Solution 1

(a) Assuming each of the K correspondences are picked without replacement:

$$P(\text{1st inlier}) = \frac{N-J}{N}$$

$$P(K \text{ inliers}) = \frac{N-J}{N}\frac{N-J-1}{N-1}...\frac{N-J-K+1}{N-K+1} \tag{1}$$

$$= \frac{(N-J)!(N-K)!}{(N-J-K)!N!} \tag{2}$$

(b) Say we want to sample S times such that there is probability P that at least one trial had a set of K inliers (no outliers). The probability that a trial fails (has >1 outlier) is shown below.

$$P(\text{trial fails}) = 1 - P(K \text{ inlier}) = 1 - \frac{(N-J)!(N-K)!}{(N-J-K)!N!}$$

For S trials to fail we have:

$$P(\text{all S trials fail}) = 1 - P = (1 - (\frac{(N-J)!(N-K)!}{(N-J-K)!N!})^S$$

Solving for S we get:

$$S = \frac{\log(1-P)}{\log(1 - (\frac{(N-J)!(N-K)!}{(N-J-K)!N!})}$$

Thus the minimum number of trials to guarantee at least one trial with no outliers is $\frac{\log(1-P)}{\log(1-\frac{(N-J)!(N-K)!}{(N-J-K)!N!}}$

(c)

$$P(\text{all K samples are inliers and are from } I_1) = (\frac{I_1}{N})^K$$

$$P(\text{all K samples are inliers and are from } I_2) = (\frac{I_2}{N})^K$$

$$P(\text{all K samples inliers and same plane}) = (\frac{I_1}{N})^K + (\frac{I_2}{N})^K$$

Solution 2

(a) For reference, solving the least squares minimization problem for linear regressions yields the following two equations for the slope and intercept.

$$m = \frac{\sum x_i(y_i - \bar{Y})}{\sum x_i(x_i - \bar{X})}$$

where $\bar{X}$ and $\bar{Y}$ are the mean of X and mean of Y, respectively.

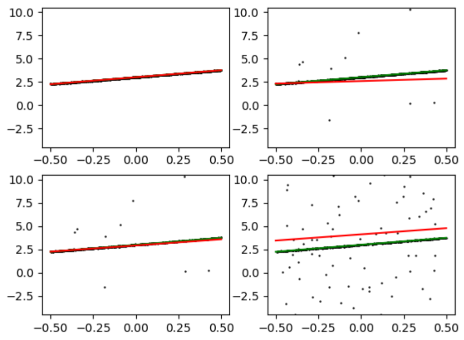


Figure 1: Least squares regression on datasets with varying amounts of outlier noise. All datasets have a small amount of gaussian noise.

This method of robust line fitting begins to fail as more outlier noise is introduced. This can be seen as increasing the outlier noise causes a shift in

(b) The RANSAC algorithm randomly samples K points and then fits a least squares line to it. This is repeated N times and the line with the largest inlier set is chosen.

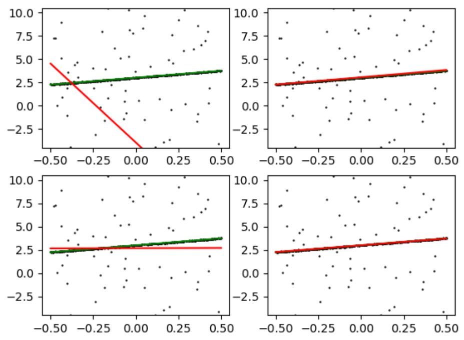


Figure 2: RANSAC algorithm outputs with varying parameters.

The two plots on the right show much better results compared to the two on the left, which show that RANSAC performs best with small K and relatively larger N. After running the script multiple times, these parameters consistently returned the best model.

Solution 3

(a) If both cameras are at the same point, then both have the same $[R|t]$ term in the model. Given $P = [x, y, z]^T$, we have

$$\vec{p_1} = \lambda_1 K_1 * P$$

$$\vec{p_2} = \lambda_2 K_2 * P$$

As such we can simply compute the inverse of one K and plug it into the other equation, as follows:

$$P = \frac{1}{\lambda_1} K_1^{-1} \vec{p_1}$$

$$\vec{p_2} = \frac{\lambda_2}{\lambda_1} K_2 K_1^{-1} \vec{p_1} \tag{3}$$

$$\vec{p_2} \sim K_2 K_1^{-1} \vec{p_1}$$

Equation 3 above is a homography, where $H = K_2 * K_1^{-1}$.

(b) Let $P = [x, y, z]$ be a point in the plane $L^T\vec{X} = 0$, where $\vec{X}$ is a 3D homogeneous vector. This can be rewritten as $L^T X = d$ for a standard cartesian vector. For this derivation, $\vec{P} = [x, y, z, 1]$.
Assume that point P is in camera 1 coordinates. As such, we can express $\vec{p_1}$ as

$$\lambda_1 \vec{p_1} = K_1[I|0]\vec{P} = K_1 P$$

$$P = \lambda_1 K_1^{-1} \vec{p_1}$$

We can solve for λ_1 by using the equation of the plane

$$\lambda_1 L^T P = d$$

$$\lambda_1 L^T K_1^{-1} \vec{p_1} = d$$

$$\lambda_1 = \frac{d}{L^T K_1^{-1} \vec{p_1}}$$

Thus, $P = \frac{d}{L^T K_1^{-1}\vec{p_1}} K_1^{-1}\vec{p_1}$.
Plugging this into the expression of $\vec{p_2}$ we get

$$\begin{aligned}
\lambda_2 \vec{p_2} &= K_2[R_2|t_2]\vec{P} = K_2[R_2 P + t_2] \\
&= K_2[R_2 * \frac{d}{L^T K_1^{-1}\vec{p_1}} K_1^{-1}\vec{p_1} + t] \\
&= K_2[\frac{d}{L^T K_1^{-1}\vec{p_1}}(R K_1^{-1}\vec{p_1} + \frac{1}{d} t L^T K_1^{-1}\vec{p_1})] \\
&= \frac{d}{L^T K_1^{-1}\vec{p_1}} K_2[R + \frac{1}{d} t L^T] K_1^{-1}\vec{p_1}
\end{aligned}$$

This can be rewritten in the following form where $H = K_2[R + \frac{1}{d} t L^T]K_1^{-1}$.

$$\vec{p_2} = \lambda H \vec{p_1}$$

Solution 4

(a) The homography matrix H was calculated using SVD to solve $Ah = 0$, where h is a flattened version of H. Here, the A matrix is expressed as follows:

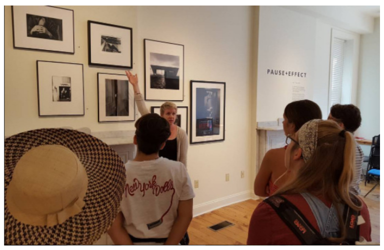
$$A = \begin{bmatrix} 0 & -p'_{iz} & p'_{iy} \\ p'_{iz} & 0 & -p'_{ix} \\ -p'_{iy} & p'_{ix} & 0 \end{bmatrix} * \begin{bmatrix} p_{ix} & p_{iy} & p_{iz} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_{ix} & p_{iy} & p_{iz} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{ix} & p_{iy} & p_{iz} \end{bmatrix} \tag{4}$$

$$= \begin{bmatrix} 0 & 0 & 0 & -p'_{iz}p_{ix} & -p'_{iz}p_{iy} & -p'_{iz}p_{iz} & p'_{iy}p_{ix} & p'_{iy}p_{iy} & p'_{iy}p_{iz} \\ p'_{iz}p_{ix} & p'_{iz}p_{iy} & p'_{iz}p_{iz} & 0 & 0 & 0 & -p'_{ix}p_{ix} & -p'_{ix}p_{iy} & -p'_{ix}p_{iz} \\ -p'_{iy}p_{ix} & -p'_{iy}p_{iy} & -p'_{iy}p_{iz} & p'_{ix}p_{ix} & p'_{ix}p_{iy} & p'_{ix}p_{iz} & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

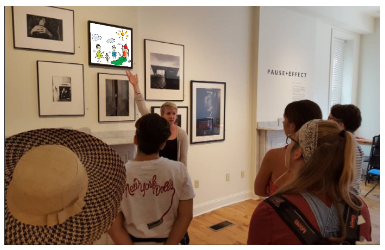
Since `np.linalg.svd` returns all singular values in sorted order, we simply take the last one (which is smallest), which corresponds to the last column of the V.

(b) The source image was spliced into the destination image by first generating a homography between the quadralateral where the source image would be spliced in, and the source image in its entirety. Since this gives us four correspondences, this lets us successfully find H. Once this was found, every pixel inside the quadralateral was converted using the homography to coordinates in the source image. Using bilateral interpolation, an intensity value was found for the given pixel. The results of this are shown below.

(a) Source image



(b) Destination image



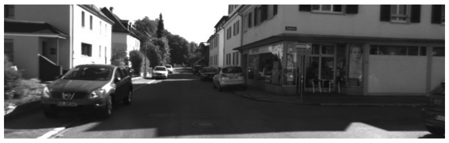
(c) Spliced image

Figure 3: Source, Destination, and spliced image using estimated homography.

Solution 5

(a) The census transform was implemented by comparing the original image to shifted and padded versions of itself. The shifted versions were padded with a value larger than 255, as to always return 0 when compared to an intensity in the original image. This was accomplished by using `np.roll` and changing the appropriate values to the constant.

(b) To generate the cost volume, a `for` loop was used to generate the Hamming distances for each layer, each time with the right census transformed image shifted to the right. Afterwards, `np.argmin` was used to extract the minimum index d along the last axis. This is what constitutes the disparity map, which is colorized and shown below


(a) Left image

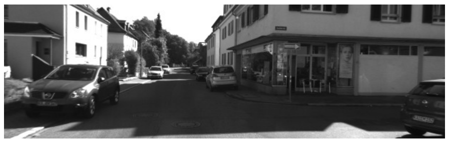

(b) Right image

Figure 4: Left and Right input images.

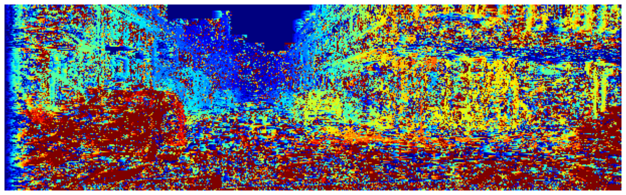


Figure 5: Disparity map output

Information

This problem set took approximately 20 hours of effort.

I discussed this problem set with nobody else

I also got hints from the following sources:

- Wikipedia article on matrix calculus at https://en.wikipedia.org/wiki/Matrix_calculus
- Read numpy tutorial from https://docs.scipy.org/doc/numpy-1.13.0/user/basics.broadcasting.html
- `np.argmin` documentation https://numpy.org/doc/stable/reference/generated/numpy.argmin.html#numpy.argmin
- Wolfram alpha for verifying matrix multiplications https://www.wolframalpha.com/
- For selecting points within a bounding rectangle https://stackoverflow.com/questions/33051244/numpy-filter-points-within-bounding-box