Fusion-VR Computer Vision 2022 Legacy Manual

By: Jeffrey Liu

Foreword

Hi! My name is Jeffrey; I was the previous intern for the Fusion-VR project back in Spring/Summer 2022. This document will provide you with a summary of all of my work that I have done, catch you up to speed on everything about the code, and let you know of all the immediate plans that I did not get to. I essentially scrapped and rebuilt the entire vision pipeline that the previous interns had worked on in order to have a much better performance, so make sure to at least look through the documentation!

For context, I am definitely not the best engineer. I'll start talking about things like OpenCV and TensorFlow, but I can assure you that I am **NOT** an expert in them. In fact, I only came in with knowledge of OpenCV and had to search online a ton about TensorFlow since I'm very new to machine learning. If you happen to have a lot of experience with either/both of these libraries, don't worry that we don't agree. Either I am wrong and there is a better approach, or there is room for discussion.

In any case, the current system is *basically* working (with the exception of a few bugs and improvements to work on). Your goal will be to scrap all of my work if you're confident in a better system, or optimize mine as much as possible (it's not very optimized).

This document will only be focusing on technical aspects, design choices, and walking you through what I have built. Throughout the documents, I will mention bugs, possible improvements, or places where I just wasn't sure what I was doing, so try to read all of it and take notes; I'll try to make it as short as possible! The previous intern didn't really leave sufficient documentation so I am making this to make up for it. **In this document, I will assume that you have complete knowledge of the Fusion-VR research project.** This includes experimental setup, BEAM/N-BACK tests results, what our goal is, etc. If anything is unclear, ask Mark or Sarah about the details. Don't hesitate to contact me as well if you have any questions or want to ask about anything about the code that is not mentioned here. I graduate in 2024 so I'm not leaving anytime soon. You can reach me at jsliu@ucsd.edu to ask questions, schedule a time to meet, complain about the project, compensate me, or make a friend! I expect that there will be some transition meetings once you are on-boarded anyways, so don't be afraid to reach out and ask for help. I'd be more than happy to come back for a bit and help out.

Good luck on the project! It is very interesting work. Mark and Sarah are great people and super smart. I had a lot of fun working on this project, and I hope you will too!

Jeffrey Liu

TABLE OF CONTENTS

I. INTRODUCTION & PREVIOUS WORK

Previous Work and Reflection

The interns before me were the first people to work on this code. They implemented a solution purely with image processing techniques using OpenCV. You can see their work in the original repository. The most up to date things that they were using is located in main.py. You can look at it if you want to understand what they did, but not necessary since I did not use any of it in my final work. I will go over the issues with their code.

I had initially tried to build off of this code by exploring more into the topics they used like color segmentation for fixed HSV intervals for things like the arrow for BEAM and template matching for avatar detection. Here is what I ran into:

- too much motion blur in between frames for image processing to be viable
- set color intervals don't work very well since some videos are more red-shifted for some reason
- arrow detection performed horrendously since they were working off contours and the arrow would sometimes blend into the sky color and the ground.

Not an exhaustive list but it was enough to make it frustrating enough to not want to continue. Again, I am not an expert in OpenCV, so this may be entirely possible with very advanced processing techniques but I would not know. I encourage you to implement a solution with OpenCV if you are up for it and scratch my work since I believe image processing techniques are way more computationally and time efficient than utilizing computer vision detection models. My current solution is rather slow, but it performs at a pretty high accuracy and does get the job done.

Overview of Current System

After failed attempts at getting the previous system to work, I decided to integrate TensorFlow to solve the issues that I was facing with the previous interns' work. The outline was:

- Obtain a custom computer vision detection model to output object locations for both N-BACK and BEAM experiments for different types of avatars and arrows
- Utilize image processing methods to extract gaze and weapon circle locations
- Store object coordinates for each video frame and export them into a CSV file
- Analyze the timings and the objects interactions with one another
- Create and generalize inferences from these interactions to reach conclusions from video test results

This document will cover how my two custom models were trained, how their locations are extracted and exported, and immediate plans that follow (along with bugs every step of the way ;)). A general overview, but more technical details will follow. All the code I have is uploaded here (**IMPORTANT: bookmark this repo this is the main repo with all of my work**).

II. Prep for Training

Getting Started

One of the first things that I got to doing was labeling a dataset to train a custom model on. I built off of a pretrained model from the TensorFlow model zoo: SSD MobileNet V2 FPNLite 640x640. The datasets that I have created for each experiment are uploaded in the Google Drive folder. Log into ettenhofer.lab@gmail.com to view. Datasets were created with the help of LabelImg. Very easy to learn if you want to add more labeled data to the dataset. Each of the datasets already have around 350 images. Each of these models correspond with different experiments and have different label maps for different objects. Set up your environment for work as well. I have set up a Google Colab ipynb that has the environment and training configs and scripts already there. There are scalability issues with Colab that I will elaborate on later. After finalizing your dataset, you create TFRecord files using record.py for our training script to use.

It's confusing at first, but I found an article that outlines the whole procedure of training really well. This basically covers almost everything I did for training. I followed this article religiously when I was trying to get TensorFlow set up. It has the file structure that I am following if you do not understand the way the files system is set up. TensorFlow Object Detection API documentation was also helpful in setup and understanding.

If you want to add more images to the dataset, I made a small script called ExtractFrames.py that goes through a video, and you just click on the button to save the frame. Do this for as many more images you would like to add, label them, and create a new record file to train with containing all images.

I also mention multiple scripts throughout the doc that I have included in the github repo. Remember to set up your local environment first. python -m venv and all that stuff. requirements.txt is included. There might be issues setting up the object detection API; if so the API docs can help.

Training Script

The training.ipynb script I linked above will be the main thing to train with. There are some niches that I won't go over such as builder.py that has to do with dependency version issues. If you are interested you can contact me, but this script has everything you need. Just run all the cells and you should be good. Remember to do captchas so your runtime session does not expire. They appear intermittently which is SUPER ANNOYING. Training for me usually took 9 hours because of how slow Google Colab is. Google Colab, even with pro version on GPU, is extremely slow and has a lot of memory constraints. I was only able to train one epoch so it isn't very robust. But then again, there is very little variance in the video data so this should be fine. If you would like more computing power, request for a gpu cluster through UCSD Datahub. I would've used this, but it was nearing the end of my internship here. It would allow for a lot more flexibility in testing and such.

III. Model Details

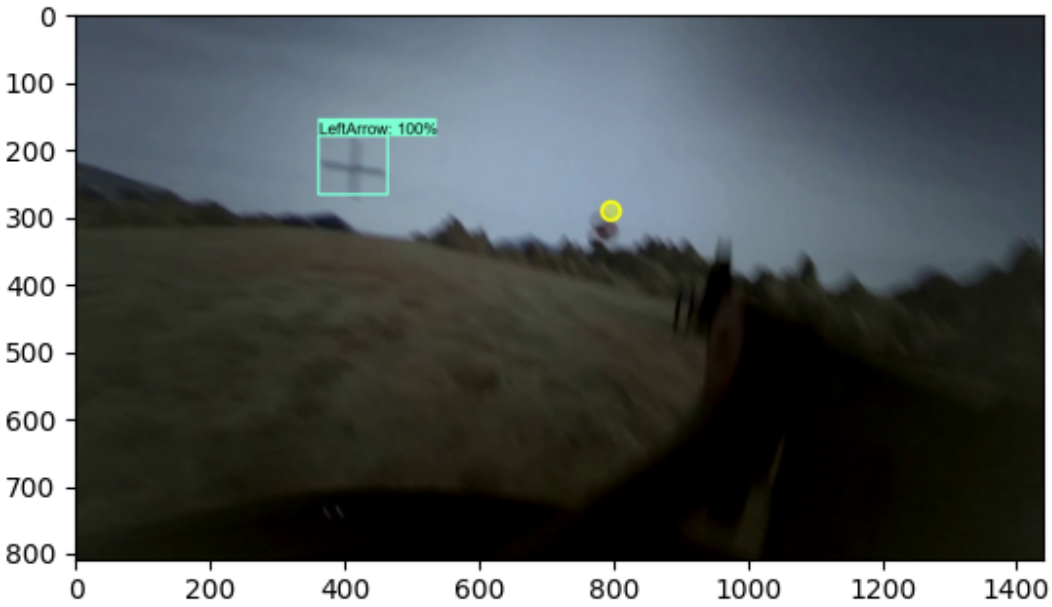
I have already mentioned most of the training details in the previous chapter, but make sure that you are training with the most up to date record files so you don't waste 9 hours. In Google Drive, I have uploaded the most recent trained models. Exported_model has training details and events that aren't really relevant, but I downloaded them just because. Exported.zip is the important file that has the most recent trained model. I will go into detail about each model.

NBACK Model

The label map should have all the things that I am identifying. The unique thing about this model is that I could not successfully train the model with google colab to identify the full body of the avatar. Instead, it is only on the colored bags since I identified that the model performed very well at identifying the type of avatar but not the avatar itself. Other than that, the model performs quite well.

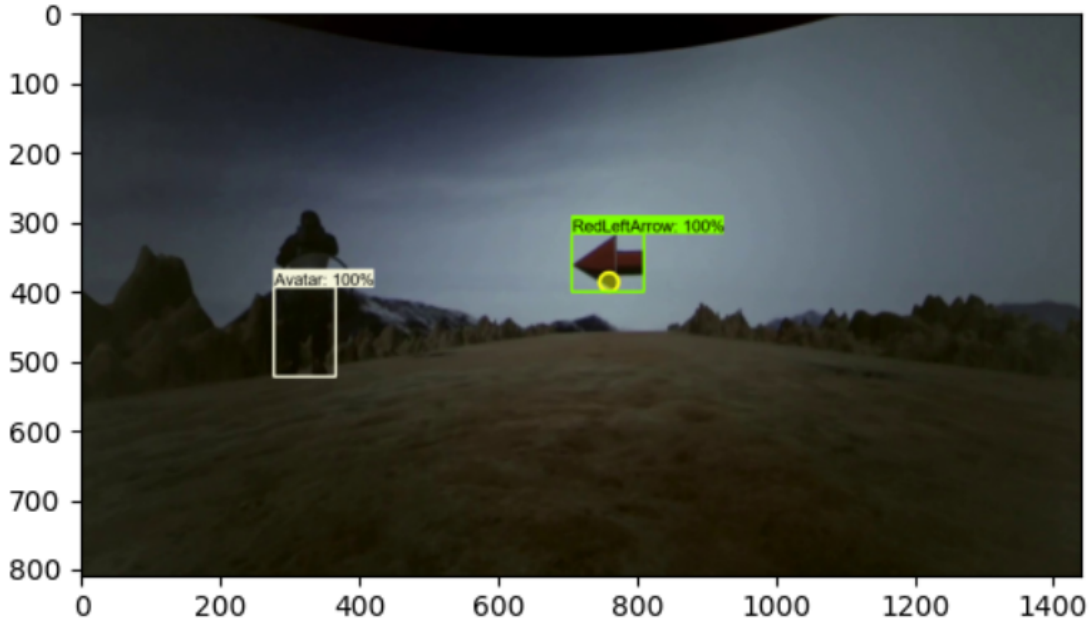
BEAM Model

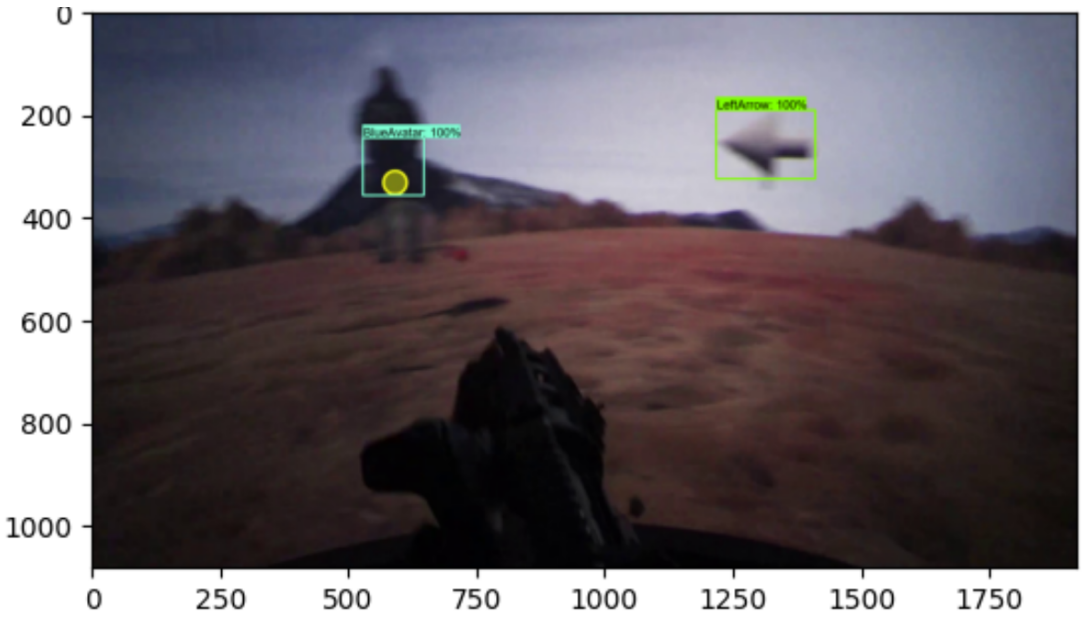
Try looking for the label map for BEAM to get used to the file structure. The difference here is that we introduce red arrows. The unique thing about this model also comes from avatar detection. Given the same constraints, I could not successfully train the model for the entire avatar, so I opted for the distinctness of their legs and it worked. There is a bug in the model though. For some reason, it is identifying the cross in the middle as a left arrow. Maybe a bad label in the dataset? I'll leave this as an exercise for you to figure out :)



Testing the Models

In order to test the models, there is a script called inference.py you can run for a folder of images. Try to take a few new images with the extract frame script that the model has never seen before and save that into a folder. Run inference.py on that folder and it'll perform inference on all the images. This is one way to manually validate the model. Try it later to verify that you understand how inference works so that you may use the result for analysis in your own work this summer. Here is an example of what inference.py spits out for both models.

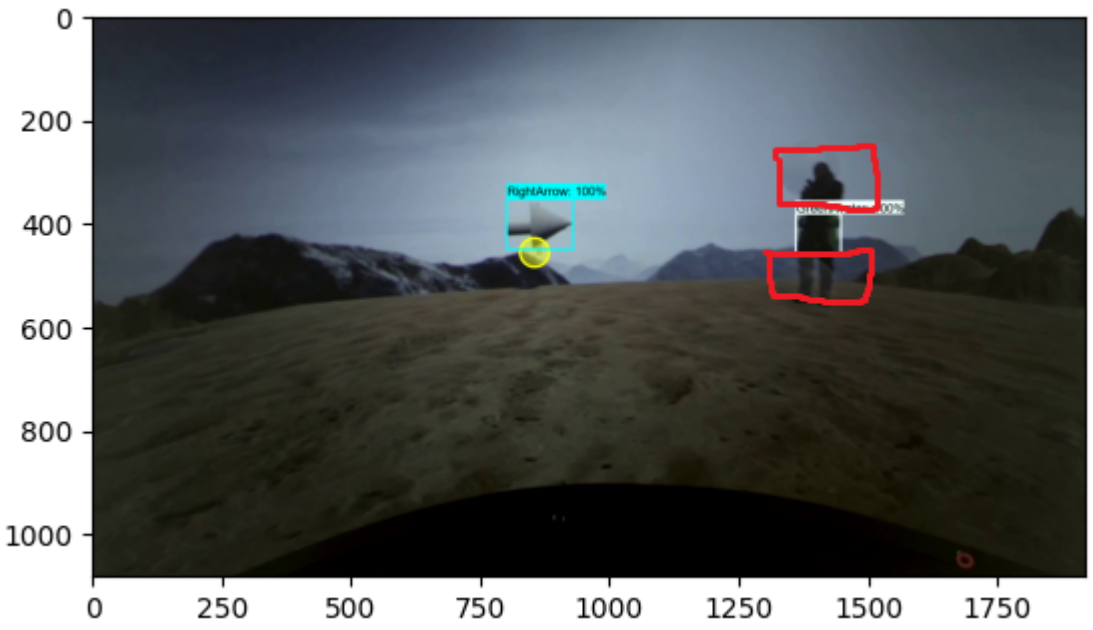


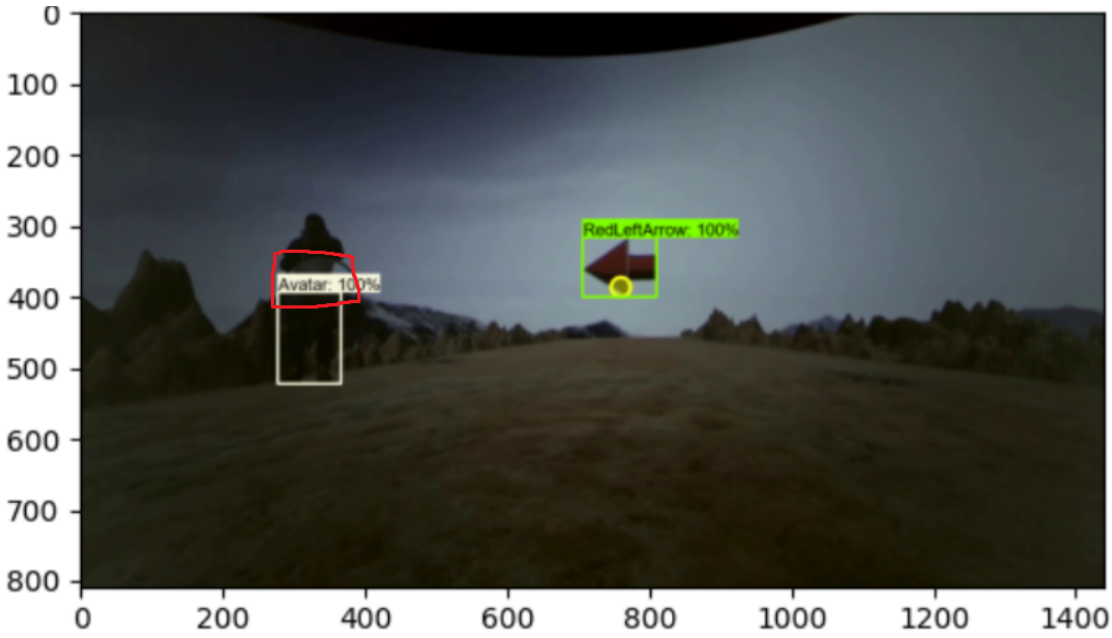


IV. Object Location Extraction

Nothing much to write about this. If you understand the inference script you most likely will understand data_extraction.py which will export all the locations into a CSV file. This one just goes through each frame of a fed-in video and writes all the coordinates of the bounding boxes. They are in the format ymin, xmin, ymax, xmax and are normalized to unit length. An example output for a video is provided in the folder video_exports.

One thing that absolutely needs to be done is figuring out the estimation of the avatar body since both models, at the moment, only capture a portion of its body. Since both experiments have their own procedures on where the patients point their gun to, figure out thresholds where the box should end. Decide for yourself where you want to make this statistical analysis either before you write it to the CSV file or while you are parsing it. Completely up to you now how you want to analyze the data. Distance from the screen is relatively fixed so work out with Mark and Sarah in a separate meeting where bounding boxes should end i.e., where the test subjects are allowed to aim in to be considered as a hit. I will paste an example below explaining what I mean. I'm kind of bad at explaining, so if it is unclear I'm pretty sure Mark and Sarah can explain what they want since we discussed this extensively. Of course this is all assuming you keep how the current model trains. Ideally, maybe try making a model that identifies their whole body if possible.





Another improvement that you can make is on the output script format. I wrote this script at the end of my time here, so maybe you can help cleanly format the bounding box coordinates. Definitely not necessary though. You could go without doing this.

V. Future Work

In summary, this was the entire process I have discussed:

- Set up python virtual environment to begin developing
- Label images to create your dataset and package them into TFRecord files for test and train
- Train on Google Colab and export the model
- Test the model with various inputs
- Use the model in your location extraction script

This is where you come in. You have two options here:

1. Trash my work or backtrack a couple steps. Start over because you are confident in another solution that may be more optimized and efficient, or because I took a wrong approach somewhere. Nothing wrong with this option, I would prefer it actually if you are an expert in image processing or machine learning and can create something smarter.
2. Continue my work. I will describe what I think will be a good plan below.

Future Goals

Here are some notes that I believe need to be done or would be nice to be done. I hope I am not forgetting anything important. This is in no particular order. I would like you to please discuss these with Mark and Sarah to determine what is best for you and the next best option for the project. Mark and Sarah, if you are reading this: HELLO!

- Work on detection for gaze circle and weapon circle. Gaze circle can easily be done with color segmentation with set intervals since these are edited in. Weapon circle, I predict, will be very tricky. It's extremely small, blends in well(and worsens with motion blur), and hard even to detect with human eyes sometimes. Color segmentation will be tricky with this, and computer vision will be hard for something as ambiguous as a blurry red dot, though maybe with a transition to Datahub we could train more epochs and develop something that can fully rely on the object detection API.
- Develop the analysis script to determine interactions. Most interactions will be just collision detection between gaze circle and avatar, weapon circle and avatar, etc. This is a simple collision detection problem on a 2d plane, like intersecting rectangles on an x-y cartesian plane. Other things include, for example, a right arrow with an avatar on the left, or weapon circle on the head instead of the foot. You can ask Sarah about other things you can interpret. Again, everything in analysis is really just location/collision and timing based (videos are 60fps btw) so it should be very simple. You'll also need to integrate the weapon trigger sound. Ask Sarah about analyzing this.
- Fix the bug that detects the cross as an arrow in the BEAM model.

- Finish bounding box estimation for avatars in both models as described in the previous chapter.
- Find a smarter way to export object locations i.e. format the CSV file in a better/intuitive way. I would describe the way I formatted it as, quite frankly, really stupid. It will make parsing easier. If you are fine parsing the CSV file as is, more power to you :D
- Blackbox the entire pipeline. I didn't have time to add argument parsing to my programs. I was planning to do it using argparse, but I didn't have time. Fall quarter Week 0 vibes. It is a very simple task though; just make it so that the people who are going to use data_extraction.py don't need to know what's going on. It should look something like this

  python data_extraction.py --model=[MODEL PATH] --video=[VIDEO PATH] --csvname=[NAME FOR NEW CSV]

  Maybe in the future for your analysis script you create you could write something like

  python data_analysis.py --experiment=[BEAM OR NBACK] --csvname=[CSV WITH OBJ LOCATIONS] --interactionsfile=[SOMETHING TO REPRESENT ANALYSIS]

- OPTIMIZATIONS PLEASE: data_extraction is really slow since there are so many frames in one video. Inference doesn't take long on a single frame, but it really starts to add up. Think of optimizations. Porting to C++ is a very definite last step. Think of ways to optimize at the system level. I admittedly didn't know how to choose a good ML architecture. I just chose MobileNet since low memory usage worked for Colab. Think of things like that.

Conclusions

This doc is meant to be comprehensive, and more as a refresher and where to get started. It'll be hard to read through the entirety of this on your onboarding week, understand everything, and take this as an introduction to your whole summer. Look over the doc throughout the first few weeks that you are understanding the system. Within those few weeks, I recommend asking a lot of questions about the experimental procedure to gain an understanding of what you are analyzing. Contact me for any questions or anything you are unsure of. I think most of my inspiration from my solutions came from actually understanding what I was trying to implement so I could come up with loopholes that sped up my progress. Good luck on this project!