similar pixels have the same intensity. This makes the edges stand out and reduces noise.

Failings of the current preprocessing step

When the image is more uniform in intensity, this algorithm can introduce more noise.



This error shown to the right is clearly unacceptable. There bed is in the image, which leads the algorithm to calculate a higher standard deviation. This leads the algorithm to believe it doesn't need to be as sensitive, and thus ignores the faint line.

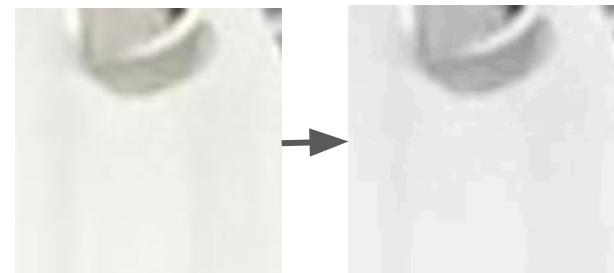
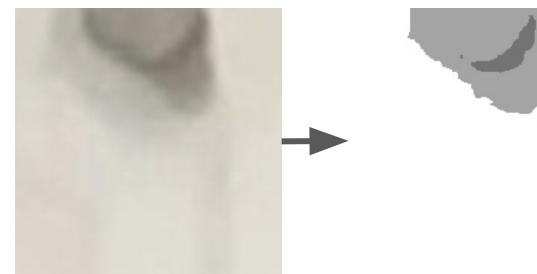
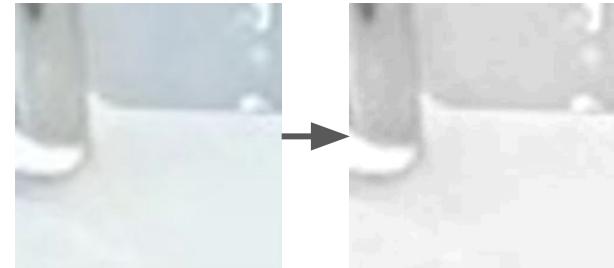


How robust?

Let's look at the images on the right. The edges are really faint. In the first image, the bottom line isn't even there.

First, this means when we're measuring diameter, we need to know when we can't get a measurement.

Second, for the last two images, the edges are there but they're hard to see. Should we give up? Or can we figure out how to handle these images, especially since they're common when layers build up?



A question of balance

What's more important - sharp edges or less noise? We need a balance. We have some faint edges. Using the standard deviation doesn't always work well because the bed or tip can be a dark color, which can make the algorithm unstable. We could crop out the tip with accurate tip tracking, but the bed is still an issue.

We need to make sure we don't lose important information. Too much noise reduction can remove important details. Too much sharpening can highlight noise or make fake edges. But a bit of extra sharpening shouldn't hurt our diameter measurement algorithm.

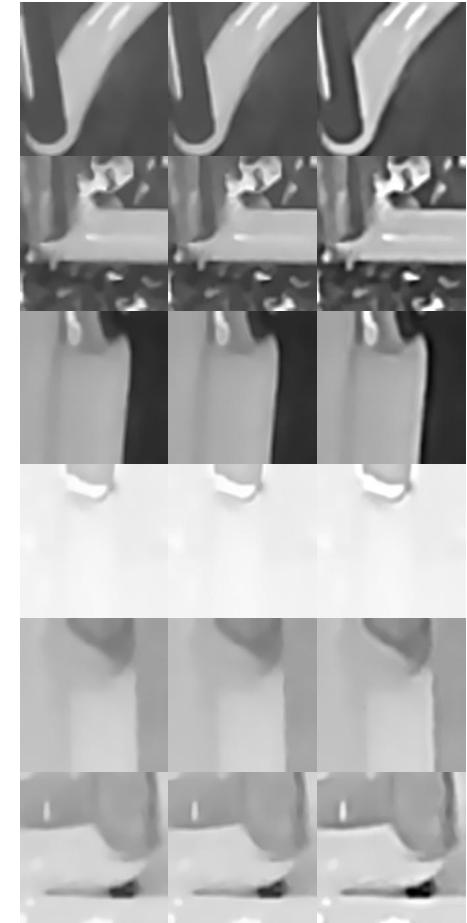
In conclusion, we must prioritize avoiding relevant information loss.

Cv2 and PIL Preprocessing

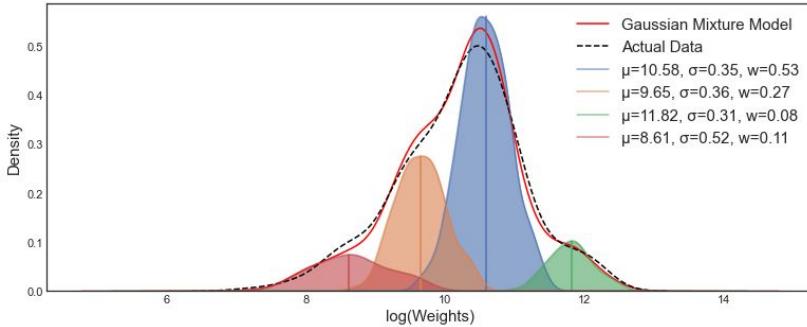
Non-local Means Denoising algorithm from cv2

Unsharp Mask from PIL

I tried out the Non-local Means Denoising algorithm from cv2 and Unsharp Mask from PIL. They work alright after I played with the settings for a while. But we still need something better to highlight faint edges. Maybe we can come up with a custom solution.



New preprocessing step



I remembered something from my textbook, 'Deep Learning'. It's about a thing called a Gaussian Mixture Model. This model thinks data is made from different groups. Each group has its own Gaussian distribution. This is really helpful for understanding complex data.

We provide number of clusters we think there are to the EM (Expectation-Maximization) algorithm, which provides us the means and variations of each cluster.

Check out the image above I found online to explain this.

Right now, we treat each pixel as if it belongs to one normal distribution. But maybe we can use a Gaussian Mixture Model to separate our pixels into multiple groups:

- The tip
- The tip's glow
- The extruded material
- Shadows
- Bed

We want to pay more attention to the brighter colors, which are the extruded material, and less to the darker colors of the bed or tip.

But a Gaussian Mixture Model doesn't let us do that. So we need to find a workaround.

Why we cannot use k-means

K-means clusters data faster than a Gaussian Mixture Model and the EM algorithm. But since our data isn't that big, the time difference doesn't really matter. I ran both 1000 times and found that the Gaussian Mixture Model is actually quicker with our input.

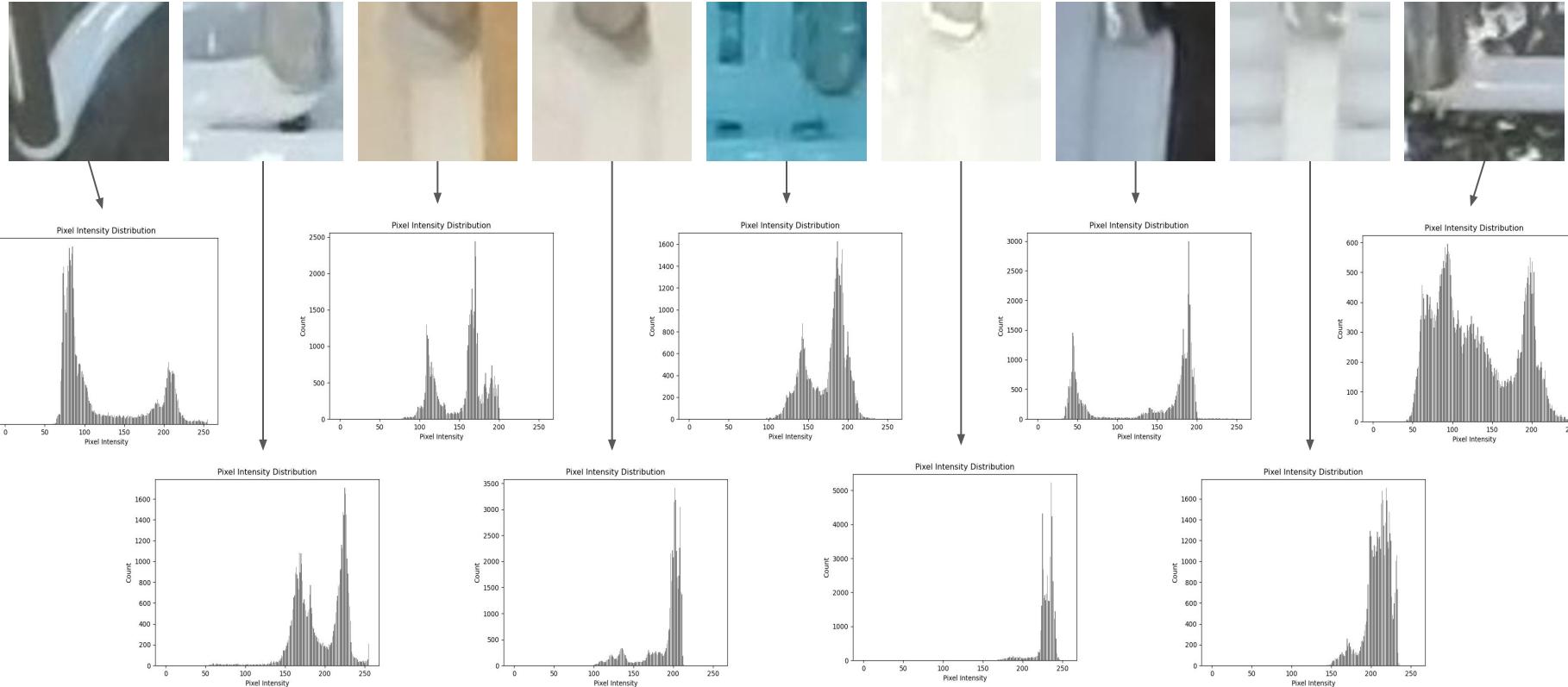
k-means \approx 0.062 seconds

GMM + EM \approx 0.014 seconds

Another issue is that k-means expects clusters to be spherical with equal variances. That's not the case for us. We also need to know the standard deviations to break up our data into sections.

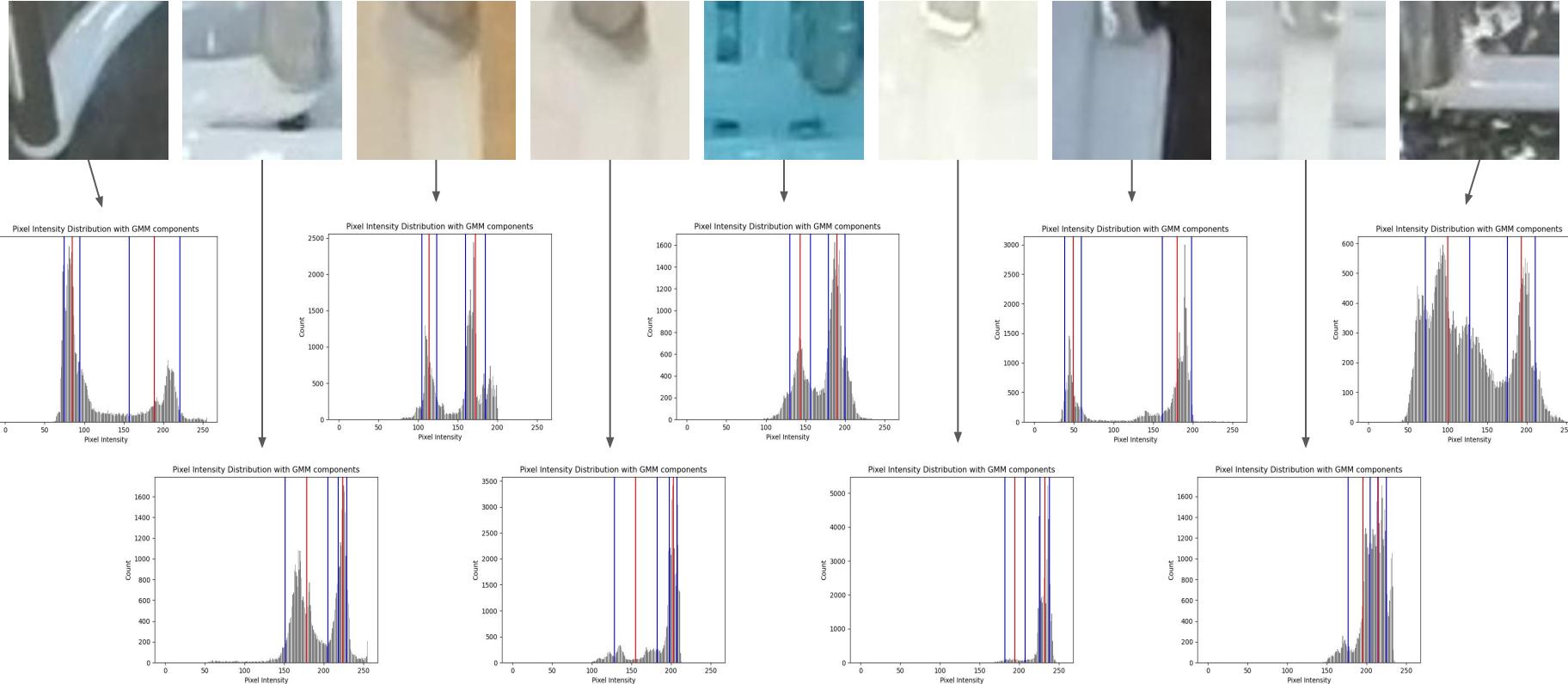
Visualization

The x axis is pixel intensity, and the y axis is the count of how many pixels have each intensity



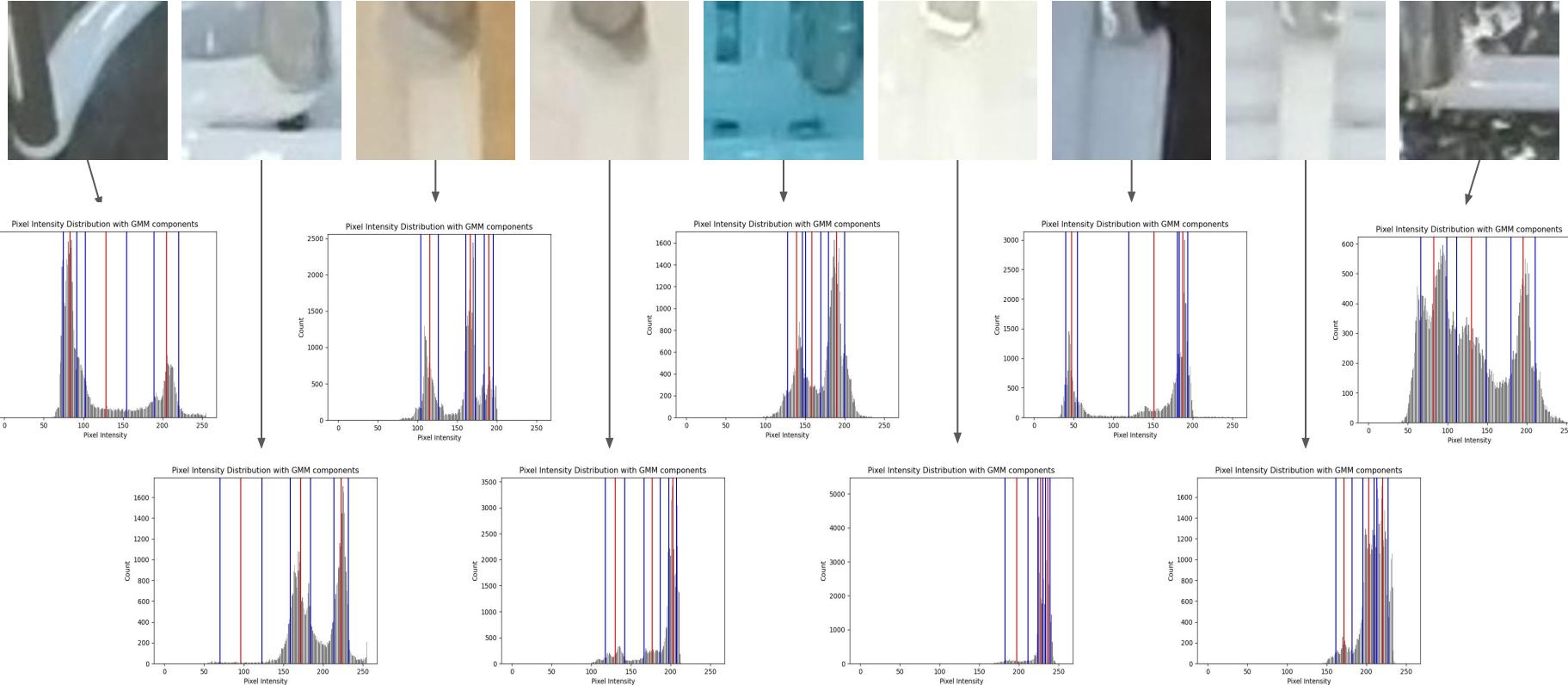
Visualization 2

This time we find **2** clusters. Red: Mean. Blue: +/- Standard Deviation.



Visualization 3

This time we find **3** clusters. Red: Mean. Blue: +/- Standard Deviation.



Observation



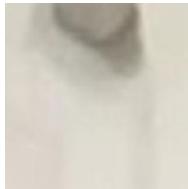
200-215



215-225



170-190



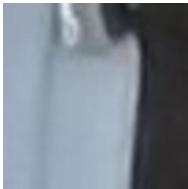
207-209



185-190



230-240



180-190



215-230



185-200

I manually measured the intensity value of the extruded material. If you look back at the last slide, you'll see it matches the third mean and its range almost exactly.

So now we can approximate the color of the material.

GMM Preprocessing Step

Parameters: image, num_components

Here are the steps for the new method using the EM algorithm.

1. Get means and variances for clusters.
2. Create ranges around each mean using standard deviation.
3. Fix gaps and overlaps in the ranges.
4. Replace all colors in each range with the same color.

This works for any image and any number of components.

GMM + Sensitivity Step

Parameters: image, num_components

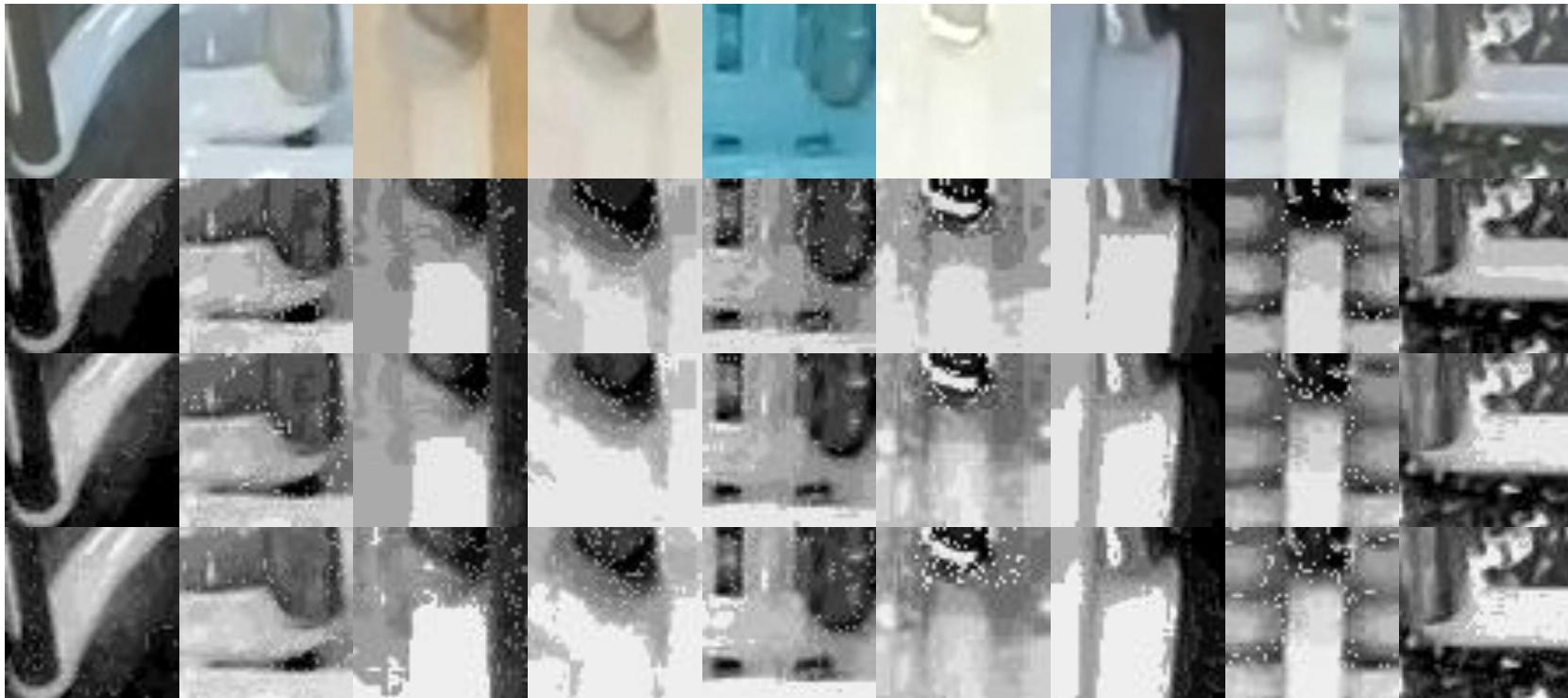
For this method, we're focusing on the highest mean (The material):

1. Get means and variances for three clusters.
2. Select all data from the highest mean (plus or minus two standard deviations).
3. Calculate means and variances for these clusters.
4. Create ranges around each mean using standard deviation.
5. Combine this with the lowest range from the first three clusters.
6. Fix gaps and overlaps in the ranges.
7. Replace all colors in each range with the same color.

This also works for any image and any number of components.

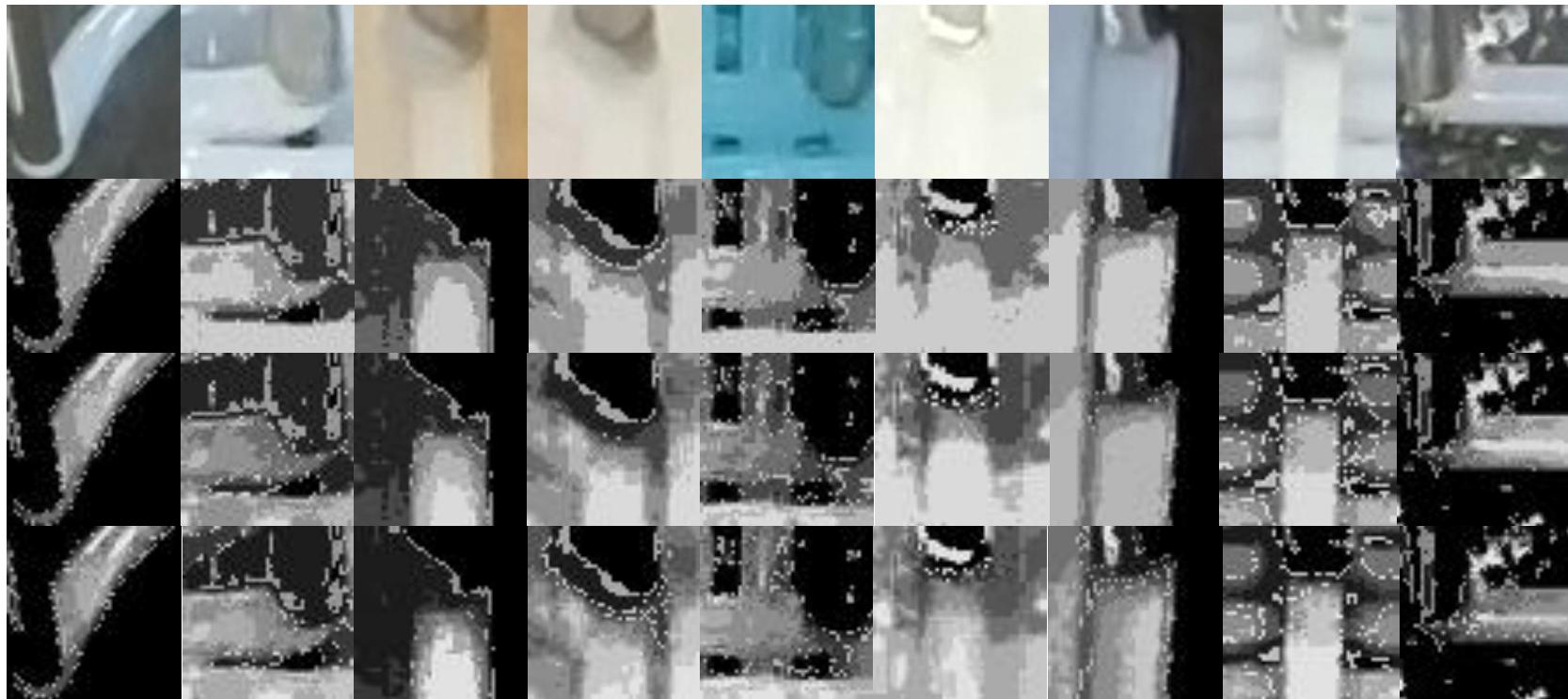
GMMx: Gaussian Mixture Model Algorithm
with x components

Original



GMMSx: Gaussian Mixture Model +
Sensitivity with x components

Original



Preprocessing Conclusion

The new method removes noise, highlights edges, and doesn't lose important info. It even highlights faint edges that are hard to see. It's more sensitive to the material, and it darkens the tip or bed.

We can use fewer clusters, which cuts down on noise and makes things faster.

Overall, this method should help with training models for classification and measuring diameter.

Colors are not important. Shape matters.

Here are a few ways we can make the model focus on shapes, not colors:

1. Use data augmentation techniques that change colors, like changing brightness, saturation, or the color palette. This is because colors aren't important for our problem. The shape of the extruded material is what matters to us.
2. Design the CNN to focus more on shapes. This could mean using larger convolutional kernels or more pooling layers. That way, the model will pay more attention to larger shapes and less attention to small color changes.

Virtual Marker Test

In a comparison, the YOLO v8 model worked better than the virtual marker. I put 30x30 markers randomly on frames with some noise, rotation, and light changes - but still less than in real life. After running each model 1000 times, the YOLO v8 model came out on top when looking at the averages.

Yolo v8 Tip Detection: 1920x1080 image ≈ 0.46 seconds

Virtual Marker: 1920x1080 image, 20x20 marker ≈ 0.61 seconds

In a real scenario, there would be even more noise and camera variation. Additionally, we haven't even pruned or implemented Tiny YOLO which would further improve the efficiency of the model.

Thus, we can safely continue with our original plan.

6/5 - 6/12

- Plan for what and how to train classification models for underflow/overflow/normal/bubble. Train a variety of models.
- Organize all trained models, and clearly show metrics and details explaining each.
- Research and choose what type of marker would be best to use in our scenario.
- Test virtual marker.
- If virtual marker is computationally efficient enough to run 30 times a second on raspberry pi, try in real world.
- Clean up, update and add documentation to code. Push to github.
Reorganize, add dependencies, and error messages.

Classification Models

Pre-Trained models for Transfer Learning I will use:

1. MobileNetV2: Prioritizing efficiency
2. ResNet50: Prioritizing accuracy
3. EfficientNet: Balance of efficiency and accuracy

Training on the unzoomed image will not work, because the model will not be able to achieve good accuracy without overfitting given our limited dataset. So I will crop around the tip at sizes of 50x50.

These models were trained on the ImageNet dataset, thus their convolutional filters have learned features at that scale. At 224x224, these models should provide better accuracy, but smaller inputs will provide better efficiency.

I'll create a variety of datasets with different preprocessing steps to compare.

- 50x50
- 50x50 resized to 224x224
- 50x50 with padding to 224x224

Preparing datasets for classification models

The code can be found [here](#). I've split it into folders for image tools (image_utils), datasets and the files which train the models.

First, I need to collect my data. I'll want to crop a 50x50 area near the tip covering the material for each image in the dataset. To do this quickly, I created a tool to collect this data, demonstrated in the video below. This is the file image_crop_gui.py

I then created the preprocessing steps to resize and add padding. I am also taking the preprocessing step from my other project and will use that to create another dataset. Perhaps that preprocessing step, removing noise and sharpening edges will lead to better model performance. But for now I still need to make some upgrades to that algorithm. This is all in the file preprocessing.py

I made sure to add comments and make everything easy to understand and read.



```
> datasets
  > bubble_datasets
    > bubble
    > no_bubble
  > bubble_pad_datasets
    > bubble
    > no_bubble
  > bubble_resize_datasets
    > bubble
    > no_bubble
  > classification_datasets
    > normal
    > over
    > under
  > pad_datasets
    > normal
    > over
    > under
  > resize_datasets
    > normal
    > over
    > under
```

Training hyperparameters

Given our smaller dataset, we want a bigger batch size. A bigger batch size reduces variation as more data points are sampled at once, and will lead to better regularization. Overfitting is not an immense concern of ours. Additionally we want learning rate decay, early stopping and data augmentation in the form of rotation.

I'll visualize metrics in TensorBoard and choose the best performing model based on validation set.

<https://pytorch.org/vision/stable/models.html>

Note: Things were going swimmingly. I had trained a MobileNetv2 model. But now when I try to import torch, I get the following error: zsh: segmentation fault

It gives no other details, and I'm really struggling to get past this.

Code improvements

- Github: <https://github.com/BrianP8701/STREAM.AI>

My branch was behind 8 commits. My Github hasn't been updated since April 19th because I made some commits containing large files which couldn't be uploaded. I reset those commits, and uploaded what I have now.

- Created Conda environment
- Added requirements.txt with only necessary dependencies
- Replaced all complex logic in main algorithms with helper functions. You can now read through the main algorithm very easily as it only consists of initialization variables and readable functions and names.
- Added more comments and restructured code

Pose Detection

AprilTags and AruCo markers are the state of the art markers for pose detection today. AprilTags are more robust, which is what we need as the camera will not always be positioned the same and there will be lighting variation.

I will do an experiment with a virtual tag first to measure the computational efficiency. For this, I'll simply place the tag on the image at a realistic scale, and measure the efficiency. If it is not more efficient virtually, it surely will not be more efficient in real life with noise.

I will compare to the Yolo v8 Tiny model for comparison.



4/14 - 4/21

- Temporal Correction

	Standard Horizontal	Pre Vertical
Inference Cap	Each instance only takes one inference, so we can have a lower cap.	Each instance we have to backtrack and do more inferences, so set a higher cap.
Reliability	Unreliable, due to inherent variation.	Near perfect metric for temporal offset.

As noted above, the prevertical is a much better way of measuring the temporal error. Yet we still want to develop the standard horizontal method, as we will find that not all prints will contain those specific movements. I have included two constants, that can easily be adjusted to choose how much of either one you want, which can be chosen as needed for whatever specific print is being done.

Implementation Details: Conclusive vs Inconclusive State

This state merely represents:

- Inconclusive: We don't know if we are ahead or behind
- Conclusive: We know if we are behind

Based off whether we are in a conclusive or inconclusive state determines how we behave in certain situations.

For example, if we are in an inconclusive state we cannot find the temporal offset during a pre-vertical move.

Figure 1:

Blue (Standard Horizontal temporal offsets)

Red (Pre-vertical temporal offsets)

This graph shows all of our data collected without removing anomalies. In the standard horizontal when the parser falls significantly behind the offsets go bonkers. The prevertical on the other hand is more steady.

Figure 1

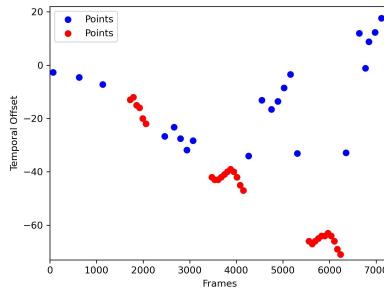


Figure 2

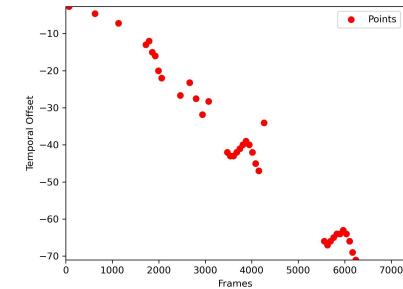


Figure 2:

Red (Standard Horizontal and pre-vertical temporal offsets)

This graph is the same as figure 1, expect that the anomalies are removed.

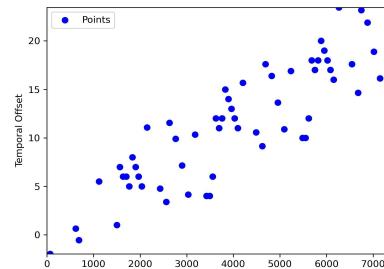
Both images above have an initial time_k of 0.99. Here we are simply running our parser and measuring the temporal offsets without taking any actions.

Figure 3 & 4:

Blue (Standard Horizontal and pre-vertical temporal offsets)

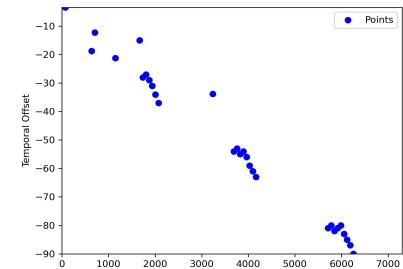
Here I just display two more graphs displaying the temporal offsets over time for two different time_k values. As we can see, the slope appears to correspond to how much we should change time_k by, as it tells us the rate at which the temporal offset is changing.

Figure 3



Time_k = 0.98

Figure 4



Time_k = 0.995

Collecting more data to see what the slope of the error vs frames is for different initial time constants. Next, we want to do this across multiple videos and find a consistent way to convert that slope into a correction.

Initial time_k	Slope	Time_k correction
0.97	0.01548156275287975	+0.015
0.975	0.004778425833443505	+0.01
0.98	0.0028197907793473874	+0.005
0.983	0.0017968711589403047	+0.002
0.984	0.0016009852751633009	+0.001
0.985	0.00014960637206662498	0
0.986	0.00014960637206662498	-0.001
0.987	-0.0007433997167368042	-0.002
0.99	-0.011362640420807129	-0.005
0.992	-0.011970263220423117	-0.007
0.995	-0.013227478738805885	-0.01

4/7-4/14

- Temporal Correction

Overview of Temporal Correction

To clarify, we are no longer seeking to correct spatial error. We are trying to see if the parser is behind or ahead temporally.

Within the gparser, we now must work with the true angles. Further processing and adjustment may be performed in measurement.

We must catch the error before it gets so bad we can't fix it.

Even in the best tracking, there is spatial error. We must only make decisions if the tracker is consistently ahead or behind.

Terminology Clarification

Offset, Spatial Error, Spatial Offset: The physical distance by which the tracker is off from the actual location.

Sampling Inference: Performing inference on the image using Yolo v8.

Time_k, acceleration constant: This is a mistake on my part sorry. In my code the line which contains the unknown constant is:

```
time for move = how long(abs(magnitude(position vector)), abs(magnitude(curr velocity vector)),  
abs(final speed), abs(max speed), ACCELERATION) * time k
```

There are two values that can be adjusted, ACCELERATION and time_k. Both have the same effect, and I will be leaving ACCELERATION constant and adjusting time_k. So from now on I will say time_k, but whenever in previous slides I had said acceleration, substitute in time_k.

Terminology Clarification

Tip tracker: When I say tip tracker, I am specifically referring to the tiny box that attempts to follow the tip around.

Temporal Error: My code parses through the gcode at a certain pace, and can fall ahead or behind. Temporally ahead or behind refers to the fact that my tip tracker is not aligned temporally with the real print.

Time Travel: Referring to when my tip tracker is ahead or behind, not making a spatial correction but rather moving forward or backwards some frames to correct.

Implementation Details of Temporal Correction

Instead of sampling inferences at given intervals, we will select our inferences based off the gcode now. When the printer is moving horizontally we will take an inference. We can make the following logic:

	Offset to left	Offset to right
Moving left	Temporally ahead	Temporally behind
Moving right	Temporally behind	Temporally ahead

Moving horizontally at an angle is acceptable. So in the middle of that line of gcode, we can take an inference and tell if we are temporally ahead or behind. We will place a cap at a maximum inference rate of 60 frames, so we won't oversample.

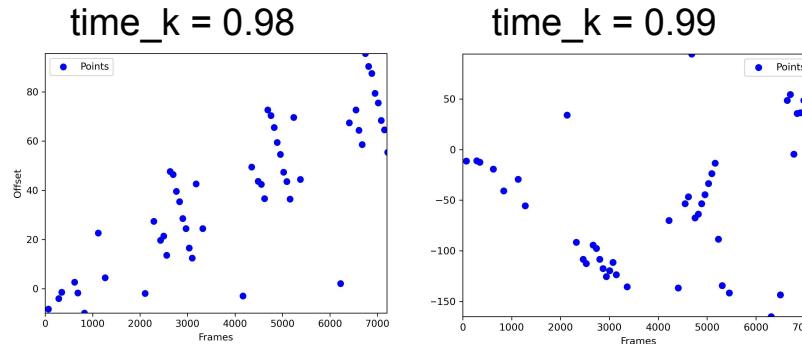
Implementation Details of Temporal Correction

We will maintain and build a list of offsets and their corresponding frames as we take inferences.

(-) behind, (+) ahead

Example, where we are clearly ahead:
[0,0,-1,0,1,3,2,5...] offsets
[5,137,256,678...] frames

We can now plot this to see how the offset looks as the video goes by. No correction is being made for now, just measurements. The correct time_k value should be 0.985, and here we can see the offset for two different values of time_k throughout the video. Positive offset means the tracker is ahead, and vice versa for negative.

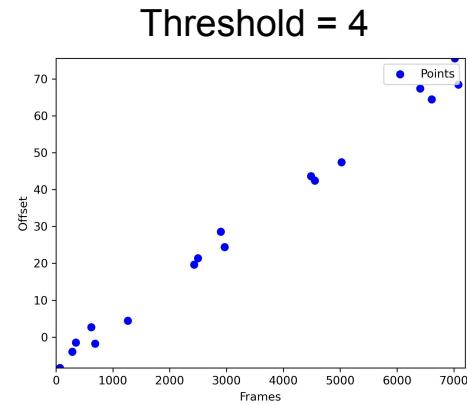
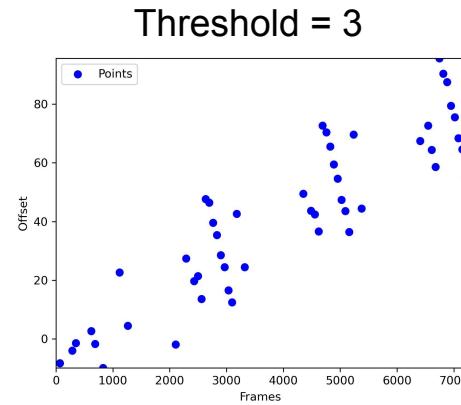
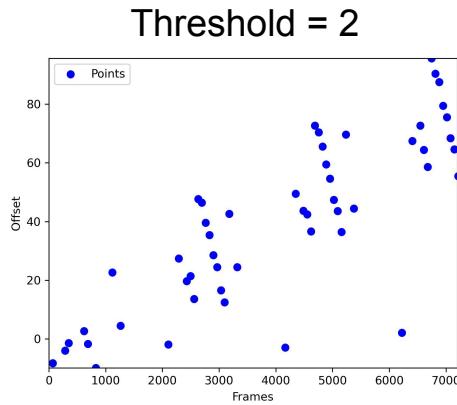


Here we can see the tracker is clearly getting steadily ahead throughout the entire video, with a couple outliers.

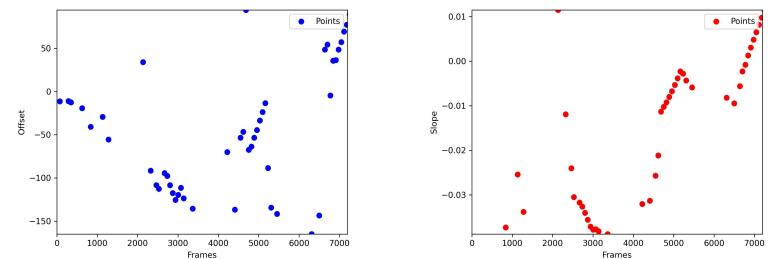
Here we can see the tracker is getting behind, but around 4000 frames, the tracker falls so much behind that our logic falls apart. We want to make corrections far before that point.

Removing Anomalies

We first calculate the best-fit line using least squares. Then, calculate residuals. Next, compute the Z-scores for residuals and filters out the points with Z-scores greater than the specified threshold. This approach considers anomalies based on their deviation from the best-fitting line, rather than the mean of the data. I chose to use a threshold of 3.

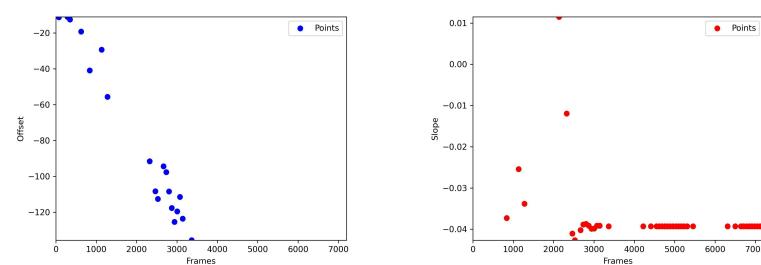


Measured offsets vs time, and the slope of the best fit line vs time without removing anomalies.



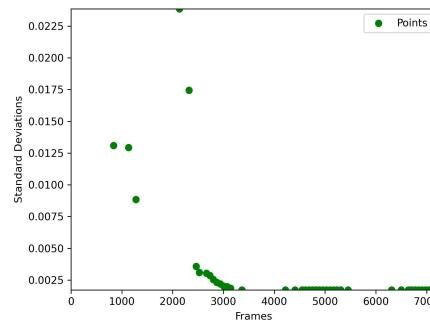
We can clearly see that, that the best fit line will go bonkers if we do not remove anomalies.

Measured offsets vs time, and the slope of the best fit line vs time removing anomalies.



Here, the slope eventually settles on a value. We can use this to tell us how much to change the time_k constant by.

Now, you might ask, how do we know when to make a correction? When the standard deviation of our slope is clearly ahead or behind we can confidently make a decision. However, as we will see later, this is not sufficient.



Implementation Details of Temporal Correction

Given our offsets and time, we can calculate the slope and standard deviation of the best fit line.

The slope tells us: The magnitude of the temporal error

Slope	Time_k correction
-	-
+	+

The standard deviation of the slope tells us: Our confidence in this decision. It appears a value of 0.003 or 0.002 tells us we are very confident

Issues

- 1) It takes too long for the slope to converge to a value and for the standard deviation of the best fit line to settle.
- 2) I mentioned last week that we want to over accelerate then decelerate to fix the temporal error. I changed my mind, that introduces too much complexity.
- 3) In the previous slides, I was proposing that we change the slope based on the magnitude of the offset. However, these current prints are very simple, consisting of lots of straight horizontal and vertical lines. If we were to move more diagonally, the magnitude of the spatial offset would seem less than it really is. Additionally, the speed of the tip contributes to this. If the tip is moving very fast, we might see a large offset, even if the temporal error is small. Thus, we must consider the angle and speed of the tip in certain situations when measuring offset.

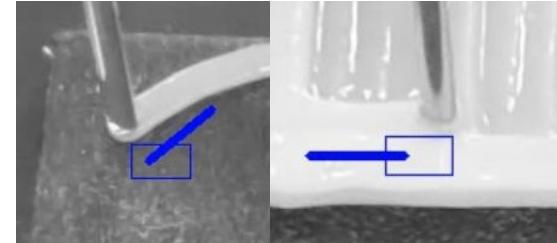
Further,

The entire program overall is still exceedingly fast, finishing the entire 5 minute video in around 15 seconds not including measurement. (Measuring diameter is quite inefficient, but I will do optimizations on that algorithm later).

My approach now is to try to find more places we can do inference so we can move quickly, and confidently conclude whether we are ahead or behind.

My issue with this, is that it doesn't make this algorithm robust with respect to the variety of possible gcode. What if a print doesn't have these types of moves? Are there other possible movements that can be valuable? In following slides, I will attempt to concisely categorize and define types of movements and the logic for determining when they occur and whether they are ahead or behind.

Standard horizontal moves



I already discussed these, but I will reiterate with the new changes. We want to standardize all of these regardless of speed or angle. So we can simply:

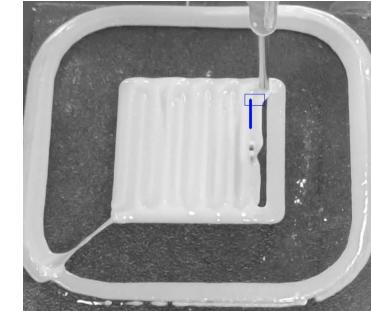
$$\text{temporal error} = \frac{\text{offset}}{\text{x velocity}}$$

Dividing by the x component of the velocity accounts for the angle and the speed, giving us a consistent view of the offsets. With this addition, we do the same thing as I mentioned previously. Additionally, to be conservative and given the inherent offsets during print, we will only consider horizontal within the range of 60 degrees in either direction. The only change here, is that offsets should be more accurate and more robust when we have gcode that has more diagonal moves.

Quick horizontal move, followed by vertical move

In this particular print, there are some very quick movements the tip makes occasionally. These are so fast, they sometimes take one frame and my algorithm doesn't do inference on them. However, these are very valuable, yet the logic for determining whether we are temporally ahead or behind changes.

	At start of move	At end of move
Tracker is ahead	Spatially correct	Spatial error
Tracker is behind	Spatial error	Spatially correct



It is important to note, that the magnitude of the offset is not relevant in this case. Merely seeing if there is an offset is all we can do. Now based off repeated instances of an error we can make a change, but we cannot determine the magnitude of this change.

Quick horizontal move, followed by vertical move

We know the moment when we start this move from the gcode. To find the temporal offset, we need to backtrack and see exactly when this spatial offset began. To achieve this the first time we do it, we need to brute force and check previous frames one by one until we find the moment the spatial offset begins. Once we begin collecting more data we can make a more informed decision.

	Temporally behind	Temporally ahead
Parser move	When spatial offset ends	When spatial offset begins
Real move	When spatial offset begins	When spatial offset ends

Time Travel

Now, to address the issue of having to “over accelerate then decelerate.” That is unnecessarily complex. Rather than that, we want to change the time_k constant to what we believe is truly correct. Then, let’s “time travel”, meaning move forwards or backwards some frames, removing the need to over accelerate then decelerate. The question now is how many frames to “time travel”?

Additionally, we cannot actually go back in time as we assume this will run in real time. We will use a method I read about in my Distributed Systems textbook for synchronization, using “leap” frames or “buffer” frames. If we want to go ahead in time we add skip some frames in our future moves, and we add extra frames if we want to slow down. The details of when this will be performed will be detailed later.

Total View, Overall Consistency

We cannot assume that we can make the exact perfect temporal and time travel correction the first time. We want to maintain a total view (A word I made up). I mean we want to keep track of all the previous data ([frames, offsets], [frames, ahead/behind]) we've collected and what actions (temporal corrections, time travel) we've taken.

Now, we will make a decision as early as we can, but as we gather more data later into the video and save it, we can use the entirety of all our data to eventually come closer to reaching a definitive and final conclusion.

An important decision is that we will run two different parsers now. One which is the original with none of our actions performed, alongside our “corrected” one. We want the original parser so we can maintain the same temporal vs time graph and continue to build confidence, rather than start all over every time we make an action.

<pre>frames: [...] int offsets: [...] int</pre> <p>Frames and their corresponding offset from standard horizontal moves</p>	<p>We use this data with the least squares method to determine the magnitude of the change we need to make to time_k.</p>
<pre>frames: [...] int ahead/behind: [...] boolean temporal_offset: [...] int</pre> <p>Frames and their corresponding values from quick horizontal -> vertical</p>	<p>When this data shows consistency we can make changes to time_k and confident time travel decisions.</p>
<pre>frames: [...] int time_k adjustment: [...] float # of time travel frames: [...] int</pre> <p>Actions taken</p>	<p>This data can be useful when we want to</p>

Acceptable Errors

There is inevitably a little error even in the best of runs. Our data shows that these are acceptable errors.

	Standard Horizontal	Quick Horizontal -> Vertical
Spatial	20 pixels	8 pixels
Temporal	N/A	2 frames

4/1 - 4/7

- Run correction purely for acceleration adjustment
- Refactor code

Spatial Correction

Spatial correction has been solved. Inference rate minimum is 5 seconds, and maximum 10 seconds. It starts at 5 seconds and can increase or decrease based on whether there are errors.

I maintain a stack data structure that keeps track of the offsets in the x direction (difference between yolo v8 prediction and my calculated location of the tip). Sometimes, inferences don't meet the confidence minimum, so we simply ignore those instances. Whenever the stack has two values, I will remove both and see if they are offset in the same direction. If so, I will make a spatial correction that is the average of both offsets.

Temporal Correction

Some things to note before talking about temporal correction:

- Spatial corrections don't fix temporal errors
- When making temporal decisions, do not account for the corrections made spatially. Keep track of all spatial corrections made, and remove it from temporal offsets.
- To catch up we need to over accelerate and then decelerate once we've caught up (Vice versa to slow down). We need to avoid oscillating back and forth between acceleration values and settle on a value.

Temporal Correction

When the parser begins we assume that it is starting correctly. In other words, in the start the tip tracker is correct. But slowly throughout the video the tracker will begin to deviate from the real location if the acceleration value is off. So we want to always be on high alert and make small corrections to the acceleration value as we see fit.

Let's ignore the spatial corrections, and look at my actual tip tracker prediction vs the yolo v8 inference when correcting the temporal error. Let's not look at the magnitude of the error, but the rate and magnitude of error changing over time. We will be conservative in changing the acceleration value, but if it is significantly wrong, we will make larger corrections.

Temporal Correction

Implementation Details:

Based off bed angle and direction of offset, we can tell if the tracker is behind or ahead. Similarly to spatial correction we maintain a stack. One stack contains offsets, but here ignoring the corrections. Another stack maintains a boolean saying whether we were ahead or behind

After 4 inferences, if the tracker is consecutively ahead or behind we will make a modification to the acceleration value.

We simply make a very small adjustment to the value.

3/24 - 4/1

- Reduce cropping size
- Adaptively adjust inference rate and acceleration constant based on error
- Run multiple inferences at a time
- Increase minimum confidence for inference
- Try slightly adjusting angle for more robustness in measure_diameter algorithm

Gparser autocorrect problem

I've found that through increasing the minimum confidence of inference to 0.5, the inference is literally perfect.

Yet, the autocorrect still jumps around. Much less, but still unacceptable.

I've concluded there can only be one cause of this bug:

When the tip moves really fast from one spot to another,
My parser is not perfectly temporally aligned with the video,
So when running an inference at those moments, it looks like the parser needs to be corrected

However, a distinction must be made between spatial and temporal errors.

By spatial I mean my tip tracker is off by some distance. Temporal error refers to my parser running behind the actual video in time.

Gparser autocorrect solution

The solution to this problem is simple. We merely collect multiple frames for inference within close time proximity, rather than doing inference on one frame.

If for all inferences the tip tracker is off by some amount of pixels, there is a spatial error, which can be corrected by simply correcting the tracker spatially.

If for some frames the tip tracker is perfect, while for others it is off, then we know there is a temporal error. Based off the order of the inferences and their spatial errors, we can decide whether to increase or decrease the acceleration constant. Additionally, we can increase the inference rate.

And obviously, if all frames are perfect we make no correction and can decrease the inference rate.

Version control

I've never really bothered to learn how to properly do version control, and I somehow made a very big mess. Something about submodules, nested git repositories etc. I ended up accidentally deleting my .onnx file, and was not able to recover it. So I need to retrain a new model and add it back in. Other than that, here is the new repository on github to keep track of the code:

<https://github.com/BrianP8701/STREAM.AI>

Adjusting predicted angle

As you recall, as I am parsing through the gcode I am calculating the angle at which the extruder is moving on the bed. However, this angle does not always exactly match up with the angle that the material is coming out, for various small odd reasons.

To counter this, I will simply try multiple angles close to my predicted angle.

I am trying these multiples: 0.85, 0.9, 1.0, 1.1, 1.15

3/17 - 3/24

- Run model concurrently alongside gparser to autocorrect tip tracker
- Train new Yolo V8 model
- Update measure diameter method to account for tip location

Gparser autocorrect

It appears that there is some unknown variable that makes the printer behave slightly differently than my calculations. Thus, to account for this I will occasionally autocorrect my tracker.

Every 120 frames I will crop part of the image and run it through my yolov8 model, which will give me the exact location of the tip. I will use this to correct my tracker. I chose 120 frames.

As we plan to use this in real time, we cannot have the inference blocking the rest of the process. Here is the process to counter that: As the gparser begins, inference will begin running concurrently on a separate thread. The gparser will continue for another 120 frames, then a lock will be placed waiting for the inference thread to complete. Upon inference, correction will happen, then another inference will run, and repeat. Some microcontrollers do not support threading, but do allow for concurrency. The value of 120 can be adjusted as needed depending on inference time.

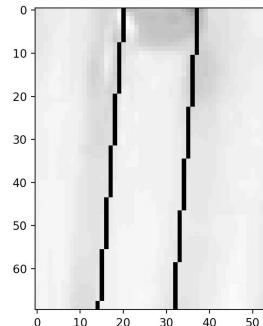
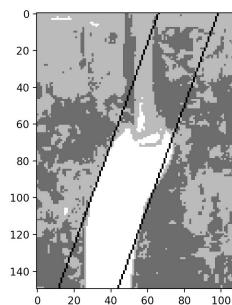
New yolov8 model

After making this program, I was confused as to why my autocorrector was causing my tracker to mess up. I realized, my old yolo model was trained on the endoscope camera view. So I've trained a new yolov8 model on 728 labeled images. 1748 images after augmentation with blurring, noise, flipping, brightness and exposure.

Also, I have found code online that does inference on the onnx model, locally, so that is no longer a problem.

Measure Diameter Fix

Previously, upon trying to assign confidences to lines my algorithm would occasionally choose the wrong lines. I have concluded that the only way to counter this is through knowing the exact location of the tip and assigning higher confidences to lines within reasonable distance to the tip, while exponentially decreasing confidence as we begin to look at lines that are surpassing what I consider a reasonable distance (About 25 pixels, or 1.6 mm radius). This method works very well.



Looking forwards

I believe this is the final step to measure the diameter in real time robustly:

I will need to use the autocorrect to not only correct the position of the tracker, but to also adjust the acceleration constant. If the tracker is often falling behind, increase the acceleration and vice versa.

I think I am ready to begin thinking about the next phase of this project. After fixing up the autocorrector, and adding in this final feature I should be able to robustly measure the diameter extruded material during a print in real time with the following:

Constants:

- mm to pixel ratio (changes with camera angle)

Adjustments:

- Integrating my algorithm with real time video and aligning the start of my algorithm with the start of the print

Additionally, we will need to add a second camera angle eventually, which shouldn't be difficult from my standpoint, as it just requires switching views when the printer is moving in a certain direction.

3/3 - 3/10

- Use algorithm on different videos
- Use time data
- Measure diameter

Gcode Parser (Tip Tracking)

Observations: While tracking the tip, when we get to building the “cube”, where there are many changes in directions, the tip tracker would run ahead. Upon analyzing the video from last week I noted that the tip tracker was lagging behind at the start and ahead by the end. Thus, I concluded that acceleration is in fact not negligible. The gcode states that the acceleration should be 1000 in the x and y direction and 200 in the z direction. However, upon experimenting I found that an acceleration of 64 best fit the video. This provided a perfect tracker for that video.

So we now have two variable constants, which can be treated as one, the acceleration and the constant by which we multiply the move times. Furthermore, these constants do not work upon other videos. It is very close, but upon letting the tracker run for a couple minutes it runs ahead or falls behind by an unacceptable distance.

I am not sure what is the cause of this. My acceleration and gcode parser is certainly correct. I've spent many hours reviewing, checking and testing my code. Additionally, the data from UBox relating to the time is not accurate enough for 1/30 of a second moves.

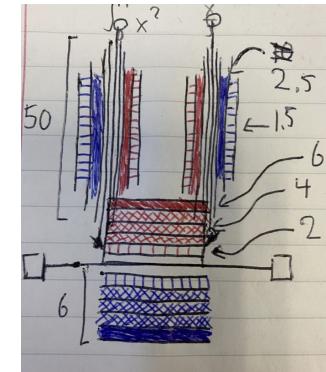
TLDR: I do not know where the error in calculating the time of each line comes from.

Table of experimental data

Video 1	fps	time_k	acceleration (mm/s)		
test_V1	30	1	1000		Slow at start, fast at end
test_V2	30	0.993	1000		Too fast
test_V3	32.65	1	1000		Way to slow
test_V4	32.65	0.993	1000		Way to slow
test_V5	30	0.99	100		Slow at start, fast at end
test_V6	30	0.99	1		Acceleration wayyy to slow
test_V7	30	0.99	50		Too slow all throughout
test_V8	30	0.99	75		Behind by the same amount, a little
test_V9	30	0.988	70		Behind by the same amount, better
test_V10	30	0.985	65		Way better, gets too fast at end
test_V11	30	0.985	60		Almost perfect
test_V12	30	0.985	64		Perfect
Video 2	fps	time_k	acceleration		
test_V1	30	0.985	64		Good to start, fast at end
test_V2	30	0.985	58		Slow to start, fast at end
test_V3	30	0.983	55		Perfect

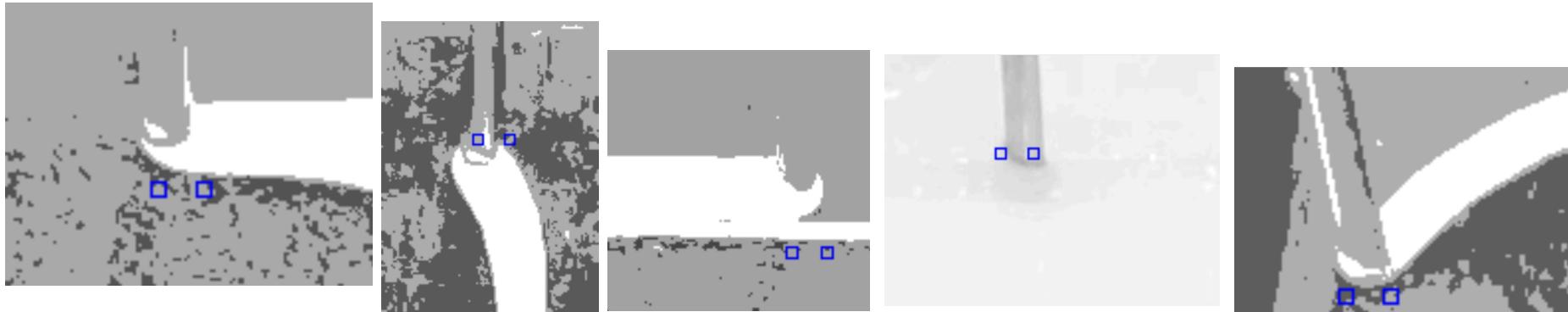
Edge Detection

By locating the exact location of the tip, we have a better idea of where the extruded material should be. Find tip by convolving over image looking for the highest confidence of the outline of the tip. Exactly from the center of the tip, draw a line out in the direction it came from. Shift that line from that point in both directions to find the edges. If there aren't two lines that surpass a minimum confidence, then we assume there is no material. Otherwise, simply calculate distance between two lines to find diameter. We can't look at brightness, because sometimes another area might be brighter. This is the most effective method I've thought of.



Edge Detection

I have created the tip detector I mentioned last slide, and it works, but not perfectly and the runtime is quite bad, taking around a second per inference. Here's some images. You can further see that the preprocessing step works very well. Even on the white image, it has very clearly defined the line, even though it's hard to see with the human eye (Difference in intensity of color doesn't matter, just difference in color.)



Edge Detection

If I could get the time data to a higher degree of accuracy (I emailed you about this) then the tip tracker would very accurately be able to get the point of the tip. Otherwise, I'd have to fix the algorithm from last slide. The next step is to draw a line in the direction we came from, and then shift it in both directions to find the best edges. It's important that the actual tube of the extruder is not included in the cropped image, as it will interfere with edge detection.

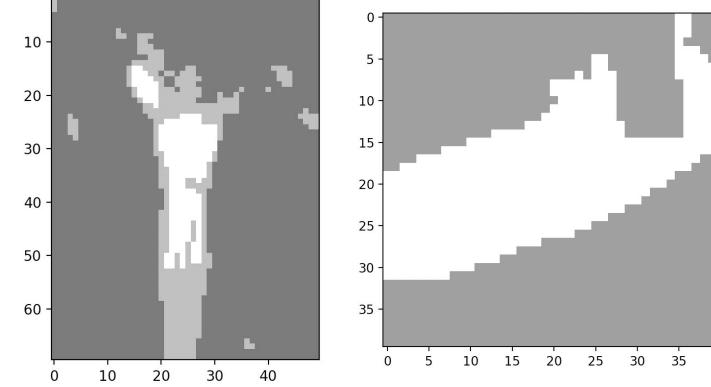
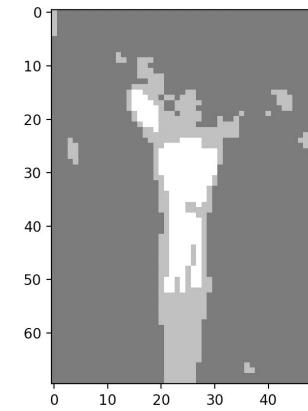
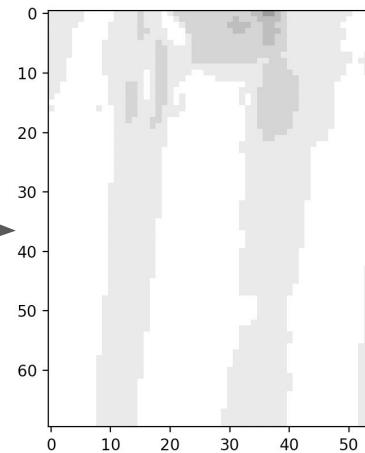
Note from future: What ended up happening is that I used more yolo v8 models to do inference to self correct, which I talk about in the future.

2/24 - 3/3

- Made good pre-processing step
- Edge detection
- Fixed gcode parser

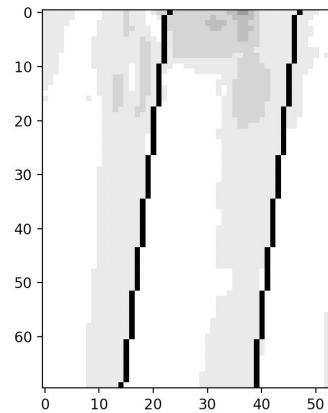
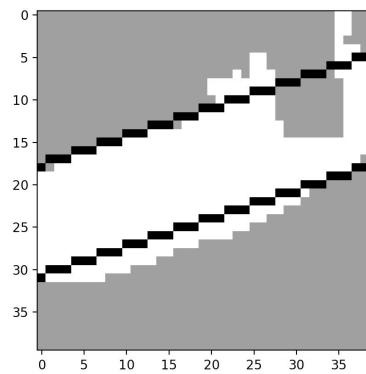
Preprocessing

Before moving further, we want to get rid of the noise and sharpen the edges. We can simplify the image by reducing similarly colored pixels to the same value with respect to the variance of the color in the image



Edge Detection and Diameter Measurement

Given the image and angle of movement, I check all lines in that direction. I measure confidences for each line, and added some extra fine details I found while experimenting. It's not perfect yet - still working on it.



Still working on
how to
overcome
some
situations like
this.

Gcode Parser

I fixed the gcode parser, but I have no math or any reasoning to back it up. It's quite accurate now. It turns out that the real time is 99% of the calculated time. I made a few more additional tweaks, and that fixed all the problems. Acceleration turns out to be negligible.

Note from future: This leads into what is the overall problem I seek to address later. Across different videos, there seems to be a different constant % of the calculated time I must find.

2/17 - 2/24

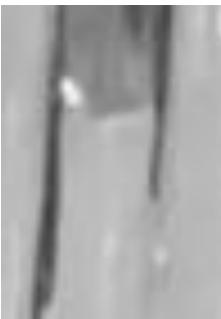
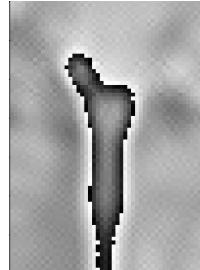
- Tried convolving matrices over images of extruded material
- Finished bulk of gcode parser, debugging
- Planned algorithm for edge detection, taking inspiration from this paper:

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8607091>

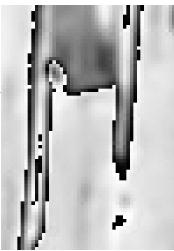
Tried convolving different filters over images



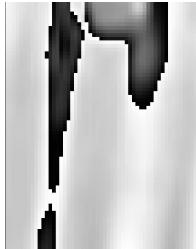
Gaussian Blur, Sharpen3, Sharpen2



Sharpen1



Long1



I tried convolving different matrices over the images, looking to find a combination that would work consistently across the variety of images. These 4 filters I found after hours of trying different combinations, and would only work for that specific image. This is a very laborious process, and it's actually what CNNs do, learning the best kernels for the image. It was pretty fun and helped build my intuition for CNNs more.

Algorithm for detecting edges

To measure the diameter of the extruded material, we need to accurately identify the edges of the material. We do not know where this is beforehand, or if it's even there. However, we do know the angle at which the material should be coming out given the gcode.

What we can do is treat this as a complete search problem, with all the lines of that angle crossing the image being the search space. We can slightly offset the angle in both directions, and after searching the entire search space identify the two lines that have the greatest probability of being the edges of the extruded material.

We have to account for the lines from the actual tip, which we can locate and factor into our probability calculation.

2/11-2/17

- Make gcode parser and tip tracker more accurate

Improving gcode parser and tip tracker

- New Challenges:
 - As we move in the z direction, the x axis will stretch, changing the ratio of mm to pixels.
 - To estimate time for a line of gcode to be performed, we cannot just divide distance by velocity. We have to account for acceleration and jerk.
 - Even though the printer isn't changing its speed, when it changes directions it needs to accelerate in that new direction.
 - I found this quote online: "I have found that the biggest error between the predicted time and the actual time has been the time the machine spends processing the instructions." I don't know if this is true, but will leave it here.
- Relevant links (For myself, no need to look at this):
 - <https://3dprinting.stackexchange.com/questions/10369/why-does-jerk-have-units-of-mm-s-rather-than-mm-s%cb3>
 - <https://3dprinting.stackexchange.com/questions/3233/calculating-the-estimated-print-time-of-an-already-sliced-file>

Gcode parser and interpretation

- I wrote a working algorithm that reads gcode and tracks the movement of the tip accordingly frame by frame
- However, the camera shakes hundreds of pixels in unpredictable directions. I will need to record new videos where the camera does not shake so much.
 - Note from future: This turned out to be a result of the fact that the printer can move in the z direction, which I didn't account for.
- I made my code such so it's easy to adapt to different types of video angles, zoomed in or out and FPS of video.

```
# Arguments:  
#   g_path    - Pathname of .gcode file  
#   fps       - FPS of video  
#   mTp      - Ratio of 1 millisecond to pixels  
#   sX        - Initial X coordinate on image  
#   Y         - Y coordinate on image  
#   tempX    - Initial x coordinate on bed  
#   tempY    - Initial y coordinate on bed  
#   tempZ    - Initial z coordinate on bed  
# Returns:  
#   A list containing the bounding boxes for tip of extruder  
def interpret(g_path, fps, mTp, sX, Y, tempX, tempY, tempZ):
```

Plan to measure diameter

- Make tracker that reads G-code. Tracker should output direction and speed extruder is moving at each frame.
- Based off speed and direction, crop a portion off the frame accordingly.
- Convolve over cropped subsection to detect edges and measure width or area of material.
- Output diameter of new volume extruded for each frame.
- Make code maintainable so we can easily adapt to different camera angles and add a second camera.

2/4-2/11

- Plan design for algorithm to measure diameter of extruded substrate

<https://hackaday.com/2016/02/05/filament-thickness-sensors-what-are-they-and-what-are-they-good-for/>

Topics Originally Planned

- Hash space anomaly detection (classification) based on Jackson DIW data/new data
 - Preparation: learn from any image classification examples
 - First, CNN for image based consistency classification (can be either at the extrusion point or better at the substrate), may need data augmentation
 - Later, classification after deep hash (see the paper by Huining Li)
 - Can be put on Arduino/Pi later
 - The above steps can be applied to sensor data too
- Hash space controller based on DIW images for consistency
 - Preparation: learn image feature extraction, regression examples
 - First, extract the diameter info during the printing and compare it with the desired value in real time, obtain the deviation.
 - Then, build a regression model to link material and printing parameters to the deviation.
 - Finally, build a PID controller for the deviation minimization.
 - Later, control under deep hash
- Attack to controller under hash space
 - Preparation: know the concepts of controller attacks
 - First, design detection mechanism to the attack
 - Second, in the network setting, perform the diagnosis