Machina Labs Writeup

John Jack Messerly

January 2022

TL;DR

1. I used pandas to synchronize timestamps, and to interpolate them with nearest neighbors. Logs were discretized to 0.1 ms intervals. Entries that did not have a timestamp match within 5ms were removed. This was necessary to compare the coordinates point-by-point. Note: I drop the Z dimension for this report.
2. I found an affine transformation between the tracker coordinates and path coordinates using least squares, and validated it by checking a consensus between hundreds of random subsets of the data.

$$T = \begin{bmatrix} 0.22 & 3.58 & 602.97 \\ -1.20 & -1.57 & 592.64 \\ 0 & 0 & 1 \end{bmatrix}$$

3. I used linear algebra to decompose the affine transformation into a translation of $\begin{bmatrix} 602.97 \\ 592.64 \end{bmatrix}$, rotation angle 1.75 radians (clockwise), a reflection about the y axis, x-shear angle of -0.67 (counter-clockwise), and scaling factors of $\begin{bmatrix} -1.22 \\ 3.25 \end{bmatrix}$. I interpret each of these distortions individually.
4. I made some visualizations of the robot arms velocity as it travels, but saw that the arm accelerates in a spiral pattern by design (speeds up and slows down over curves), so couldn't get many insights.
5. Included a small section at the end for running the code

Synchronizing the Logs

You need to compare the logs point-by-point in order to do any data analysis. To do this, it's required to line up the timestamps at even intervals so that you compare points when you are supposed to. Since the timestamps are not necessarily synchronous, you need to interpolate at a given interval dt with an algorithm like so:

$T \leftarrow t_f - t_i$;
$dt \leftarrow T/steps$;
$t \leftarrow ti$;
$merged \leftarrow \{\}$;
while $t \leq t_f$ **do**
  $data_1 \leftarrow NearestNeighbor(log_1, t)$;
  $data_2 \leftarrow NearestNeighbor(log_2, t)$;
  $merged \leftarrow merged \cup (data_1, data_2, t)$;
  $t \leftarrow t + dt$;
end

Timestamps are set to fixed intervals, and data is interpolated from both logs with nearest neighbors. Pandas basically has a one-liner that does this, so I used that. If Pandas can't find a reasonable nearest neighbor (within 5ms), it sets that row to NaN, but I basically found this didn't happen a lot (just at the beginning, when one was active and another lagged). I believe it discretized the logs into 10 ms steps over the two minutes of data gathering.

Estimating / Interpreting the Affine Transformation

When you plot the data raw, you just kind of notice there's an obvious homography between the tracker points and the planned path.

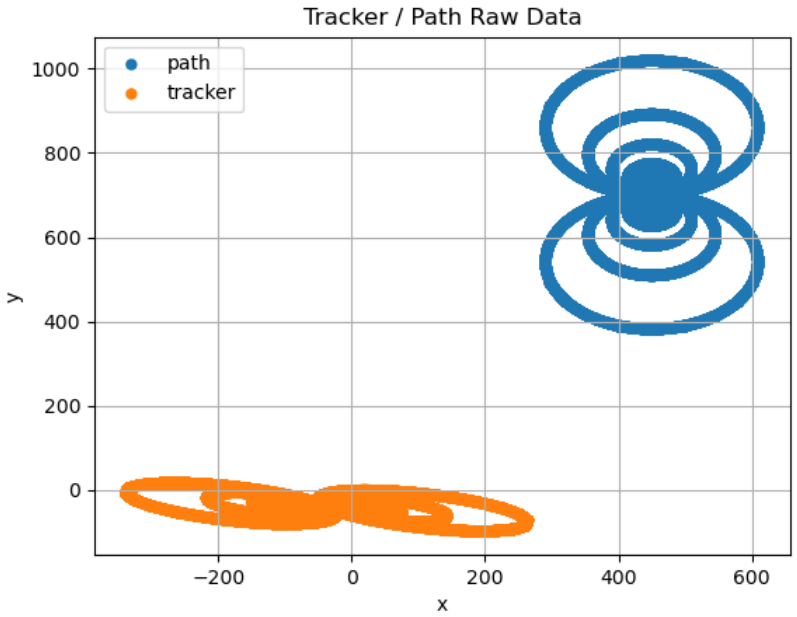


Figure 1: Obvious Homography

If we're lucky, we can find an affine transformation T between the points, such that $TA = B$, where A is our tracker coordinates and B is our path coordinates, and T is a $3x3$ matrix in homogeneous coordinates. From such a T, we can determine the rotation, translation, shear and scaling between the coordinates.

Linear regression is the go-to for these registration problems (we want to fit a linear model, that is hopefully affine, because fitting something complex between the coordinates would be useless to us for analysis). SKlearn makes this very simple. Using linear_model, we can find this matrix

$$T = \begin{bmatrix} 0.22 & 3.58 & 602.97 \\ -1.20 & -1.57 & 592.64 \\ 0 & 0 & 1 \end{bmatrix}$$

Of course, we want someway to validate our regression. I'll go over what I did for that at the end. But first, I want to talk about how to interpret this matrix

Decomposing / Interpreting the Coordinate Transform

All 2D affine transforms represented by homogenous matrices

$$\begin{bmatrix} a & b & c \\ e & f & g \\ 0 & 0 & 1 \end{bmatrix}$$

can be decomposed into their **rotation**, **translation**, **shear**, and **scaling**. These distortions can be interpetted separately to figure out what is wrong with the Kuka arm.

Translation

This is easy: the right-hand column is our coordinate transform. This is 602.97 in the x direction and 592.64 in the y direction. This implies the robot is just placed in the outside world at the wrong coordinate, or something similar.

Rotation, Scaling, Shear

Rotation requires some math to dig out of the matrix T. A well-known property of rotations is that they only change the direction of vectors they are applied to, but not the magnitude. Thus, all rotation matrices are unitary: they have a magnitude of I. Understanding this, we can first factor T into a unitary matrix Q and an upper triangle matrix R using QR decomposition. Taking out the translation and just looking at the other distortions, we have

$$T = \begin{bmatrix} 0.22 & 3.58 & 0 \\ -1.20 & -1.57 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.18 & 0.98 & 0 \\ 0.98 & 0.18 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1.22 & -2.18 & 0 \\ 0 & 3.25 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can match up our Q matrix $\begin{bmatrix} -0.18 & 0.98 & 0 \\ 0.98 & 0.18 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ with the rotation matrix everyone knows $\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$, but, the negatives on the diagonal are switched. This just means there is not only a rotation, but a reflection, which we can include using the y-reflection $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Fortunately, the inverse of the reflection is itself, so we can write

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -0.18 & 0.98 & 0 \\ -0.98 & -0.18 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1.22 & -2.18 & 0 \\ 0 & 3.25 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The first matrix is a reflection, the second is a clockwise rotation of 1.7 radians, but what is the last one, the R? We can decompose that matrix into two of the form

$$\begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where s is the x-shear angle, and a, b are scaling factors. Using matrix arithmetic, this comes out to:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -0.18 & 0.98 & 0 \\ -0.98 & -0.18 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -0.67 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1.22 & 0 & 0 \\ 0 & 3.25 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Visualizing The Distortions

It's fun to visualize what each of these transformations does to the trajectory.

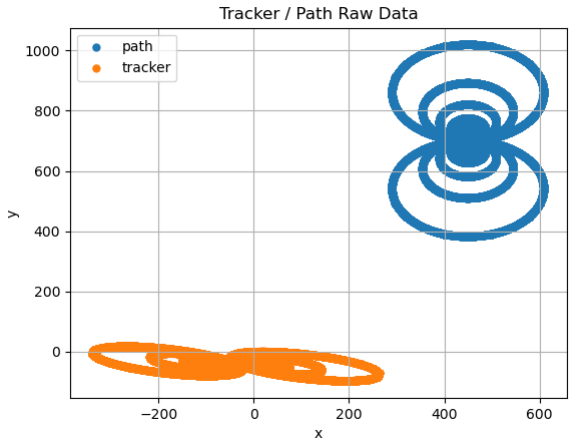
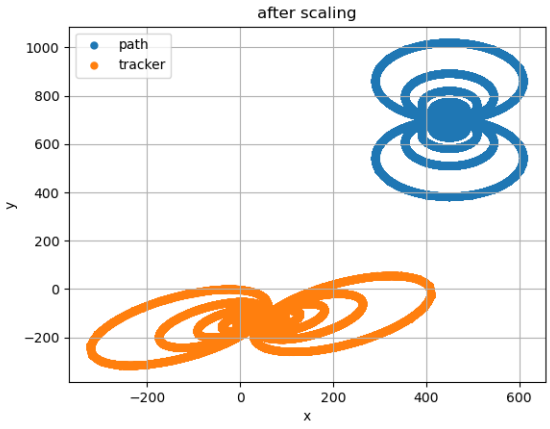
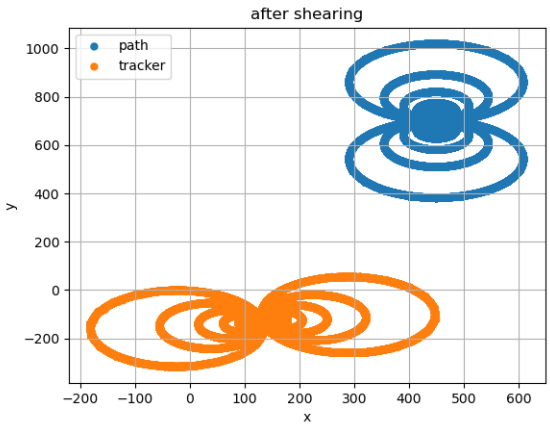
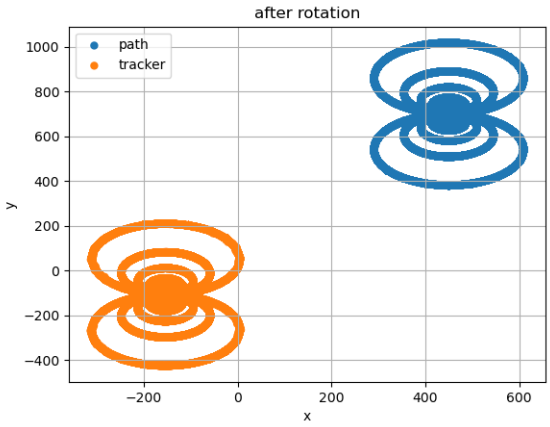
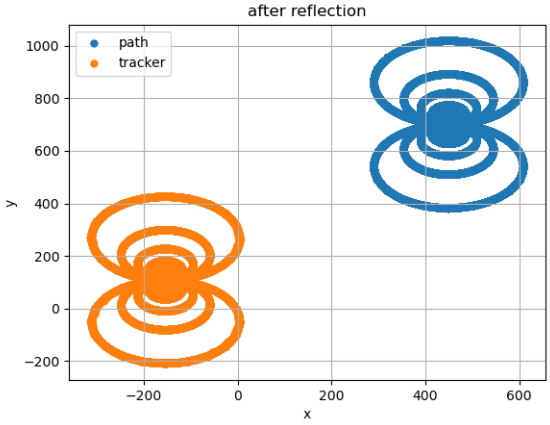


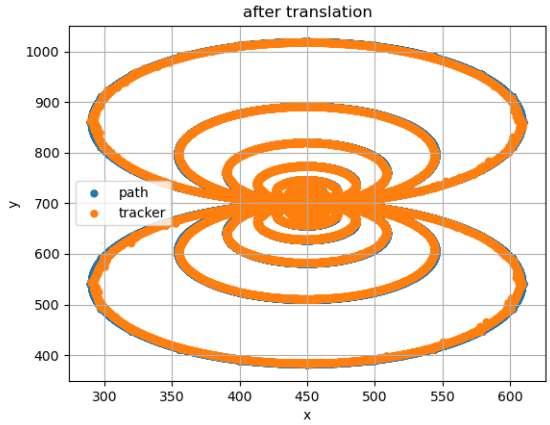
Figure 2: original data



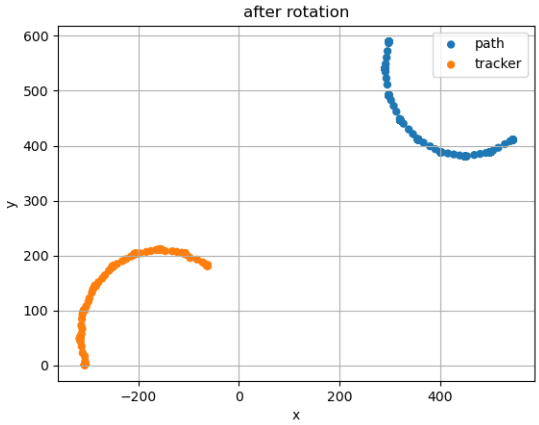


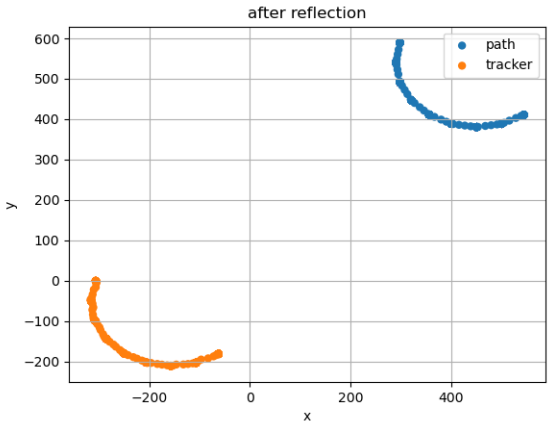

after rotation
path
tracker
1000
800
600
400
200
0
−200
−400
y
−200
0
200
400
600
x


after reflection
path
tracker
1000
800
600
400
200
0
−200
y
−200
0
200
400
600
x


after translation
path
tracker
1000
900
800
700
600
500
400
y
300
350
400
450
500
550
600
x

(The picture after reflection doesn't look that interesting, because the thing we are reflecting is obviously symmetrical. Here's an "after rotation" and "after reflection" of a sub-segment of the trajectory to show the impact of the reflection)

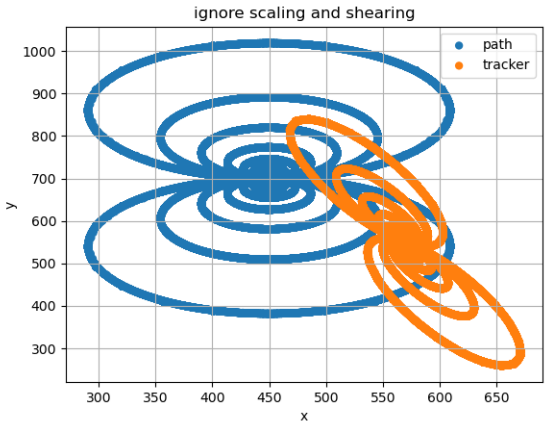




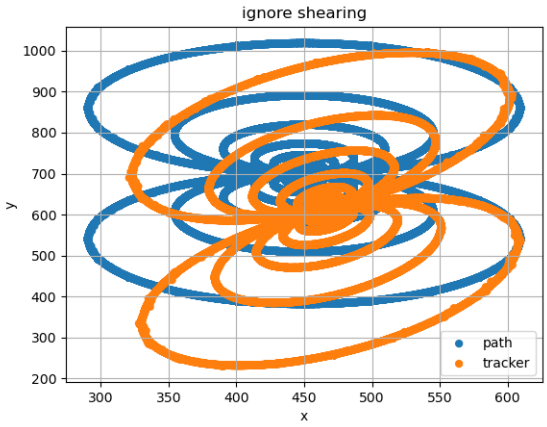
Interpreting the Distortions

I think the rotation, translation and reflection is simply the robot being placed in the wrong location, oriented wrong or there just not being a proper coordinate conversion in the control node. The more difficult to interpret parameters are the shear and scaling.

Here's what happens if we ignore both the shear and scaling in the coordinate transformation:



Even with the rotation and translation (the coordinate change) taken care of, it still looks terrible. Correcting the scaling distortion (but not the shear distortion), and we get this:



Here's my guess about what's wrong:

1. The scaling could be a unit conversion. It just expands the circles of the track. The robot arm is assuming units that the units of motion are smaller than they actually are.

2. The shearing is really hard to interpret. It represents multiple rotations and an additional scaling that can't be decomposed into the rotation and translation we already have. It could be a single joint that isn't calibrated in a big chain of joints. It could be just a weird error in the controller itself.

It's in general hard to separate the effects of shearing and scaling, so this analysis is very incomplete.

My big takeaway here: The error isn't simply a body coordinate frame problem, it's something more nontrivial than that (given by the scale and the skew). You can also notice that from just looking at the original data and seeing it is a non-rigid transformation.

The biggest insight I probably found here was noticing that there's a reflection, which is invariant to fixing any other distortion.

Validating The Fit

I did a few hundred experiments recalculating the affine matrix T on different random subsets of the data. I took batch sizes equal to one tenth of the total number of samples, and did 500 experiments.

Here are the statistics I ended up finding:

Distortion	Mean	Variance
translation x	602.97	0.003824
translation y	592.64	0.007674
rotation (rad)	1.75	0
scale x	-1.22	0
scale y	3.25	0.000001
shear x	-0.67	0
r^2	0.985	0

The fit is so consistent I'm confident it's correct.

Velocity Analysis

Velocity analysis is really hard when the data is noisy. I noticed right away, trying to plot the velocity and positions together would be a challenge. In order to compute the velocities, I took advantage of pandas again, and resampled the logs at every $0.2, 0.5, 1$ seconds respectively using the resample feature and the given timestamps, and then used finite difference approximations on the positions to get the speed.

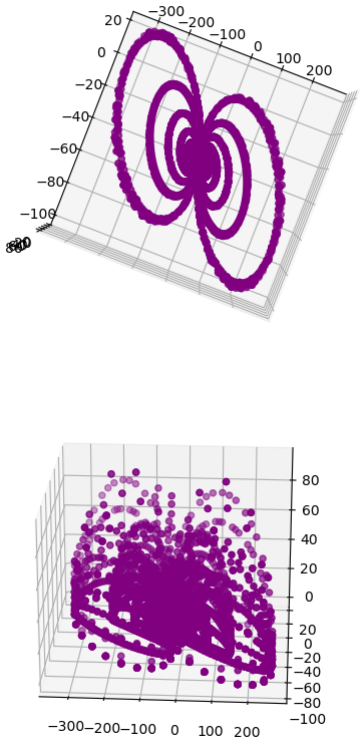


Figure 3: z-axis is speed

From the 3D plot its clear that the finite difference velocity reading is all over the place. However, instead of trying to clean up the data with low-pass filtering etc., I compared it to the path planner on smaller segments:

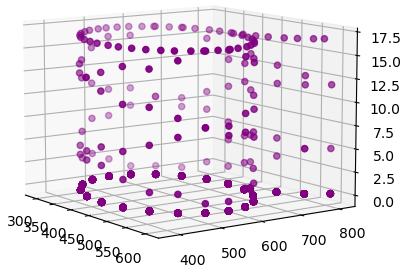


Figure 4: planner accelerates / slows down by design

And saw that the planner itself is intended to speed up and slow down in a double helix pattern. So, the velocity is all over the place by design, and I couldn't really get any more insights from it.

Note: in the code, I'm lazy with dividing out the exact time past between finite difference (the pandas time resampler takes in a datetime string, and I didn't write a parser to turn it into a float for arithmetic). So the speed is off by a constant scale factor, but I think it's fine.

Running the Code

1. Install requirements in requirements.txt with pip (matplotlib, numpy, pandas, scikit_learn).
2. Run the main file to generate insights (will find and validate the affine transformations, plot the original trajectories, and the spiral of speeds from the planner path)
3. All functions are implemented in utils.py. Reliance on sklearn and pandas meant no class design or even much looping