We have a chessboard with 25 inner points (i.e. the board is really 6×6). We have left and right cameras of the same chessboard *and* the exact pixel coordinates of the corners due to OpenCV. Our next task is to figure out a mapping from the left camera to the right camera, assuming *a fixed height*. (We *already* have robot-to-camera and camera-to-robot matrices which Sanjay generated, but this is instead mapping from left-camera to right-camera points and vice versa.) To check that our solution is working, we should take the left camera's image of the chessboard, apply the mapping, and see that it looks almost exactly like the right camera's image.

Our data from the chessboard looks like:

$$P = \begin{bmatrix} p_1^{(1)} & p_2^{(1)} \\ p_1^{(2)} & p_2^{(2)} \\ \vdots & \vdots \\ p_1^{(25)} & p_2^{(25)} \end{bmatrix} \quad \text{and} \quad Q = \begin{bmatrix} q_1^{(1)} & q_2^{(1)} \\ q_1^{(2)} & q_2^{(2)} \\ \vdots & \vdots \\ q_1^{(25)} & q_2^{(25)} \end{bmatrix} \tag{1}$$

where P and Q are the chessboard coordinates for the left and right cameras, respectively, on the same image/board. We're using 2-D here since we assume the height is fixed and known.

Our optimization problem is to find some 2×2 matrix A which cam map from left points to right points. We can use the Frobenius norm optimization:

$$\underset{A \in \mathbb{R}^{2\times 2}}{\text{minimize}} \quad \|PA - Q\|_F^2.$$

which reduces to an L_2 norm optimization if we split A and Q into columns. Let $a^{(1)}$ and $a^{(2)}$ be the first and second columns of A, and do a similar thing for $q^{(1)}$ and $q^{(2)}$. We get:

$$\underset{A \in \mathbb{R}^{2\times 2}}{\text{minimize}} \quad \|Pa^{(1)} - q^{(1)}\|_2^2 + \|Pa^{(2)} - q^{(2)}\|_2^2.$$

Here, $a^{(1)}$ and $a^{(2)}$ appear in independent terms of the objective, so we can minimize the two terms in the sum independent of each other. The two terms are differentiable, real-valued functions so their solution can be computed by taking gradients with respect to (for the first one, $a^{(1)}$). We combine our optimizers together to form A^*, our optimizing matrix. In math:

$$\nabla_{a^{(1)}} (Pa^{(1)} - q^{(1)})^T (Pa^{(1)} - q^{(1)}) = 0 \quad \Rightarrow \quad (a^{(1)})^* = \underbrace{(P^T P)^{-1}}_{2\times 2} \underbrace{P^T q^{(1)}}_{2\times 25}. \tag{2}$$

Now we can get a mapping from the entire left image to the entire right image. Each point in the left image can be expressed as a 1×2 matrix, so we would left-multiply that with A^* to get the corresponding right point. Do this for all the stacked points on the left image (so the left image is an $N \times 2$ matrix for some large N)[1] and we get the desired mapping, again, *for a fixed height*.

I assume we don't need to add regularization to the objective, since the data would not be

[1] For the images we have, they're 1920×1080 so maybe $N = 1920 \cdot 1080$?

noisy.

Also, since the matrix A will almost certainly be invertible, I assume we can use the inverse to map in the reverse direction as needed.