Noah Wilson
MMAE 539 Final

I used an off-the-shelf document camera as a visual sensor to determine camera poses based on perspective distortions of a known object within the camera's field of view. The basis for this pose estimation technique is the fact that relative positions of points of interest in an image depend upon their relative positions in space along with the overall position of the camera, and if the first two things are known, then the third can be determined. In practice, accomplishing this requires some knowledge about the object and the camera.

Of the object, there must be a known correspondence between all points of interest on the object and the points in the image, and the actual geometry of the object – the relative positions of object points in space – must be known as well. It is important to note that it is not necessary to know the absolute position of these points in a global coordinate system, only their positions relative to one another. This is an important distinction, because it can otherwise sound similar to having a map of landmarks in the environment. The object in question could be something like a notable building, whose height and width are known, but does not need to be known where that building is relative to others.

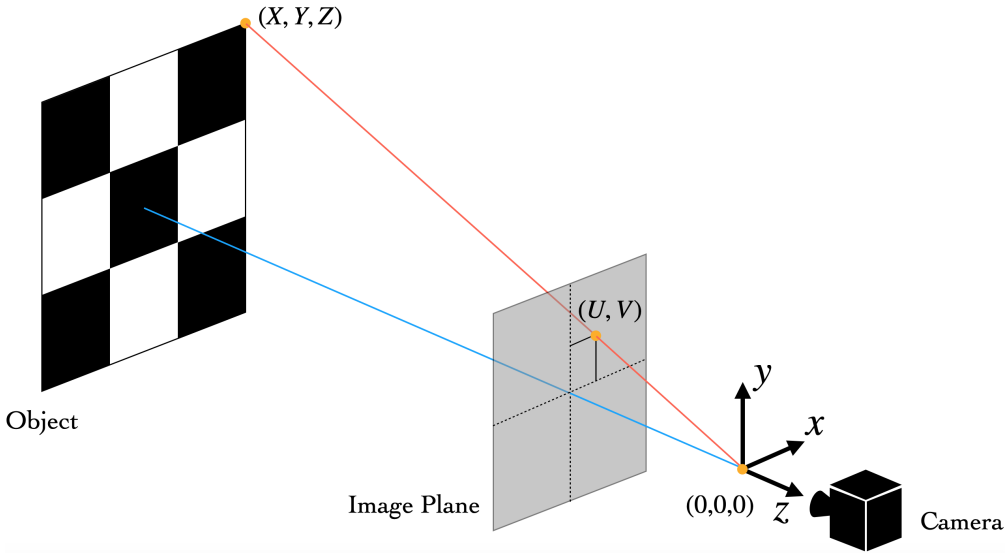

Figure 1: Schematic of the camera pose model. Note the origin of spatial coordinates does not need to be placed at the camera.

Of the camera, it is required to know the intrinsic camera matrix $\mathbf{K}$. This is a matrix that transforms the position of a point in space, $\mathbf{X} = [X, Y, Z]^T$, to the pixel position of the image of that point in the image plane, $\mathbf{P} = [U, V, 1]^T$ as shown in Figure 1. The image plane is 2-D, while object positions are 3-D, so $\mathbf{P}$ is expressed in homogeneous coordinates. As a result, $\mathbf{K}$ does not transform $\mathbf{X}$ into $\mathbf{P}$ but rather $\mathbf{X}$ into $Z\mathbf{P}$ such that

$$Z\mathbf{P} = \mathbf{K}\mathbf{X} \tag{1}$$

So it is not possible to determine Z a priori using Equation 1 alone; it must be known in advance, or found numerically. This is the reason multiple points and object geometries are

required to find the camera pose. Since multiple points are needed and the perspective shifts of those points are used, this is sometimes called the Perspective-n-Point problem.

Equation 2 shows the full form of the intrinsic camera matrix.

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

All of the parameters in the intrinsic camera matrix are, like the name would suggest, intrinsic to the camera. This matrix is invariant with $\mathbf{X}$, and so once found, it does not need to be found again. All of the values have units of pixels: f_x and f_y are the focal lengths of the camera (not the camera's lens), C_x and C_y are the positions of the camera's optical center, and γ is a skew value that helps correct other abnormalities in the image.

Finding $\mathbf{K}$ is a process called camera calibration. To do so with my camera, I took eleven different images of the same checkerboard pattern at varying distances while keeping the X and Y positions constant. A couple of these are shown in Figure 2. The checkerboard

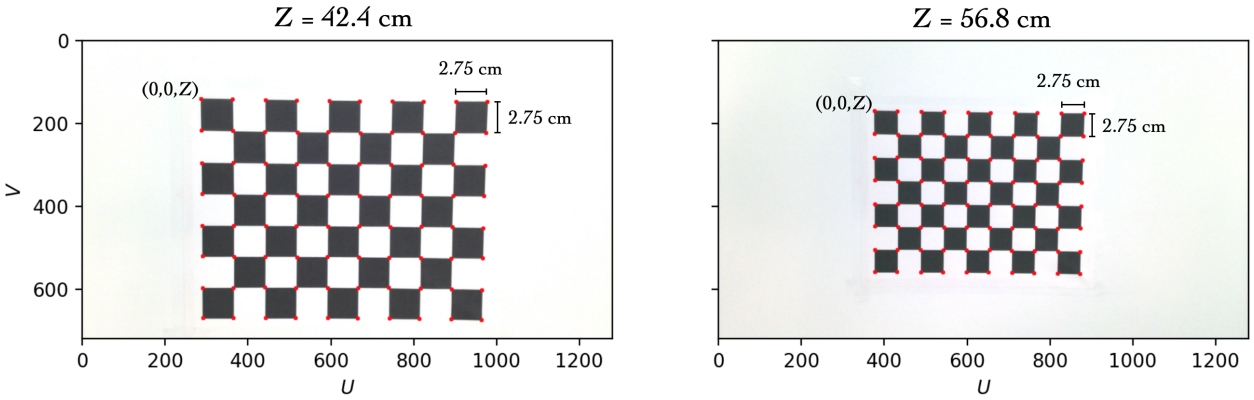

Figure 2: Two sample calibration images used. The red dots mark the corners detected with OpenCV.

has eight rows of ten intersection points, and the pixel positions of these were found in each image using the OpenCV library in Python. The camera's orientation was known – I ensured that the image plane was aligned with and parallel to the checkerboard – so the known correspondence of points was achieved by simply ordering their pixel positions from top-left to bottom-right. The width of each square was 2.75 cm, and since all the points are coplanar, they share the same value of Z, which was also measured for each image. So with $\mathbf{X}$ and $\mathbf{P}$ known, $\mathbf{K}$ can be found by multiplying Eq. 1 by the pseudoinverse of $\mathbf{X}$,

$$\mathbf{K} = Z\mathbf{P}\mathbf{X}^{+} \tag{3}$$

I used eleven images for the calibration process to help smooth out any potential errors in the camera orientations or x and y positions.

Having the camera calibration matrix, I could take images of the same checkerboard pattern from multiple different poses and compute those poses using the following basic process:

1. Calculate the known geometry of the object, $\mathbf{R}$.
2. Form a matrix of pixel positions for all points of interest in the image, $\mathbf{P}$.
3. Calculate $\mathbf{X}$ using Eq. 1 given an initial guess for Z.
4. Compute the object's geometry based on $\mathbf{X}$, $\mathbf{R}'$.
5. Iterate to find Z that minimizes $||\mathbf{R} - \mathbf{R}'||_2$.
6. Calculate $\mathbf{X}$ using value of Z found in step 5.

For the checkerboard pattern, $\mathbf{R}$ is a 79×79 matrix whose elements are the distance between each point and every other point, so

$$R_{ij} = (1 - \delta_{ij})\sqrt{(U_i - U_j)^2 + (V_i - V_j)^2} \tag{4}$$

for $i = 1, 2, ..., 10$ and $j = 1, 2, ..., 8$.

In order to test this procedure I took 154 images of the same checkerboard pattern seen in Fig. 2 all at different positions in X and Z. The orientation of the camera was maintained using a small level. I attempted to keep the Y position constant as well, but in the process of adjusting the orientation as I moved the camera, the Y value slowly drifted downwards. This has no effect on the measurements of X and Z. The results are shown in Figure 4 The

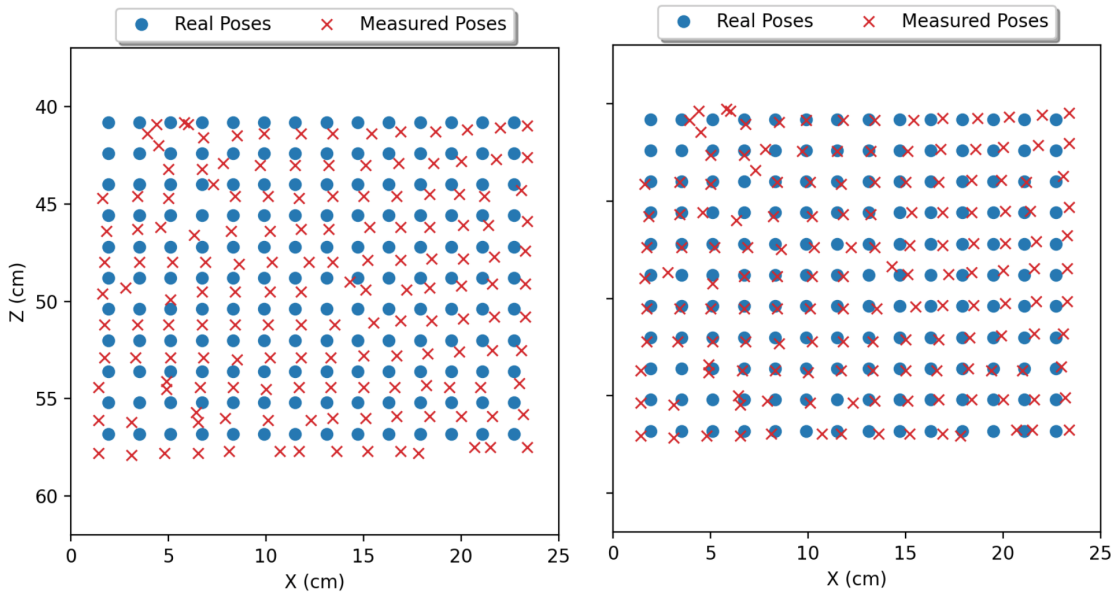

Figure 3: (Right) Real and measured poses. (Left) Real and measured poses when the Z value has been adjusted to correct overshoot.

measured values for Z are almost all overestimates, implying a systematic error in Z rather than a random one. The error in Z averages 1.7%. By multiplying all Z values by 0.987, the measured Z values become much closer overall to the real poses; resulting in an average error of only 0.008%. The error in X appears more random, but is still small, averaging around 3%.

Being reasonably satisfied that the I could accurately measure the camera pose using this method, I wanted to apply it to a localization algorithm, but I did not have any parts to actually control the camera as I might a robot. My solution was to create an incline and

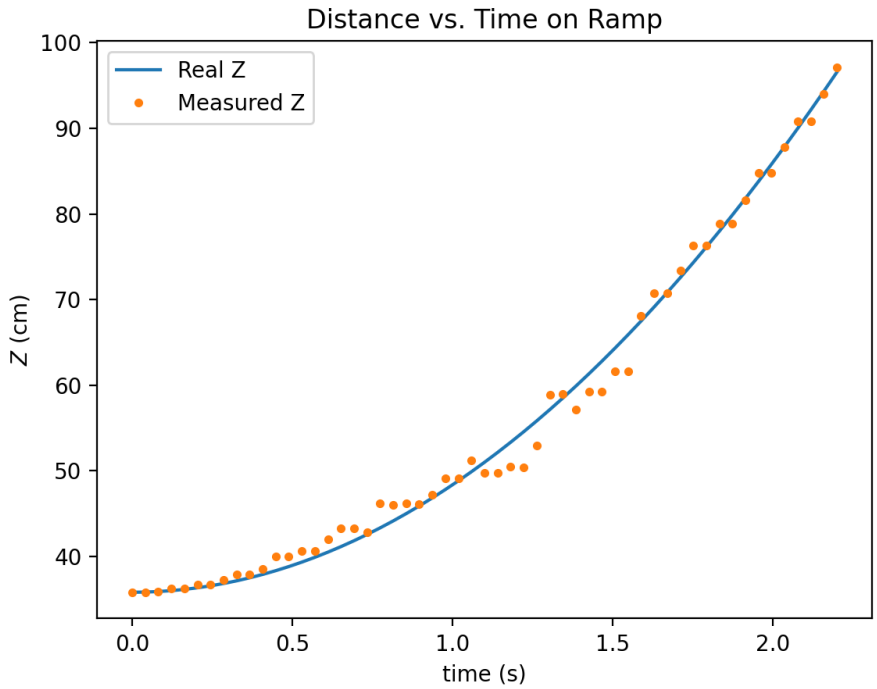

Figure 4: (Right) Real and measured poses. (Left) Real and measured poses when the Z value has been adjusted to correct overshoot.

add simply put the camera on wheels. I ran some additional tests, timing the camera as it rolled down the ramp, to determine the effect of friction along the incline, and then knowing the angle of the incline, I could model the motion using basic kinematics. i then placed the checkerboard pattern on a stand and placed it at one end of the incline and allowed the camera to film the pattern as the camera rolled backwards down the incline. Figure 4 shows the result of one run. Finally, I used this footage, along with the kinematic model of the camera, to implement the sensor into a particle filter. The results can be seen either by running the "main_PF" function in "MAIN.py" or by watching the "Incline_PF.mp4" file included with the submission.