This document serves three main purposes. First and foremost, it should roughly outline some key points of how this project came to be, the setbacks, information learned, and where it is going. Secondly, I hope it shows that I have very little idea what I am doing such that thirdly, it lets others looking to get into the fields of data science and machine learning see that it isn’t that difficult to get started on a project, and that every failure or setback comes with information to guide you further.

The original idea driving this project was to improve upon the project [Mediapipe-VR-Fullbody-Tracking](https://github.com/ju1ce/Mediapipe-VR-Fullbody-Tracking). Compared to more advanced (and more expensive) fullbody tracking systems, Mediapipe-VR uses a single webcam input into a 3d pose estimation model that outputs keypoints for tracking a human body, and then converts those keypoints into steamvr coordinates for use. The original target of this project was to use assumptions we can make about tracking a pose for vr in order to improve the general model in mediapipe.

Some of these potential changes are:

* The camera can be assumed to be stationary, allowing for easier background removal and focusing on the target person
* We can assume there is only one person in a given image
* Via the headset and controllers, we already have 3 keypoints that can be passed into the model, and keypoints could be measured against the headset as the origin point

Additionally some behaviors of the current model that could be improved are:

* Reducing keypoint ‘jitter’, as in vr this can contribute to motion sickness
* When crouching down, your feet may slide out from under you
* Loss of tracking of a point can result in strange behavior

A potential solution to some of these problems may be to train with a loss constraint that looks at the pose over time and weights the loss of moving around the ground truth along with the distance to the ground truth points. Effectively, we want the model to not be penalized too harshly for being a few centimeters off, as long as it isn’t jittery.

Finally, I wanted to keep the stretch goal of a multi-camera setup in mind, which would allow running two or more models (or a specially designed singular model) on multiple camera views, which could vastly improve prediction quality by accounting for occlusion from different angles.

Because of these constraints and modifications, the primary focus of the project became to procure a dataset to start with. Creating a real dataset was impractical, as it would have required many expensive fullbody tracking setups and camera setups to put together, and would require a massive scale to create a dataset large enough to train or even finetune a model. It may have been possible to take an already existing 3d pose dataset and reframe it for this project's purpose, but this makes multi-frame and multi-camera models significantly more difficult or impossible, and reduces flexibility to choose keypoint structure. A final option, to create a synthetic dataset, became the backbone of this project. Simulated synthetic datasets have become more popular as assuming you can capably simulate a real dataset, you can infinitely generate randomized samples for training. In our case, building a synthetic dataset also gives us the flexibility to choose whether or not to include multiple frames per sample, multiple camera angles per sample, and tweak other various parameters that may help in building a robust model.

In order to build this synthetic dataset I used the Unity game engine, along with Unity’s in development Perception module, which is designed for leveraging the game engine for creating synthetic datasets. [PeopleSansPeople](https://github.com/Unity-Technologies/PeopleSansPeople) is a project based off of this that handles generating a dataset for 2d pose estimation, as well as image segmentation, and so worked well as a base for the dataset. This repository is thus dubbed PeopleSansPeopleVRPose, which represents the data half of the project, with the VRPose model itself being as of yet incomplete. One difficulty I ran into here was reducing the number of unusable images in the dataset. By default PeopleSansPeople is designed to track many different characters and objects at once, in a much more chaotic environment in terms of lighting and clutter. For the dataset I wanted to generate, I only want to track one character in a scene at a time, and don’t need as chaotic of an environment. Post-processing, average lighting, and small translations and rotations of the camera can all be simulated after the fact as part of the dataset loading process in tensorflow.

In its current state, there are still issues with the camera angles and distances, as well as some post-processing and lighting results that create almost entirely black or white images, which will be useless in training. There are plans on how to go back in and fix these problems within the unity environment, but for now there is code designed to filter these out upon loading the dataset in python.

With a basic version of the dataset working, moving over to the modeling side of the project we see where the overambition of this project really shines through. I have a fair amount of experience with tabular datasets and machine learning concepts, but for image processing networks, I didn’t know much beyond the basics. I had to do a lot of reading to catch up to modern solutions for image based networks, and then familiarizing myself with implementations of these in tensorflow. I build up a massive backlog of potential ideas for solutions or interesting architectures to try out, and effectively what I will be doing as I continue working on this project is exploring many of these ideas in order to develop a more implicit understanding of what works and doesn’t work for image processing as a whole as well as 2d and 3d pose estimation.

Some of these ideas are:

* Fine tuning a pre-trained mediapipe model on the new data
* Modifying the mediapipe model to account for the keypoints we have (headset and controllers)
* Explore vision transformer model architectures
* Explore tiny-ML vision architectures
* Use a 2x1 hybrid heatmap for training and prediction (ideally faster than full 2d, but still accounts for spatial information in the image
* Use a physically constrained humanoid model to put in place of the keypoints, and instead of directly predicting keypoints, predict the movement necessary in the model to end up with an accurate pose.
* Use an intermediary model to predict an entire pose at once, and then the secondary model takes the information regarding that pose and predicts the individual keypoints
* Use an initial instance segmentation to remove the background and focus on the character, and then predict depth of the image across the segmentation mask, then use the depth mask to fit or restrict possible poses before the final keypoint predictions
* Use specialized loss functions designed to target and penalize specific behaviors in a model.