Estimating Walking Human Figure Trajectory using Figure Poses

Jonas Hirshland, Shruti Ambekar
University of Michigan, EECS
500 S State St, Ann Arbor, MI 48109
<jhirshey, ashruti>@umich.edu
https://gitlab.eecs.umich.edu/jhirshey/eecs442_project

Abstract

This report will explain the research expanding on Carnegie Mellon's OpenPose software to determine future poses of walking human figures. Some key experimental factors discussed in the paper are frame per second, different orders of linear regression, and methods of grading estimation performance.

1. Introduction

Computers will soon be able to understand motion in the physical world better than humans. Estimating the path of a home run is already a task that computers are suited for, but harder tasks like estimating the trajectory of animate objects is also on the horizon. Carnegie Mellon's OpenPose software is a state-of-the-art pose estimator that estimates the orientation of human figures within an image and was very helpful. In our final project, we used a sequence of poses of a walking figure to estimate the next pose.

1.1. Problem

Computers have a limited understanding of the way animate objects interact throughout the physical world but must be able to predict human behavior if they are going to interact with humans. Some places this concern is especially poignant is in factories where the future relies on human-robot interaction, and in the autonomous vehicle industry. Knowing where someone is going leads to an understanding of human intention. This is not a completely unsolved problem, but it is one whose solution is increasingly more accessible due to advances in machine learning and open source software such as OpenPose.

One indication that estimating human kinematics is solvable is that athletes are constantly doing so when they interact with another player in their sport. Take the example of tagging someone out in baseball, a field player is solving a collision problem between his hand holding the ball and a running opponent.

Though problems computers and humans are better at solving different problems, there is increasing evidence that most problems humans can solve, can also be solved by computers. Lots of examples that are thought to be uniquely solvable by humans, like object classification or detection, are being solved by machine learning.

1.2. Data Collection

In order to collect data to estimate future poses, we simply took camera footage of people walking with cellular phones. Any sort of digital source like a dashcam, CCTV, or broadcast could also be used. We used Open Pose to process the images that we captured to give us poses of human figures in the image.

1.3. Approaching a Solution

Our first intuition to solve this estimation problem was to simplify the data into something more approachable. We processed three sequential frames, and tried to use two to predict the third's pose using simple linear fitting. We then moved forward to polynomial fitting to estimate the next pose. We continued to test more variables and tune the estimator to get the best estimation possible. Now we will explain a little more about the specific method and factors we used to come up that estimation.

2. Estimation Methodology

In order to give an overview of how each estimation is computed, the next few sections outline each step of the process to go from a picture to a future pose estimation.

Key point: a human joint that is paired with a 2D coordinate.

Paired Key Point: set of an individual key point at multiple consecutive frames.

Pose: a collection of 25 key points that collectively make a complete human figure

Trajectory: the next estimated pose/point derived from at least two previous poses/points.

2.1. Run OpenPose on an Image Sequence

Open Pose takes JPGs as input and will output 2D and 3D key points of many figures in a frame.

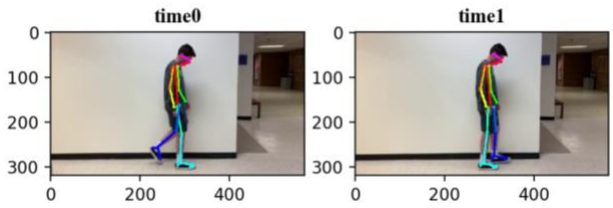

Figure 1: Visualized OpenPose results between two consecutive frames

2.2. Fit a Curve for Each Paired Key Point

We used numpy.polyfit(x, y, degree) in order to fit the curve of the key points in two-dimensional space. We believed that a polynomial fit would be sufficient to estimate the curves outline by the movement of joints.

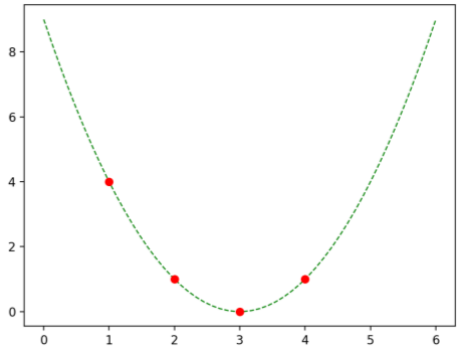

Figure 2: The red dots are the paired key points set and the green line is the np.polyfit() result plotted in the linear space of [0,6].

2.3. Determine the X-Coordinate Along the Fitted Curve

In order to estimate the x-coordinate that should be plugged into the fitted polynomial curve, you would add some estimated dx to the x-coordinate of the last point. The simplest estimation of dx is the distance between the current and previous x-coordinates.

$$x_{next} = x_{current} + dx,$$
$$dx = x_{current} - x_{prev}$$

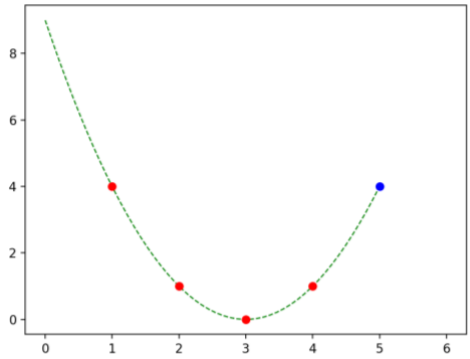

Figure 3: Plotting the estimated trajectory where the blue dot is the estimated next keypoint.

2.4. Measure the Estimation Accuracy

In order to grade the estimation, we calculated the error by looking at the difference between the estimation and its ground truth. The ground truth is the actual key points that OpenPose calculates in the next frame. Error was defined as the following:

$$Error = Euclidean\_Norm(keypoint_{est} - keypoint_{GT})$$

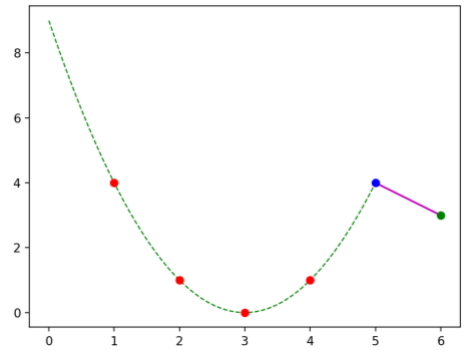

Figure 4: The ground truth key point is the green dot and the Euclidean norm is depicted as the magenta line between the real and estimated points.

2.5. Combining Into one Trajectory

Polynomial estimates were made for each individual paired key point of a full human pose and combined to make a full new estimated pose. The combined error is the summation of the individual errors, and gave us a concrete comparison between how good an estimate was in addition to visual cues that we could look at.

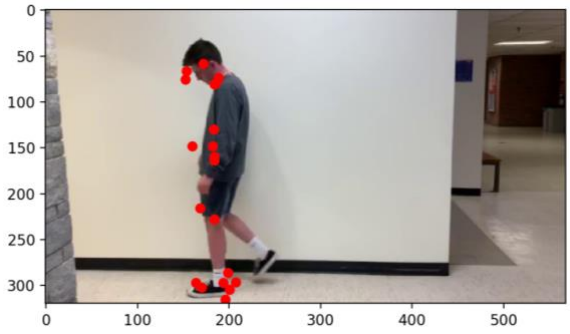

Figure 5: This shows the previous frame and the estimated next pose.

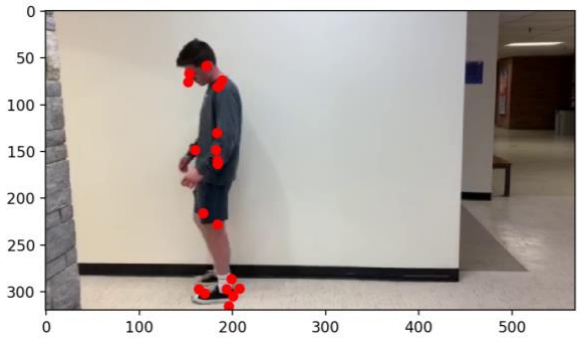

Figure 6: This shows the same estimated pose and the actual ground truth human figure.

The estimated pose is very close to the real one. The trajectory was a good estimation. You can particularly look at the estimated right hand and right leg that you could probably estimate if you knew the figure was walking forward. This was a particularly good estimation. The error was estimated at ….

3. Experimental Factors

There are several variables we tested while making our estimation of the trajectory of a walking human. 1. Frames Per Second 2. Number of Frames Considered 3. Order of Polynomial Fit 4. dx Estimations.

3.1. Frames Per Second (FPS)

The frames per second can change your estimation because the time between paired key points can alter the accuracy of the curve. In an ideal world, you might want to have the maximum FPS possible. That way, your curve should be the most accurate. However, because the key points take time to compute, a real-world solution would likely want to reduce the FPS.

When having too high of an FPS, we realized that the key points were extremely unrelated to each other. Also, using every frame would be too computationally expensive, and prohibit any chance of a near real-time trajectory estimation.

In our experiments, the iPhone X camera took footage at 30 FPS. We tested using every 3rd, 5th, and 7th frame or 10, 6, and 4.3 FPS, respectively.

3.2. Number of Frames Considered

At least two time periods are needed to estimate f(x). If all of the key points between time frames are correlated in a function, the more opportunity for a better fit curve. However, if the points are not a true function

There is also a time tradeoff because if you have more than **x** frames, np.polyfit() begins to take a long time to fit a curve to all the points.

3.3. Order of Polynomial Fit

The order allowed to be used in the polynomial fit is bounded by the number of previous frames considered. If, for example, you take into account four frames, than three is the largest order to get a mathematically sound fit.

3.4. Dx Estimations

To review, the dx estimation gives us what input x to put into the estimated polynomial curve f(x). The dx estimations we check for were a simple $dx_{next} = dx_{previous}$ and a complex dx, calculated by fitting the curve of the dx_t values with respect to time.

3.5. First Factors Used

We started off with 7 FPS, 2 previous frames considered, and a linear fit. This was the base case, and easiest to compute an estimation from.

3.6. Continued Experimentation

We continued to experiment by ranging through values of all different values of the factors, and plotted histogram to see the effectiveness of each of the solutions. Next, we will analyze those results.

4. Results

4.1. Results of Different FPS

We skipped and checked the results of three, five and seven frames skipped between, giving a resulting FPS of 10, 6, and 4.3 FPS. Sometimes, different FPS might be unattainable due to hardware and/or time constraints, but we assumed that we didn't have any. If you are using a webcam on a laptop computer, for example, higher FPS processing would probably reach a bottleneck when running OpenPose. If high FPS seemed to be such a large factor, you could reduce resolution to also get the high FPS needed for a more reasonable estimation.

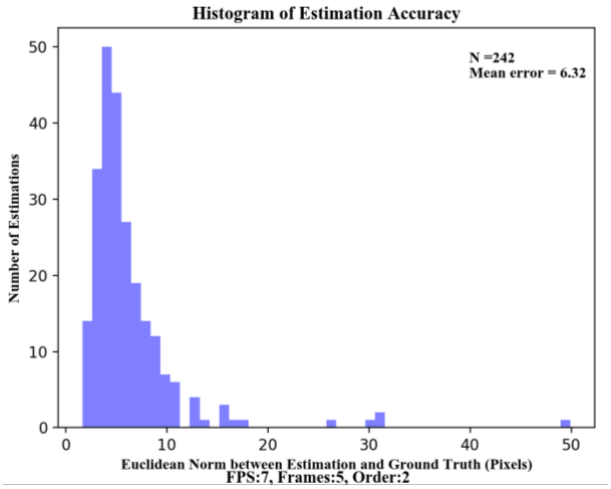

Figure 5: Histogram of error for 3 frames skipped (10 FPS), where the mean error is 6.32 and 242 trials were used.

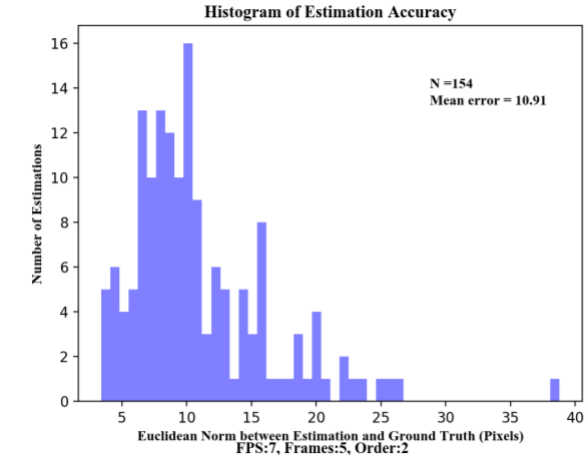

Figure 6: Histogram of error for 5 frames skipped (6 FPS), where the mean error is 10.91 and 154 trials were used.

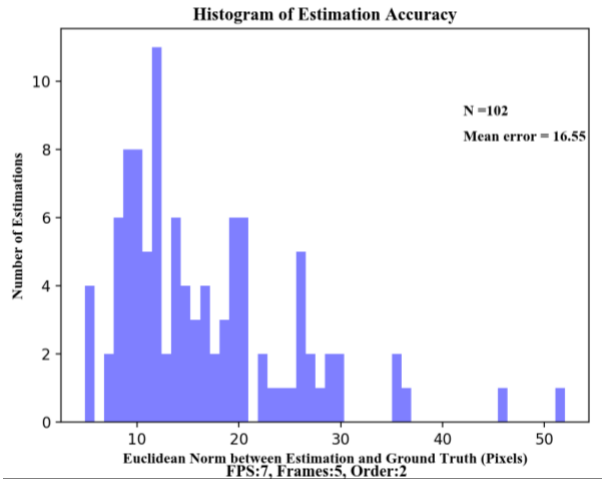

Figure 7: Histogram of error for 7 frames skipped (4.3 FPS), where the mean error is 16.55 and 102 trials were used.

There is a clear tradeoff between the lower FPS and the accuracy of the estimation. However, because OpenPose takes a few seconds to produce results for even a small resolution (used 480p), you may want to trade lower FPS for more concurrent estimation. Concurrent estimation is important in applications such as autonomous driving.

4.2. Effect of Order of the Polynomial

We varied the order of the polynomial that fits the keypoints, which we extrapolate to find the coordinates of that corresponding keypoint in the next pose. To observe the effect of the order we set all other parameters to a fixed value as follows:

FS=3; Number of Frames considered=5;
dx=simple difference

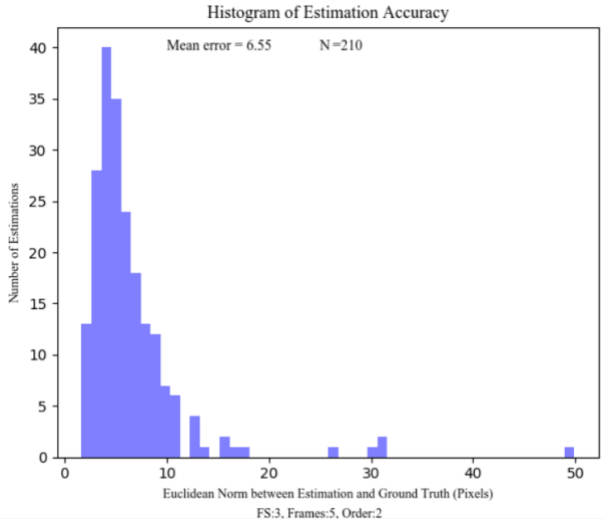

Figure 8: Histogram of error for order 2 polynomial, where the mean error is 6.55 and 210 trials were used.

In figure 7 we see that by fitting a second order polynomial to the feedback pose key points we observe that the error ranges from 0 to about 50 with the mean error being 6.55.

Figure 8 shows us that, for fitting a polynomial with order 4, range of the error is expanded to over 350 and the mean error increases to almost three times the mean error for order 2.

Mean Error Variation for order ranging from 1 to 5 is shown in figure 9. With FS, number of feedback frames and method for calculating dx kept constant the mean error is directly proportional to the order of the polynomial i.e. error increases as order of fitting polynomial is increased.

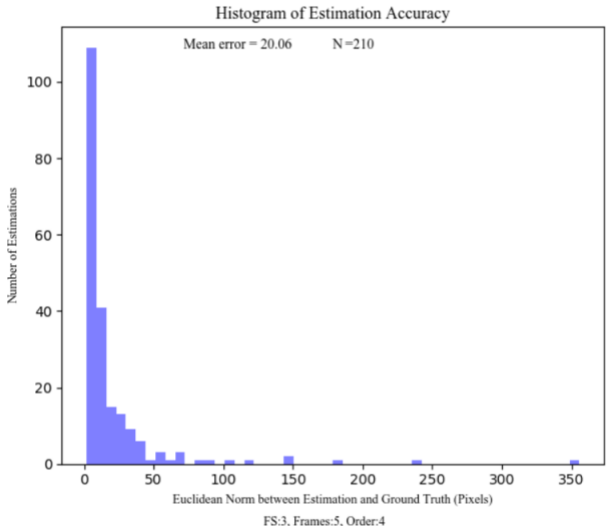

Figure 9: Histogram of error for order 4 polynomial, where the mean error is 20.66 and 210 trials were used.

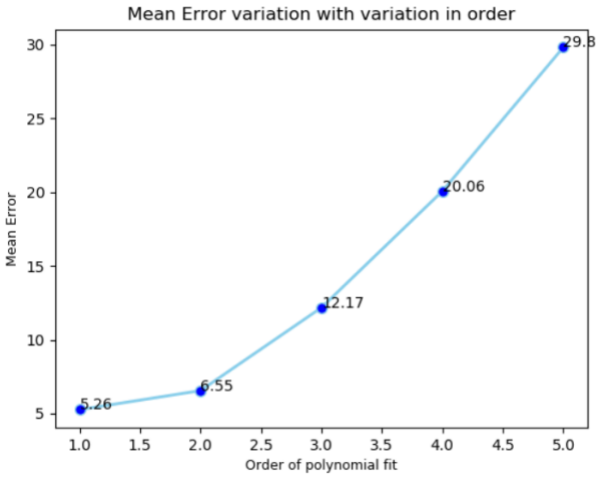

Figure 10: Scatter plot of mean error variation with respect to order of polynomial fit.

4.3. Effect of curve fitting dx

To observe and plot the effect of method used for estimating the x coordinate of the next pose, we tried one more approach for estimation. To be clear, we are changing the dx in the formula:

$$x_{next} = x_{current} + dx,$$
$$dx = x_{current} - x_{prev}$$

The first attempt is calculated as shown in the above formula. The next attempt was to use np.polyfit() to get a more realistic one that attempts to take acceleration into account by looking at the second derivate of x displacement.

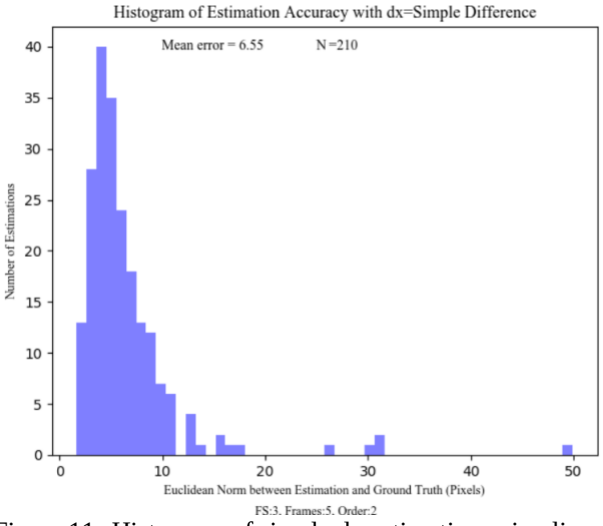

Figure 11**:** Histogram of simple dx estimation using linear fitting.

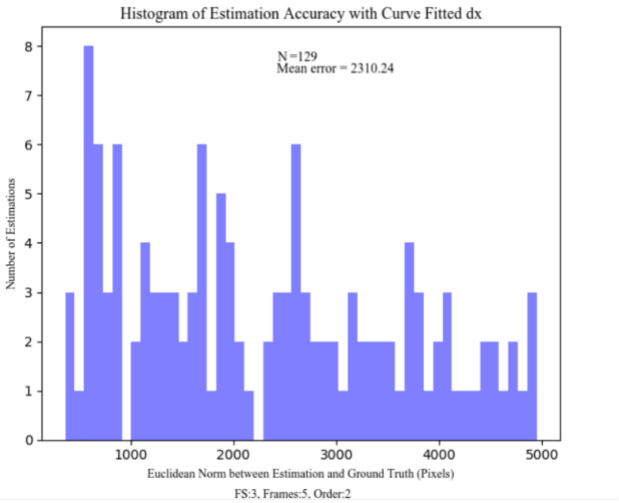

Figure 12: Histogram of complex dx estimation using polynomial fitting dx.

Comparing the results in figure 10 and figure 11, curve fitting x coordinates clearly increases the error exponentially. The error ranges 5000 and we found that only 27 points have error below 1000. Estimating dx using simple difference gives better results.

4.4. Effect of variation in number of feedback frames

In this case the number of frames considered to estimate the next pose is varied to observe its effect on the accuracy. All other factors are kept constant as:

FS=3; Order of Polynomial=2;
dx=simple difference

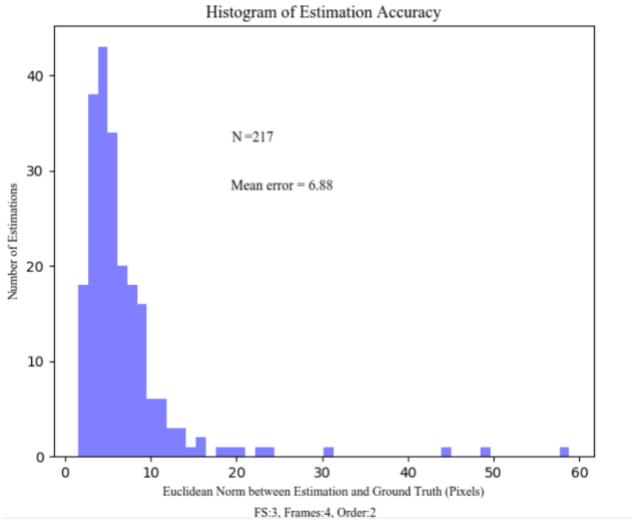

Figure 13: Histogram of error with 4 feedback frames.

Figure 13 shows that when 4 feedback frames are considered the error ranges from 0 to 60 with mean value 6.88.

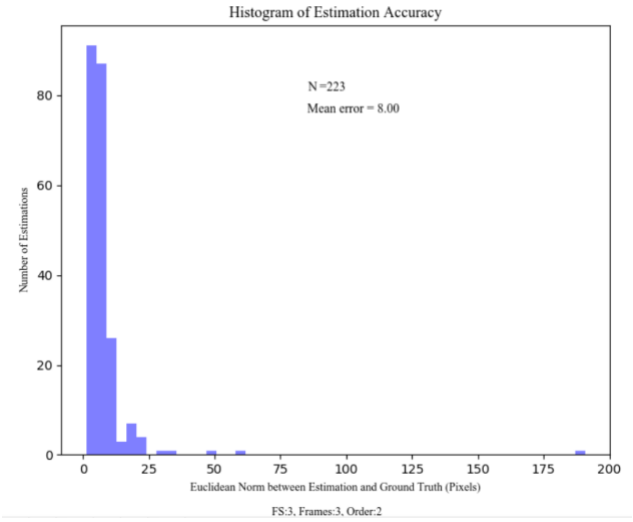

Figure 14: Histogram of error with 3 feedback frames.

When the number of previous frames considered to estimate the next pose decreases to just 3 frames the mean error increases to 8 and the range of error values is extended to 200. This is shown in figure 14.

Figure 15 shows the effect of number of feedback frames on estimation of next pose. As we can intuitively predict, the error minimizes as we take more previous frames into consideration.

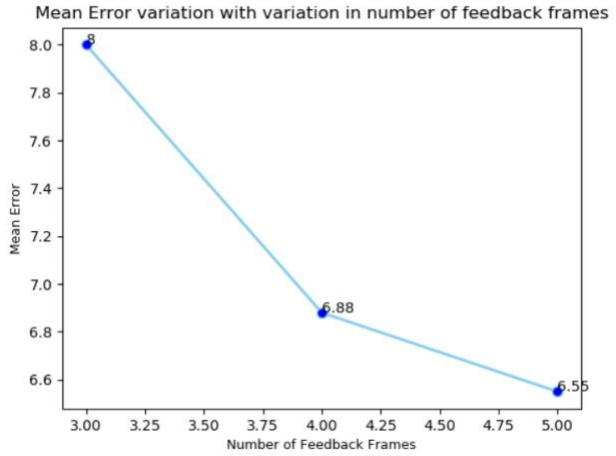

Figure 15: Mean error with respect to number of feedback frames used.

5. Conclusion

5.1. Limitations of Polynomial Estimation

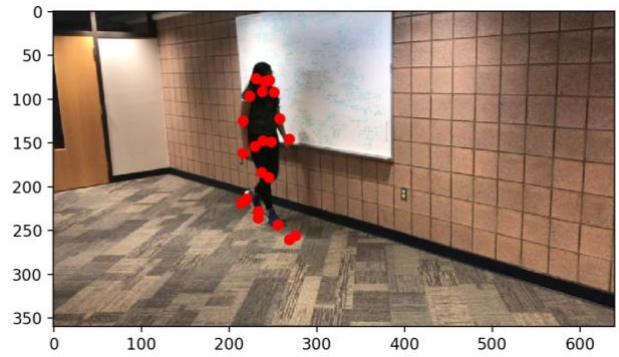

Figure 16: Estimation on a diagonal walking path. Error from ground truth is 7.807.

The diagonal path still gives a good estimation. This is exciting. However, you can see that the feat key points are very far off for the front foot. This is clearly an error with the polynomial estimation, and signals that a more complex model might be better suited for a non-parallel path to the camera plane. The model used for estimation is clearly a simplified one, and has its failure points as shown in Figure 16.

Sometimes, the estimation gives on point that is very far off, like in the following figure.

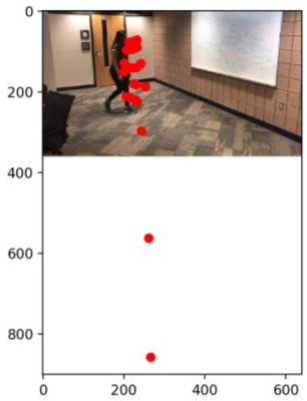

Figure 17: Estimation is off the image space, giving a very large error of 167.88.

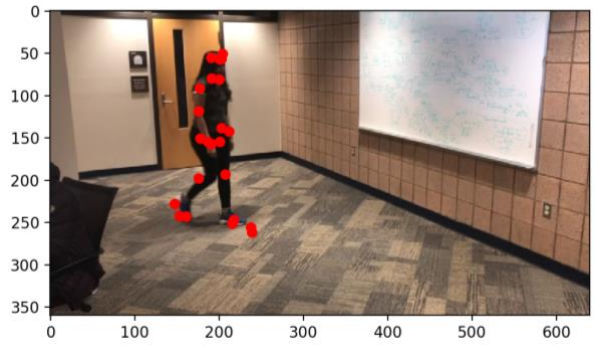

Figure 18: Good estimation on a non-parallel path.

The non-parallel paths still can come up with a very good estimation as shown above.

5.2. Further Research

Though our analysis of polynomial estimation proved to work well for movement analysis of pose estimation, we could likely get better results by treating this problem as a machine learning problem. There is a lot of paired data, which makes it uniquely suited for such a problem.

The only part to solve to make this a machine learning problem would be to determine what other features should be included besides the actual points. Should velocity, acceleration, or other aspects of the poses be in the feature vector? Should we continue to treat each point separately or group points, like grouping leg key points?

5.3. Final Conclusion

Our analysis on the frames per second, number of feedback frames, order of polynomial, and method to estimate dx suggests that 10 FPS, order 1 polynomial fitting, simple dx fitting, and 5 feedback frames proved to be the best combination for a continue pose estimation. The most surprising factor is the order 1 polynomial, because the first intuition would be that complexity would increase the estimation. This means that some sort of curve overfitting might occur when you use a higher order. The FPS factor is the least surprising. The tradeoff between the ability to estimate better and the computational power needed should be determined on a per usage case. While attempting to increase the accuracy based on a complex dx estimation, we unexpectedly got worse results.