Mathematical Description and Code Implementation

In this file you will find the mathematical calculations behind making the projection onto the wall, as well as my implementation of this into the code.

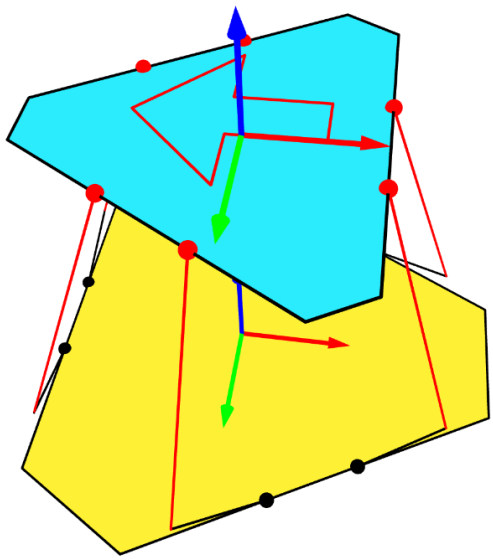
Inverse Kinematics

The mathematical calculations utilized in this project to achieve the inverse kinematics are extensively detailed in Robert Eisele's paper. It is highly recommended to review this resource before proceeding. Inverse kinematics is used to determine the necessary servo angles to achieve a desired platform **position** and **orientation** by solving the equations that describe the platform's geometry.

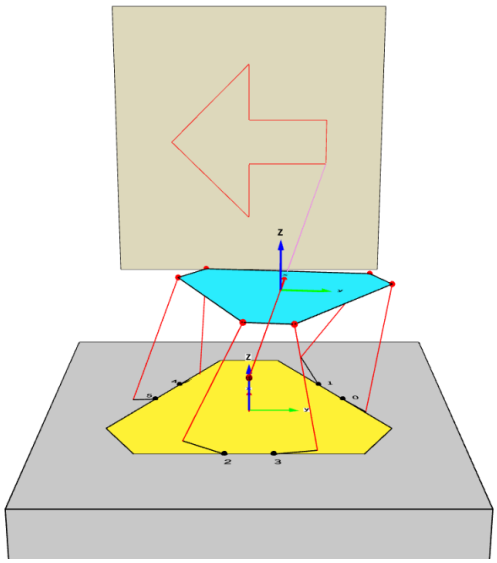
The Problem

The challenge is to determine how the platform must rotate and translate to project an SVG image onto the wall.

Initially, the SVG plotter configuration was as follows:



The desired outcome is:



Given the desired projection size, wall distance, and rotation axis offset, the task is to determine the platform's movements to achieve this projection.

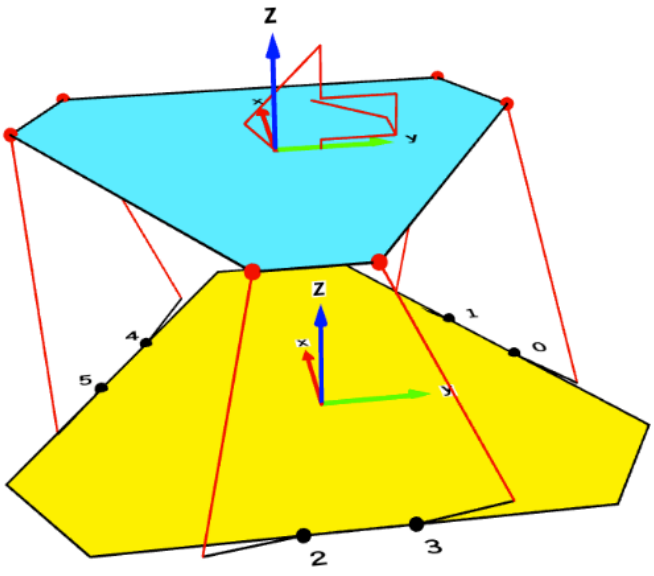
The Solution

Transpose to Vertical Plane, Translation Only

The first step was to make the SVG plot on the vertical plane without any projection, simply by changing the axes. Attaching a laser to the platform would allow it to draw the SVG onto the wall. However, the drawing size would be limited to the platform's translation range.

By inspecting the code and SVG parsing functions, I found that swapping some values (changing "x" to "z", "y" to "x", and "z" to "y") and adding a negative sign to a specific equation transformed everything to the vertical plane.

After adjusting the camera position and orientation, the result was:

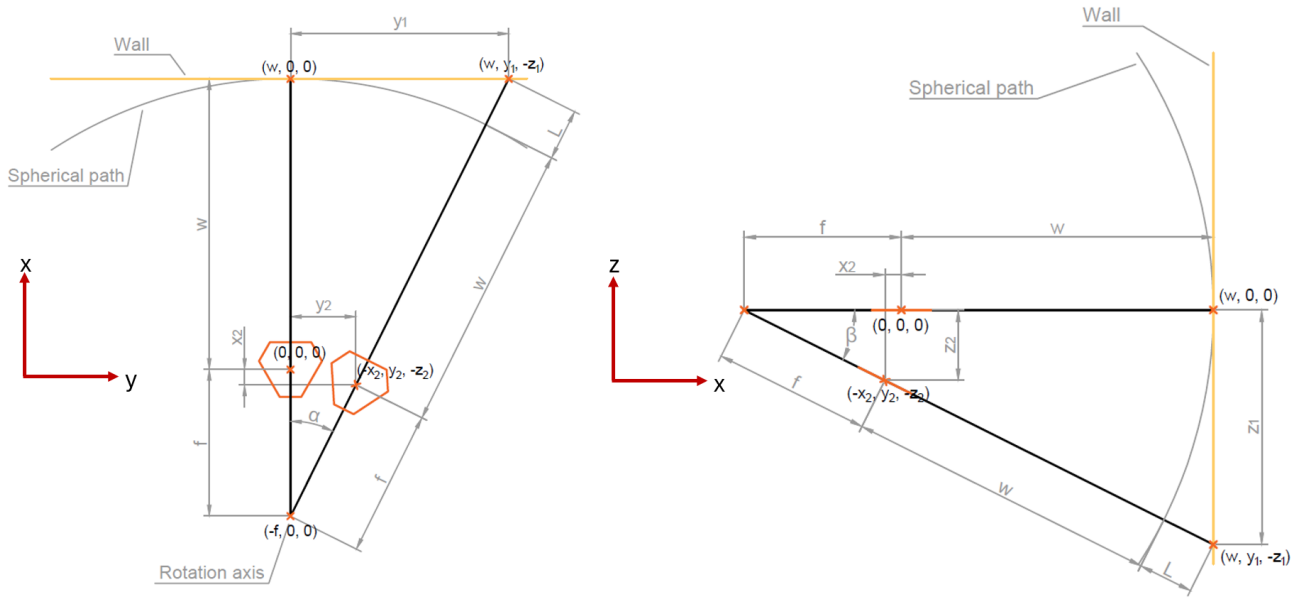


It's important to note that the SVG drawing is now perpendicular to the "x" axis.

Calculate the Projection

With the position for the laser to draw the SVG shape established, the next step was to transform this position into the platform's new translation and rotation values.

Using the platform's coordinate system, we can visualize the problem in both the y-x and x-z planes. The following diagram shows the platform in its zero position, the platform in its "desired position", its rotation axes and the wall.



Given:

- y_1: Distance of desired y projection.
- z_1: Distance of desired z projection.
- w: Distance to wall from coordinate origin.
- f: Distance of the rotation axis offset.

Find:

- $α$: Angle to rotate around z axis.
- $β$: Angle to rotate around y axis.
- x_2: Distance of desired x translation caused by rotations $α$ and $β$.
- y_2: Distance of desired y translation.
- z_2: Distance of desired z translation.
- L: Laser length extension.

The angles α and β can be determined using the trigonometric function $\tan \theta = \frac{\text{Opposite Side}}{\text{Adjacent Side}}$.

$\alpha = \arcsin\left(\frac{y_1}{f + w}\right)$, $\beta = \arcsin\left(\frac{-z_1}{f + w}\right)$

With these angles, the values of x_2, x_2 and x_2 can be found using the trigonometric functions $\cos \theta = \frac{\text{Adjacent Side}}{\text{Hypothenuse}}$ and $\sin \theta = \frac{\text{Opposite Side}}{\text{Hypothenuse}}$.

$x_2 = f - f \cdot \cos(\alpha) \cdot \cos(\beta)$
$y_2 = f \cdot \sin(\alpha)$
$z_2 = f \cdot \sin(\beta)$

The point for the platform to translate to will be $(-x_2,y_2,-z_2)$

Finally, the laser length extension L is calculated using:

$(L + f + w) \cdot \cos(\alpha) \cdot \cos(\beta) = f + w$

Solving for L:

$L = \frac{f + w}{\cos(\alpha) \cdot \cos(\beta)} - f - w$

Code implementation

With the values of each variable known, the final step is to implement this in our code. The pre-calculated y_1 and z_1 values (obtained from the `Animation.SVG` and `parseSVGPath` functions) are converted into the new translation and orientation values for the platform.

The original `Animation.Interpolate` function, which interpolates through the points of the path to create a smooth transition between vertices and returns the normalized object to run the animation, was modified. The changes include:

- Adding the `calculateMovements` function to implement the previously discussed equations and return the corresponding position and orientation values for the platform.
- Implementing orientation interpolation, as the SVG plotter initially only used translation movements.
- Adding a new `path` function to calculate the drawing path positions of the animation, both in real-time and for the end result.

From there, the functionality of the code remains largely unchanged (with minor tweaks for additional features), where an orientation and translation are set, and the platform is updated with the corresponding values.