OSU Mechanical Engineering

Smart Products Laboratory

Camera Calibration | Eight

ME Course Number
Spring 2019

Table of Contents

List of Figures

1 Overview

In this lab you will be using a calibration jig to identify the intrinsic and extrinsic parameters of your camera.

2 Background

The information below should provide a basic introduction to the concepts.

2.1 Definitions

The key concepts for this lab are defined and demonstrated below, see figure 1 for depiction of concepts.

- **World coordinate system, $\boldsymbol{P_i^W} = [X_i^W, Y_i^W, Z_i^W, 1]^{\boldsymbol{T}}$**
  - The world coordinate system, denoted by capital X, Y, and Z, is the provides a common reference frame for all objects in space.
  - The location of the i^{th} point of the interest in the global coordinate frame is denoted by $\boldsymbol{P_i^W}$
  - To simplify this further let us define $\boldsymbol{P_i'} = [X_i^W, Y_i^W, Z_i^W]^T$ as a feature, then a point in the would coordinate system would read
    - $\boldsymbol{P_i^W} = \left[\boldsymbol{P_i'^T}, 1\right]^{\boldsymbol{T}}$
- **Camera coordinate system, $\boldsymbol{p_i^C} = \omega\left[x_i^C, y_i^C, 1\right]^T$**
  - The camera coordinate system is a 2D plane whose units are in pixels whose locations are denoted by small x and y.
  - The camera plane also includes a scaling factor, ω.
  - The location of the i^{th} point of interest in the camera plane is denoted by $\boldsymbol{p_i^C}$
  - To simplify this further let us define $\boldsymbol{p_i'} = \left[x_i^C, y_i^C\right]^T$ as a ***pixel***, then a point in the camera coordinate system would read
    - $\boldsymbol{p_i^C} = \omega\left[\boldsymbol{p_i'^T}, 1\right]^T$
- **Intrinsic camera matrix, $\boldsymbol{K}$:**
  - Transforms the scene as seen from the location and orientation of the pinhole of the camera to the 2D plane of the image sensor
  - Its parameters are inherent within the hardware of the camera, does not change as you move the camera)
  - Intrinsic calibration matrix
    - $\boldsymbol{K} = \begin{bmatrix} \alpha_x & -\alpha_x \cot\theta & x_O^C \\ 0 & \frac{\alpha_y}{\sin(\theta)} & y_O^C \\ 0 & 0 & 1 \end{bmatrix}$
  - ***Intrinsic parameters***
    - **Principle point, $\boldsymbol{p_{orgin}^C} = \left[\boldsymbol{x_O^C}, \boldsymbol{y_O^C}\right]^{\boldsymbol{T}}$:**

            - Offset from the left-hand corner of the origin of the optical center
        - **Axis magnification and aspect ratio, $\boldsymbol{\gamma} = [\boldsymbol{\alpha_x}, \boldsymbol{\alpha_y}]^{\boldsymbol{T}}$:**
            - Scale factors which depends on the focal length and the dimensions of each pixel sensor
        - **Skew parameter, $\boldsymbol{\theta}$:**
            - The skew parameter models the effect when the x and y axes for the camera plane aren't at 90 degrees.
        - **Internal scale factor, $\boldsymbol{\rho}$:**
            - The internal scale factor depicts the information lost in the z-axis when transforming from 3D world coordinates to 2D camera coordinates.

- **Extrinsic camera matrix, $\boldsymbol{S}$:**
    - Takes you from the world coordinate system to the coordinate system located at the pinhole of the camera and orientated to align with the axes of the image sensor
    - Extrinsic matrix
        - $\boldsymbol{S} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$
    - Extrinsic parameters
        - **Rotation matrix, $\boldsymbol{R}$:**
            - The extrinsic transformation that describes the rotation of the camera from its initial orientation at the origin of the world coordinate system to its current orientation.
            - Rotation matrix
                - $\boldsymbol{R} = R_Z(\phi)R_Y(\gamma)R_X(\psi)$
                - $\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r_1^T} \\ \boldsymbol{r_2^T} \\ \boldsymbol{r_3^T} \end{bmatrix}$
        - **Translation vector, $\boldsymbol{t}$:**
            - The translation vector that describes the position of the pinhole of the camera relative to the origin of the world coordinate system but described in terms of the camera coordinate system.
            - Translation vector:
                - $\boldsymbol{t} = \begin{bmatrix} t_X^C \\ t_Y^C \\ t_Z^C \end{bmatrix}$
- **Camera (calibration) matrix, $\boldsymbol{M}$:**
    - Transforms a point in the world coordinate frame to a point in the image plane.

- The calibration matrix combines both the intrinsic and extrinsic camera matrices.
- Calibration matrix:
  - $\boldsymbol{M} = [\boldsymbol{K} \quad \mathbf{0}] \cdot \boldsymbol{S}$
  - $\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \begin{bmatrix} \boldsymbol{m}_1^T \\ \boldsymbol{m}_2^T \\ \boldsymbol{m}_3^T \end{bmatrix}$
  - $\boldsymbol{M} = [\mathbf{A}\ \mathbf{b}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{bmatrix} == \begin{bmatrix} \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \boldsymbol{a}_3^T \end{bmatrix} & \boldsymbol{b} \end{bmatrix}$
    - $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \boldsymbol{a}_3^T \end{bmatrix}$
    - $\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$
- Transformation from a world point to a image point:
  - $\boldsymbol{p}_i^C = \boldsymbol{M}\boldsymbol{P}_i^W$

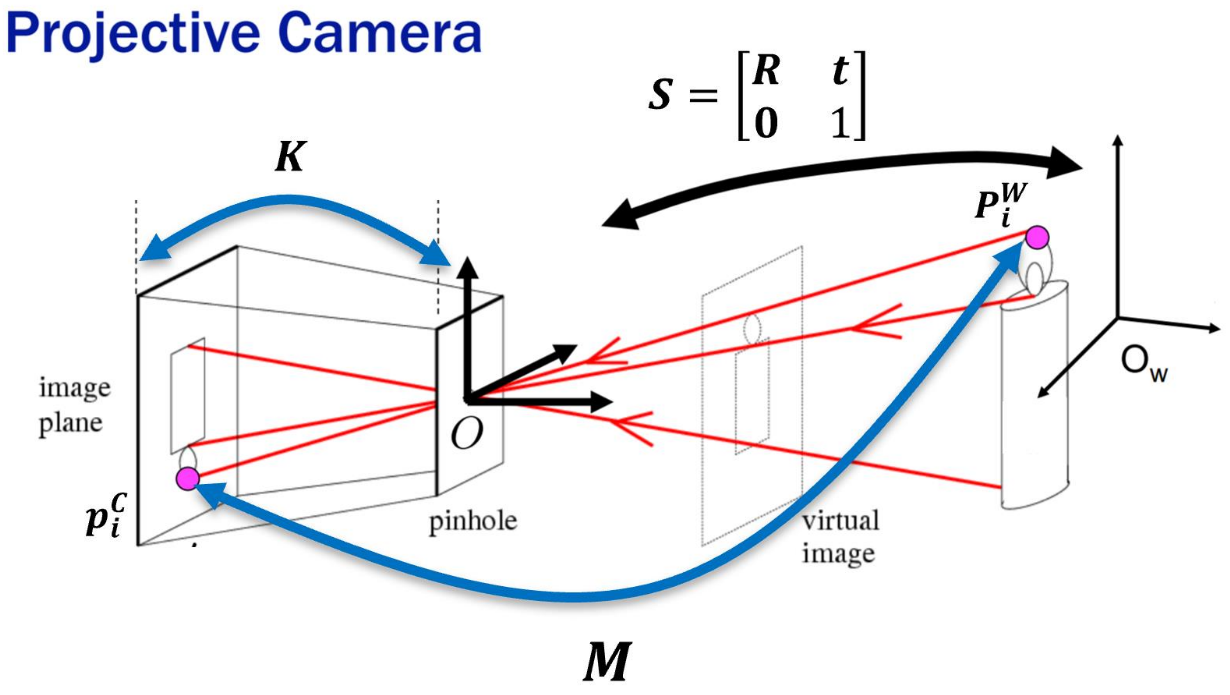


Figure 1 – Demonstration of addressing and offsets in C++

2.2 Derivation of Calibration Process

The goal of the camera calibration process is to calculate the intrinsic and extrinsic camera parameters. It is more important however to calculate the intrinsic camera parameters because these will not change as you move the camera in the world coordinate. This way if you know the intrinsic camera transformation and you can derive the extrinsic camera matrix later on in terms of variables, then you can continuously compute the locations of objects in the world coordinate frame as seen from the image coordinate plane as you move the camera in space.

To calculate the intrinsic and extrinsic camera parameters you must have a calibration rig with identifiable features whose world coordinate points are known. In our case, we will use a corner of a box with a checkered pattern on its walls and floor. We will use the corners of the squares as the identifiable features. Since we know the size of each square (~8 cm), every coordinate point of a corner should be an integer multiple of the width of the square used in the pattern. You must choose at least 6 points (I recommend 24, 8 on each plane) so that we can fit our parameters with a minimal error equation. See the following derivation and figures to better understand why and how this is done.

In this process, we are assuming we have taken a picture of n features with known world coordinates, $\boldsymbol{P_i^W}$, and we know each feature's corresponding point in the image coordinate system, $\boldsymbol{p_i^C}$, by simply looking at the picture and picking out the pixel with corresponds to that feature. Let us start with the general transformation equation from a point in the world coordinate system to the image coordinate system.

$$\boldsymbol{p_i^C} = \boldsymbol{M P_i^W}$$

Since we already know $\boldsymbol{p_i^C}$ and $\boldsymbol{P_i^W}$, the only unknown is the transformation equation $\boldsymbol{M}$. Therefore, our goal is to derive an equation that lets us solve for $\boldsymbol{M}$. To do this let us restate the previous equation in its matrix form,

$$\omega \begin{bmatrix} x_i^C \\ y_i^C \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X_i^W \\ Y_i^W \\ Z_i^W \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{m_1^T} \\ \boldsymbol{m_2^T} \\ \boldsymbol{m_3^T} \end{bmatrix} \boldsymbol{P_i^W}$$

$$\begin{bmatrix} \omega x_i^C \\ \omega y_i^C \\ \omega \end{bmatrix} = \begin{bmatrix} \boldsymbol{m_1^T P_i^W} \\ \boldsymbol{m_2^T P_i^W} \\ \boldsymbol{m_3^T P_i^W} \end{bmatrix}$$

If we expand this equation, multiplying through we get

$$\omega x_i^C = m_{11} X_i^W + m_{12} Y_i^W + m_{13} Z_i^W + m_{14}$$
$$\omega y_i^C = m_{21} X_i^W + m_{22} Y_i^W + m_{23} Z_i^W + m_{24}$$
$$\omega = m_{31} X_i^W + m_{32} Y_i^W + m_{33} Z_i^W + m_{34}$$

Here we can see that the collapsed information of the Z dimension in the image coordinate system equates to

$$\omega = m_{31}X_i^W + m_{32}Y_i^W + m_{33}Z_i^W + m_{34}$$

$$\omega = \boldsymbol{m_3^T P_i^W}$$

Therefore, if we replace ω by $\boldsymbol{m_3^T P_i^W}$ in the equations for ωx_i^C and ωy_i^C we can eliminate the dependence on the scale factor, ω,

$$\begin{bmatrix} \omega x_i^C \\ \omega y_i^C \\ \omega \end{bmatrix} = \begin{bmatrix} \boldsymbol{m_1^T P_i^W} \\ \boldsymbol{m_2^T P_i^W} \\ \boldsymbol{m_3^T P_i^W} \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{m_3^T P_i^W} x_i^C \\ \boldsymbol{m_3^T P_i^W} y_i^C \end{bmatrix} = \begin{bmatrix} \boldsymbol{m_1^T P_i^W} \\ \boldsymbol{m_2^T P_i^W} \end{bmatrix}$$

Expanding on this equation into its component form we get,

$$\begin{bmatrix} m_{31}x_i^C X_i^W + m_{32}x_i^C Y_i^W + m_{33}\, x_i^C Z_i^W + m_{34}x_i^C \\ m_{31}y_i^C X_i^W + m_{32}y_i^C Y_i^W + m_{33}y_i^C\, Z_i^W + m_{34}y_i^C \end{bmatrix} = \begin{bmatrix} m_{11}X_i^W + m_{12}Y_i^W + m_{13}Z_i^W + m_{14} \\ m_{21}X_i^W + m_{22}Y_i^W + m_{23}Z_i^W + m_{24} \end{bmatrix}$$

Let us assume that $m_{34} = 1$, which is equivalent to assuming that the scale factor is 1. Doing this we can simplify the previous equation to,

$$\begin{bmatrix} m_{31}x_i^C X_i^W + m_{32}x_i^C Y_i^W + m_{33}\, x_i^C Z_i^W + x_i^C \\ m_{31}y_i^C X_i^W + m_{32}y_i^C Y_i^W + m_{33}y_i^C\, Z_i^W + y_i^C \end{bmatrix} = \begin{bmatrix} m_{11}X_i^W + m_{12}Y_i^W + m_{13}Z_i^W + m_{14} \\ m_{21}X_i^W + m_{22}Y_i^W + m_{23}Z_i^W + m_{24} \end{bmatrix}$$

Now we can move all of the terms with unknown parameters, i.e. the terms with a component of $\boldsymbol{M}$, onto the right-hand side of the equation.

$$\begin{bmatrix} x_i^C \\ y_i^C \end{bmatrix} = \begin{bmatrix} m_{11}X_i^W + m_{12}Y_i^W + m_{13}Z_i^W + m_{14} - m_{31}x_i^C X_i^W - m_{32}x_i^C Y_i^W - m_{33}\, x_i^C Z_i^W \\ m_{21}X_i^W + m_{22}Y_i^W + m_{23}Z_i^W + m_{24} - m_{31}y_i^C X_i^W - m_{32}y_i^C Y_i^W - m_{33}y_i^C\, Z_i^W \end{bmatrix}$$

We will now do a slight of hand to reformat the equations for x_i^C and y_i^C to include all the terms of the unknown transformation matrix $\boldsymbol{M}$, so that we can derive the vectorized form of it. This would entail adding the unused components of $\boldsymbol{M}$ in the equations above and multiplying them by zero. This would look like the following,

$$\begin{bmatrix} x_i^C \\ y_i^C \end{bmatrix} = \begin{bmatrix} m_{11}X_i^W + m_{12}Y_i^W + m_{13}Z_i^W + m_{14} + 0 \cdot m_{21} + 0 \cdot m_{22} + 0 \cdot m_{23} + 0 \cdot m_{24} - m_{31}x_i^C X_i^W - m_{32}x_i^C Y_i^W - m_{33}\, x_i^C Z_i^W \\ 0 \cdot m_{11} + 0 \cdot m_{12} + 0 \cdot m_{13} + 0 \cdot m_{14} + m_{21}X_i^W + m_{22}Y_i^W + m_{23}Z_i^W + m_{24} - m_{31}y_i^C X_i^W - m_{32}y_i^C Y_i^W - m_{33}y_i^C\, Z_i^W \end{bmatrix}$$

The vectorized form of matrix $\boldsymbol{M}$ is denoted as

$$\vec{\boldsymbol{m}} = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}$$

With this information we now can factor out the unknown components from the known data points in the above equations,

$$\begin{bmatrix} x_i^C \\ y_i^C \end{bmatrix} = \begin{bmatrix} X_i^W & Y_i^W & Z_i^W & 1 & 0 & 0 & 0 & 0 & -x_i^C X_i^W & -x_i^C Y_i^W & -x_i^C Z_i^W \\ 0 & 0 & 0 & 0 & X_i^W & Y_i^W & Z_i^W & 1 & -y_i^C X_i^W & -y_i^C Y_i^W & -y_i^C Z_i^W \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}$$

Let us further simplify this equation by recognizing that $\boldsymbol{P_i^W} = [X_i^W, Y_i^W, Z_i^W, 1]^{\boldsymbol{T}}$, $\boldsymbol{P_i'} = [X_i^W, Y_i^W, Z_i^W]^T$, and that a pixel is defined as $\boldsymbol{p_i'} = \left[\, x_i^C, y_i^C \right]^T$. Let us also define a zero vector as $\vec{\boldsymbol{0}} = [0,0,0,0]^T$, then we can reduce the above equation to,

$$\boldsymbol{p_i'} = \left[\begin{bmatrix} \boldsymbol{P_i^{W^T}} & \vec{\boldsymbol{0}}^{\boldsymbol{T}} \\ \vec{\boldsymbol{0}}^{\boldsymbol{T}} & \boldsymbol{P_i^{W^T}} \end{bmatrix} \middle| \left[-(\boldsymbol{p_i'}) \boldsymbol{P_i'}^{\mathrm{T}} \right] \right] \; \vec{\boldsymbol{m}}$$

For convenience, let us define the ***coupled data matrix*** in the above equation as a function of the i^{th} feature $\boldsymbol{P_i'}$ with a corresponding pixel $\boldsymbol{p_i'}$ in an image so,

$$\boldsymbol{d_i}(\boldsymbol{P_i'}, \boldsymbol{p_i'}) \; = \left[\begin{bmatrix} \boldsymbol{P_i^{W^T}} & \vec{\boldsymbol{0}}^{\boldsymbol{T}} \\ \vec{\boldsymbol{0}}^{\boldsymbol{T}} & \boldsymbol{P_i^{W^T}} \end{bmatrix} \middle| \left[-(\boldsymbol{p_i'}) \boldsymbol{P_i'}^{\mathrm{T}} \right] \right]$$

Then we can provide one more level of simplification for the transformation equation,

$$\boldsymbol{p_i'} = \boldsymbol{d}(\boldsymbol{P_i'}, \boldsymbol{p_i'}) * \; \vec{\boldsymbol{m}}$$

The above equation is used as a model for determining the pixel points of features, which can be expressed in the following manner. For every distinct feature, $\boldsymbol{P'_i}$, represented as a point, $\boldsymbol{P_i^W}$, in the world coordinate system that is captured in an image by a pinhole camera, there exists a transformation $\boldsymbol{M}$, with constant parameters $\overrightarrow{\boldsymbol{m}}$, that takes that feature from the world coordinate system to a unique pixel, $\boldsymbol{p'_i}$. This means that if we have n features captured in an image the transformation for every feature into a unique pixel should be exactly the same and can be modeled by a linear system of equations as such,

$$\begin{bmatrix} \boldsymbol{p'_1} \\ \boldsymbol{p'_2} \\ \vdots \\ \boldsymbol{p'_i} \\ \vdots \\ \boldsymbol{p'_n} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d(P'_1, p'_1)} \\ \boldsymbol{d(P'_2, p'_2)} \\ \vdots \\ \boldsymbol{d(P'_i, p'_i)} \\ \vdots \\ \boldsymbol{d(P'_n, p'_n)} \end{bmatrix} \overrightarrow{\boldsymbol{m}}$$

Which is equivalent to,

$$\begin{bmatrix} x_1^C \\ y_1^C \\ \vdots \\ x_i^C \\ y_i^C \\ \vdots \\ x_n^C \\ y_n^C \end{bmatrix} = \begin{bmatrix} X_1^W & Y_1^W & Z_1^W & 1 & 0 & 0 & 0 & 0 & -x_1^C X_1^W & -x_1^C Y_1^W & -x_1^C Z_1^W \\ 0 & 0 & 0 & 0 & X_1^W & Y_1^W & Z_1^W & 1 & -y_1^C X_1^W & -y_1^C Y_1^W & -y_1^C Z_1^W \\ & & & & & & \vdots & & & & \\ X_i^W & Y_i^W & Z_i^W & 1 & 0 & 0 & 0 & 0 & -x_i^C X_i^W & -x_i^C Y_i^W & -x_i^C Z_i^W \\ 0 & 0 & 0 & 0 & X_i^W & Y_i^W & Z_i^W & 1 & -y_i^C X_i^W & -y_i^C Y_i^W & -y_i^C Z_i^W \\ & & & & & & \vdots & & & & \\ X_n^W & Y_n^W & Z_n^W & 1 & 0 & 0 & 0 & 0 & -x_n^C X_n^W & -x_n^C Y_n^W & -x_n^C Z_n^W \\ 0 & 0 & 0 & 0 & X_n^W & Y_n^W & Z_n^W & 1 & -y_n^C X_n^W & -y_n^C Y_n^W & -y_n^C Z_n^W \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}$$

Let us denote the vector of pixels as $\mathcal{p}$ and the matrix of couple data matrices as $\mathcal{D}$ so that,

$$\mathcal{p} = \mathcal{D}\overrightarrow{\boldsymbol{m}}$$

Where

$$\mathcal{p} = \begin{bmatrix} \boldsymbol{p'_1} \\ \boldsymbol{p'_2} \\ \vdots \\ \boldsymbol{p'_i} \\ \vdots \\ \boldsymbol{p'_n} \end{bmatrix}$$

$$\mathcal{D} = \begin{bmatrix} \boldsymbol{d(P'_1, p'_1)} \\ \boldsymbol{d(P'_2, p'_2)} \\ \vdots \\ \boldsymbol{d(P'_i, p'_i)} \\ \vdots \\ \boldsymbol{d(P'_n, p'_n)} \end{bmatrix}$$

Since we want to solve for $\vec{\boldsymbol{m}}$ which has ℓ unknown parameters where $\ell = 11$, we need at the very least $n = \frac{\ell}{2} = 6$ features to be able to simultaneously solve $\vec{\boldsymbol{m}}$ using $\mathcal{p} = \mathcal{D}\vec{\boldsymbol{m}}$. However, this requires that $\mathcal{D}$ be square and invertible, which is not often the case and when it is it can result in massive overfitting of the camera model depending on the features you choose. Therefore, to avoid these issues we will use more than $\frac{\ell}{2}$ features so that we can fit our model, $\vec{\boldsymbol{m}}$, to our features in a manner that minimizes the squared error. This process is often denoted as linear least squared error fitting.

Let us assume we have a true vector of known pixels which represent features denoted as $\mathcal{p}_t$. Let us also assume that we have a model of our camera which provides us an estimate of the pixels, denoted as $\tilde{\mathcal{p}} = \mathcal{D}\vec{\boldsymbol{m}}$ for the same set of features. Then we can compute the error between the true values and the estimated values of the pixels as,

$$\epsilon = \mathcal{p}_t - \tilde{\mathcal{p}}$$

Now to minimize the squared error of our estimation generated from our model, we would create a cost function $\mathcal{J}$ in which we want to minimize with respect to our model, i.e.,

$$\mathcal{J} = \int \frac{1}{2}\epsilon^2 d\vec{\boldsymbol{m}}$$

If we take the partial derivative of $\mathcal{J}$ with respect to $\vec{\boldsymbol{m}}$ then we find the value of $\vec{\boldsymbol{m}}$ which produces a minimum cost, i.e.,

$$\frac{\partial \mathcal{J}}{\partial \vec{\boldsymbol{m}}} = 0$$

Substitute the integrand for $\mathcal{J}$ and evaluate

$$0 = \frac{1}{2}\frac{\partial(\epsilon^2)}{\partial \vec{\boldsymbol{m}}} = \frac{2}{2}\epsilon\frac{\partial \epsilon}{\partial \vec{\boldsymbol{m}}}$$

Substitute $\epsilon = \mathcal{p}_t - \tilde{\mathcal{p}}$,

$$0 = (\mathcal{p}_t - \tilde{\mathcal{p}})\frac{\partial(\mathcal{p}_t - \tilde{\mathcal{p}})}{\partial \vec{\boldsymbol{m}}}$$

$$0 = (\mathcal{p}_t - \tilde{\mathcal{p}})\left(\frac{\partial \mathcal{p}_t}{\partial \vec{\boldsymbol{m}}} - \frac{\partial \tilde{\mathcal{p}}}{\partial \vec{\boldsymbol{m}}}\right)$$

Substitute $\tilde{\mathcal{p}} = \mathcal{D}\vec{\boldsymbol{m}}$,

$$0 = (\mathcal{p}_t - \mathcal{D}\vec{\boldsymbol{m}})\left(0 - \frac{\partial(\mathcal{D}\vec{\boldsymbol{m}})}{\partial \vec{\boldsymbol{m}}}\right)$$

$$0 = (\mathcal{p}_t - \mathcal{D}\vec{\boldsymbol{m}})(-\mathcal{D})$$

Rearrange the lone $\mathcal{D}$ term,

$$0 = -(\mathcal{D}^T)(\wp_t - \mathcal{D}\vec{\boldsymbol{m}})$$

$$0 = -\mathcal{D}^T\wp_t + \mathcal{D}^T\mathcal{D}\vec{\boldsymbol{m}}$$

Get unknown parameter vector by itself and solve for $\vec{\boldsymbol{m}}$,

$$\mathcal{D}^T\mathcal{D}\vec{\boldsymbol{m}} = \mathcal{D}^T\wp_t$$

$$\boxed{\vec{\boldsymbol{m}} = (\mathcal{D}^T\mathcal{D})^{-1}\mathcal{D}^T\wp_t}$$

Once we solve for the unknown parameters we can then solve for our intrinsic and extrinsic parameters. First, recall that

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = [\boldsymbol{K} \quad \mathbf{0}] \cdot \boldsymbol{S} = [\mathbf{A}\ \mathbf{b}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{a_1^T} \\ \boldsymbol{a_2^T} \\ \boldsymbol{a_3^T} \end{bmatrix} & \boldsymbol{b} \end{bmatrix}$$

$$\boldsymbol{K} = \begin{bmatrix} \alpha_x & -\alpha_x \cot\theta & x_O^C \\ 0 & \dfrac{\alpha_y}{\sin(\theta)} & y_O^C \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\boldsymbol{t} = \begin{bmatrix} t_X^C \\ t_Y^C \\ t_Z^C \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r_1^T} \\ \boldsymbol{r_2^T} \\ \boldsymbol{r_3^T} \end{bmatrix}$$

Then the intrinsic parameters are as follows,

$$\rho = \frac{\mp 1}{\|\boldsymbol{a_3}\|}$$

$$x_O^C = \rho^2(\boldsymbol{a}_1 \circ \boldsymbol{a_3})$$

$$y_O^C = \rho^2(\boldsymbol{a_2} \circ \boldsymbol{a_3})$$

$$\theta = \operatorname{acos}\left[-\frac{(\boldsymbol{a}_1 \times \boldsymbol{a_3}) \circ (\boldsymbol{a}_2 \times \boldsymbol{a_3})}{\|\boldsymbol{a}_1 \times \boldsymbol{a_3}\| \cdot \|\boldsymbol{a}_2 \times \boldsymbol{a_3}\|}\right]$$

$$\alpha_x = \rho^2\|\boldsymbol{a}_1 \times \boldsymbol{a_3}\| \sin\theta$$

$$\alpha_y = \rho^2\|\boldsymbol{a}_2 \times \boldsymbol{a_3}\| \sin\theta$$

And our extrinsic parameters are as follows,

$$\boldsymbol{r_3} = \rho \boldsymbol{a_3}$$

$$\boldsymbol{r_1} = \frac{(\boldsymbol{a}_2 \times \boldsymbol{a_3})}{\|\boldsymbol{a}_2 \times \boldsymbol{a_3}\|}$$

$$\boldsymbol{r_2} = (\boldsymbol{r}_3 \times \boldsymbol{r_1})$$

$$\boldsymbol{t} = \rho \boldsymbol{K^{-1} b}$$

In which $\times$ is the cross-product, $\circ$ is the dot product, and $\|\boldsymbol{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2}$. To check the error of your model you should calculate the mean square error. To do this you first calculate the error for each of your points via,

$$\boldsymbol{\epsilon_i} = \boldsymbol{p_i^C} - \widetilde{\boldsymbol{p_i^C}} = \boldsymbol{p_i^C} - \boldsymbol{M P_i^W}$$

Then calculate your mean square error by,

$$\boldsymbol{MSE} = \frac{\mathbf{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \boldsymbol{\epsilon_i^T \epsilon_i}$$

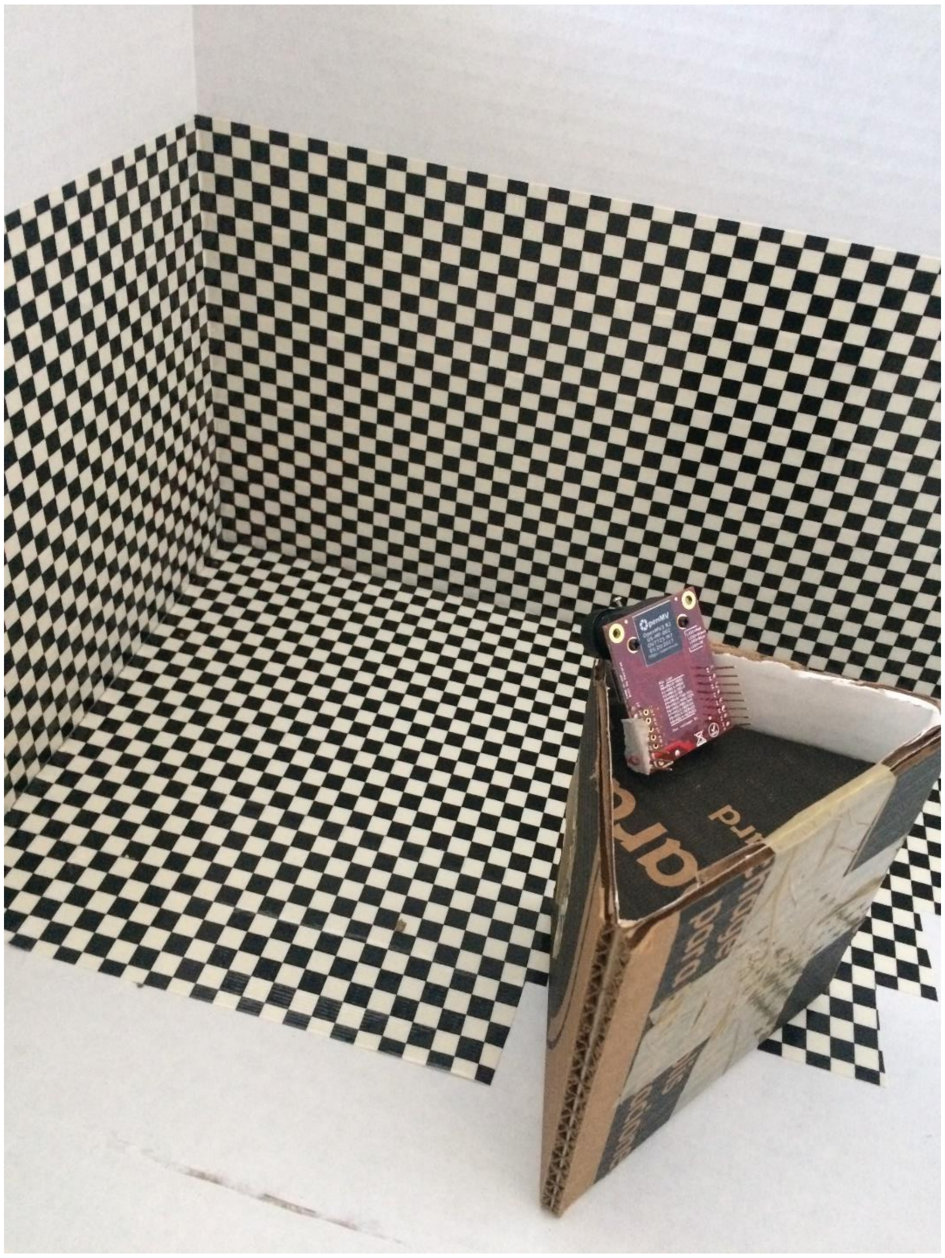


Figure 2 – Camera Calibration Jig Setup

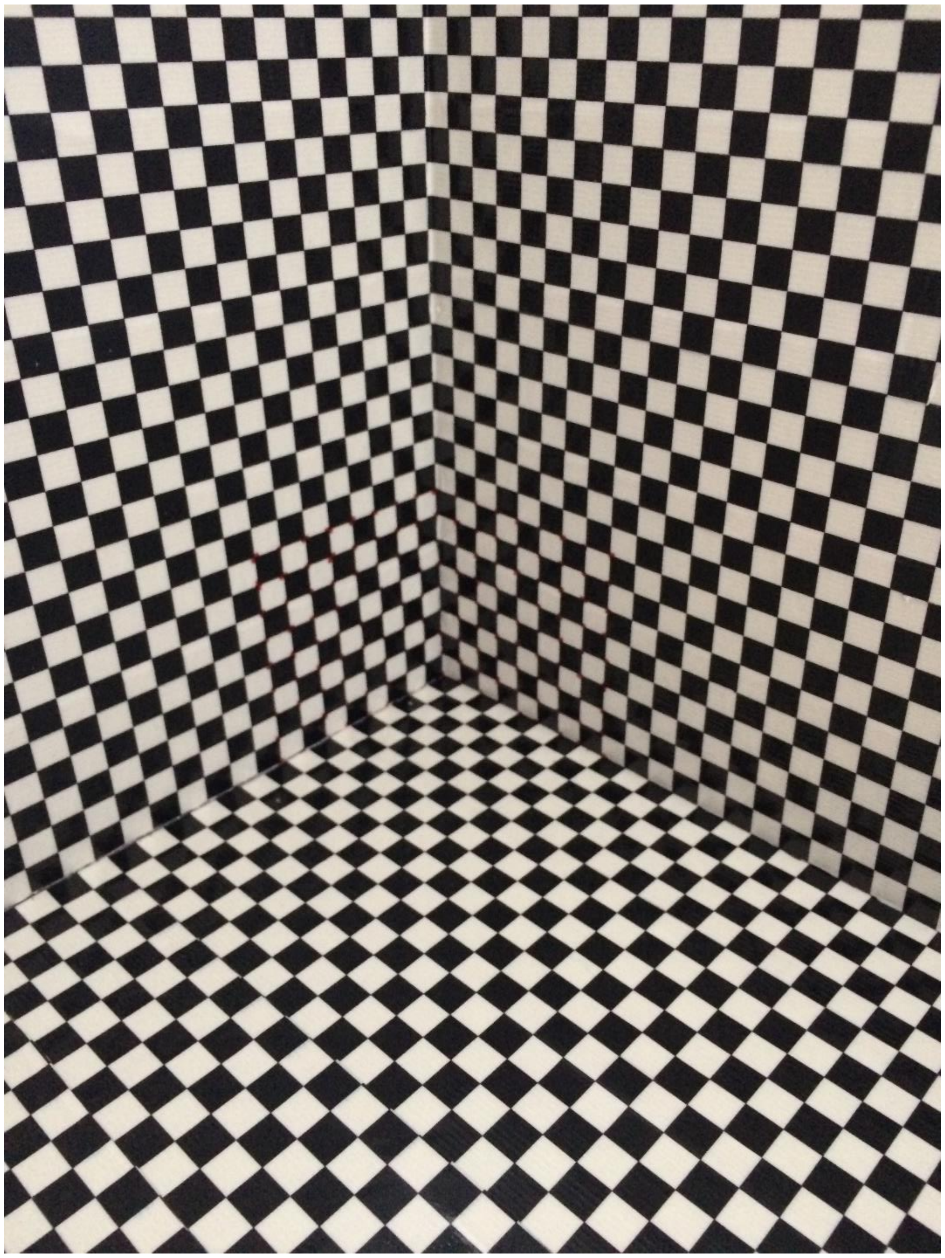


Figure 3 – Photo from camera perspective of features

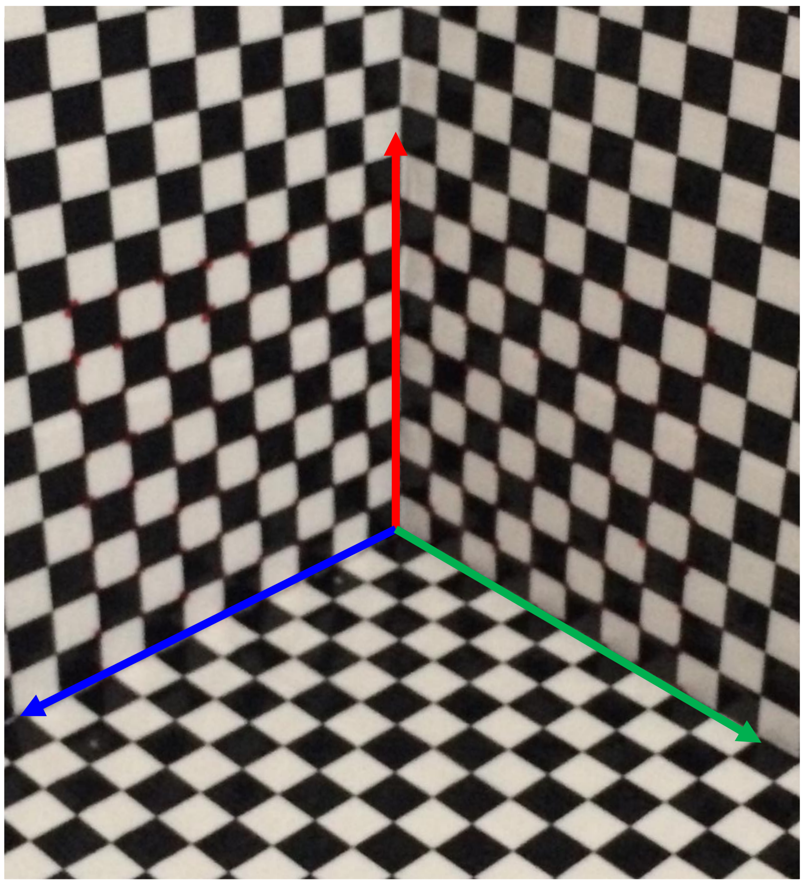


Figure 4 – Close up of 3D coordinate system from camera perspective

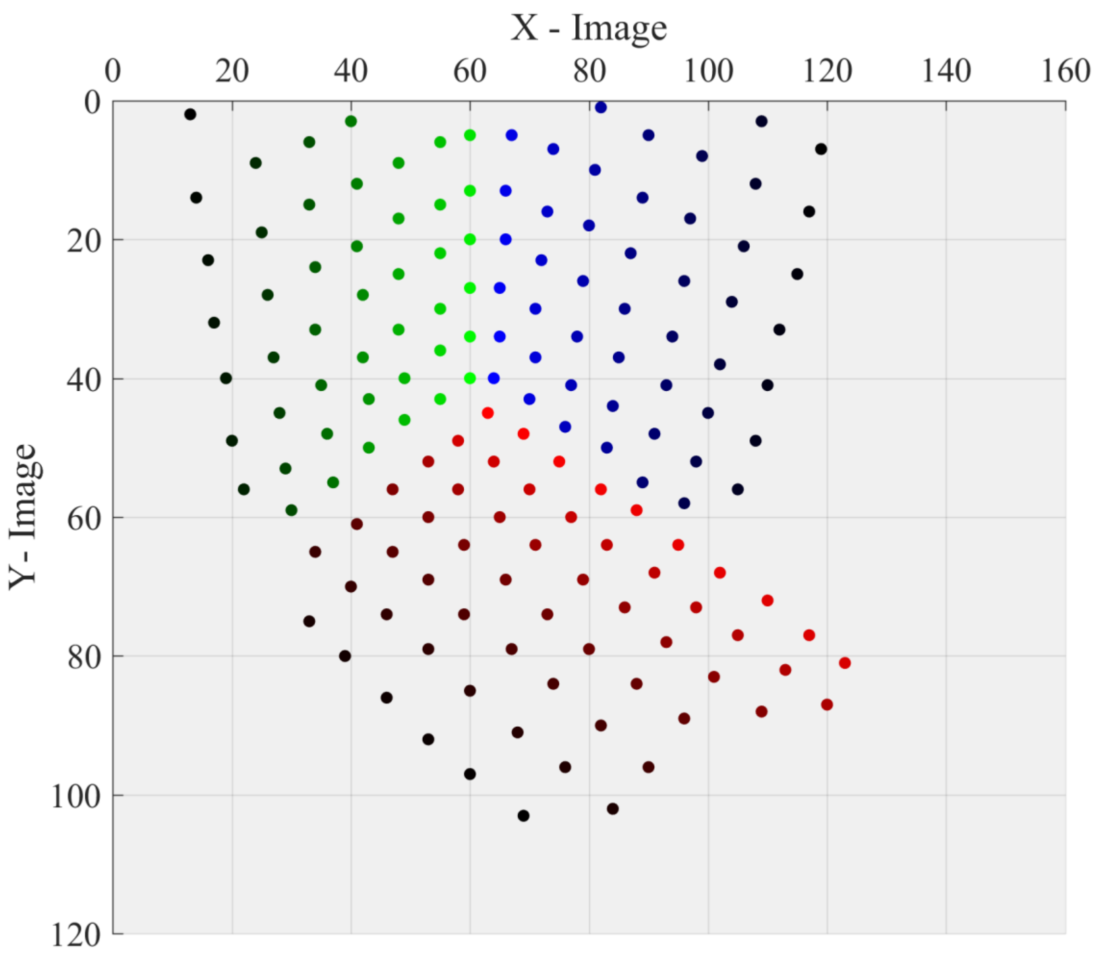


Figure 5 – Pixel points of features in image.

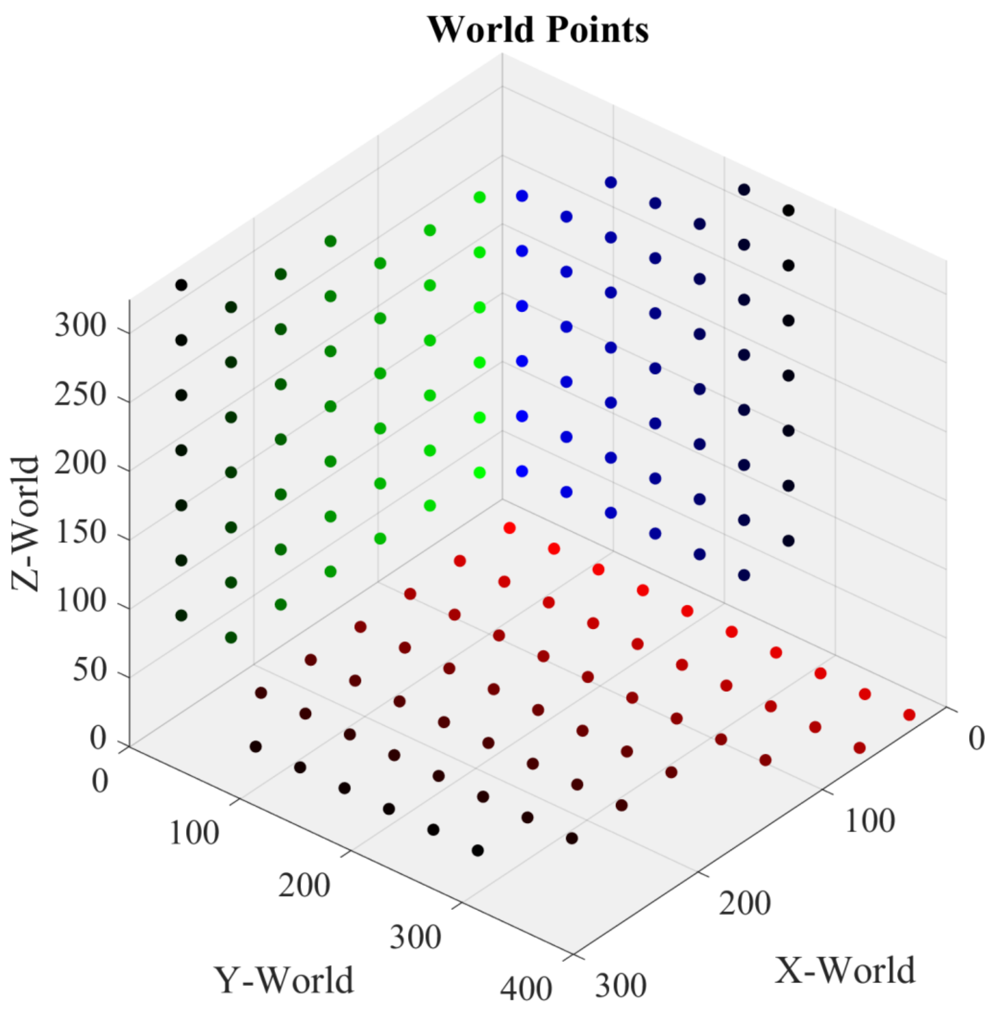


Figure 6 – 3D feature points

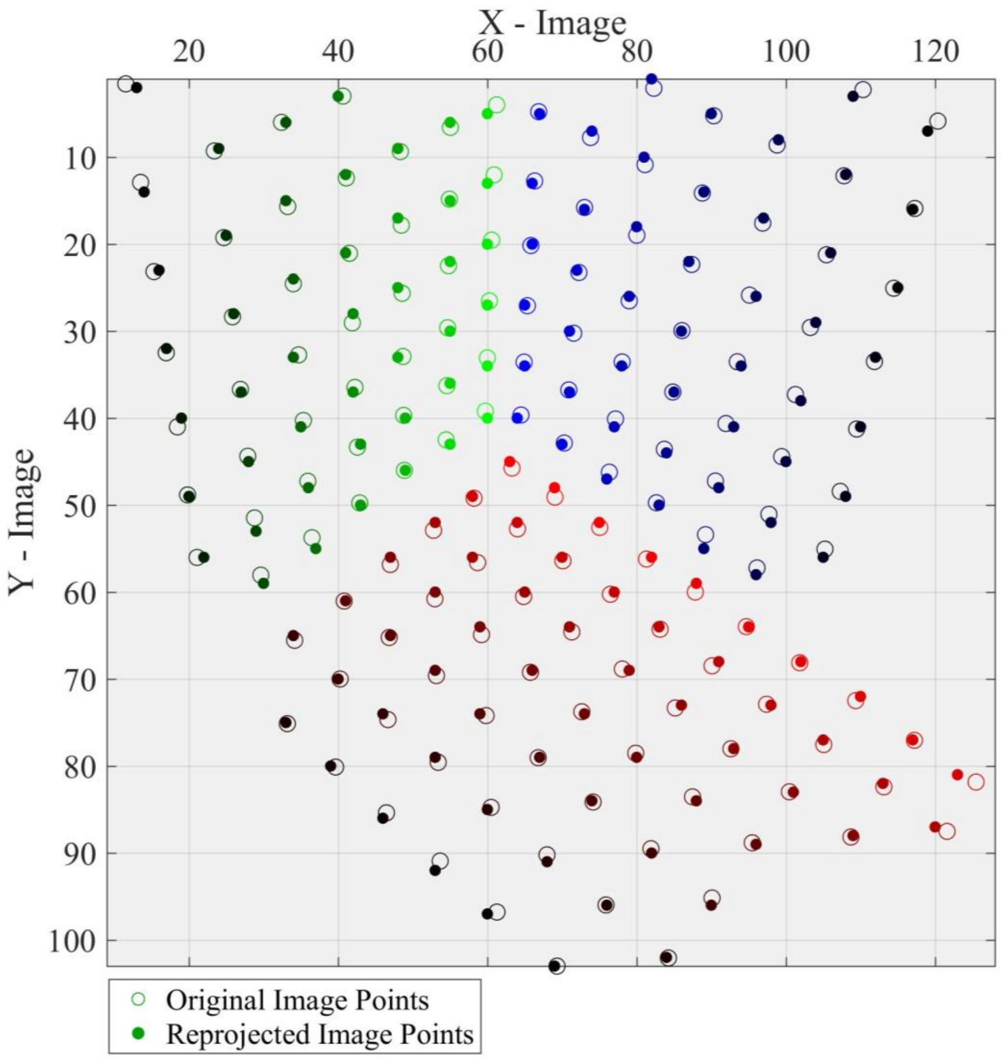
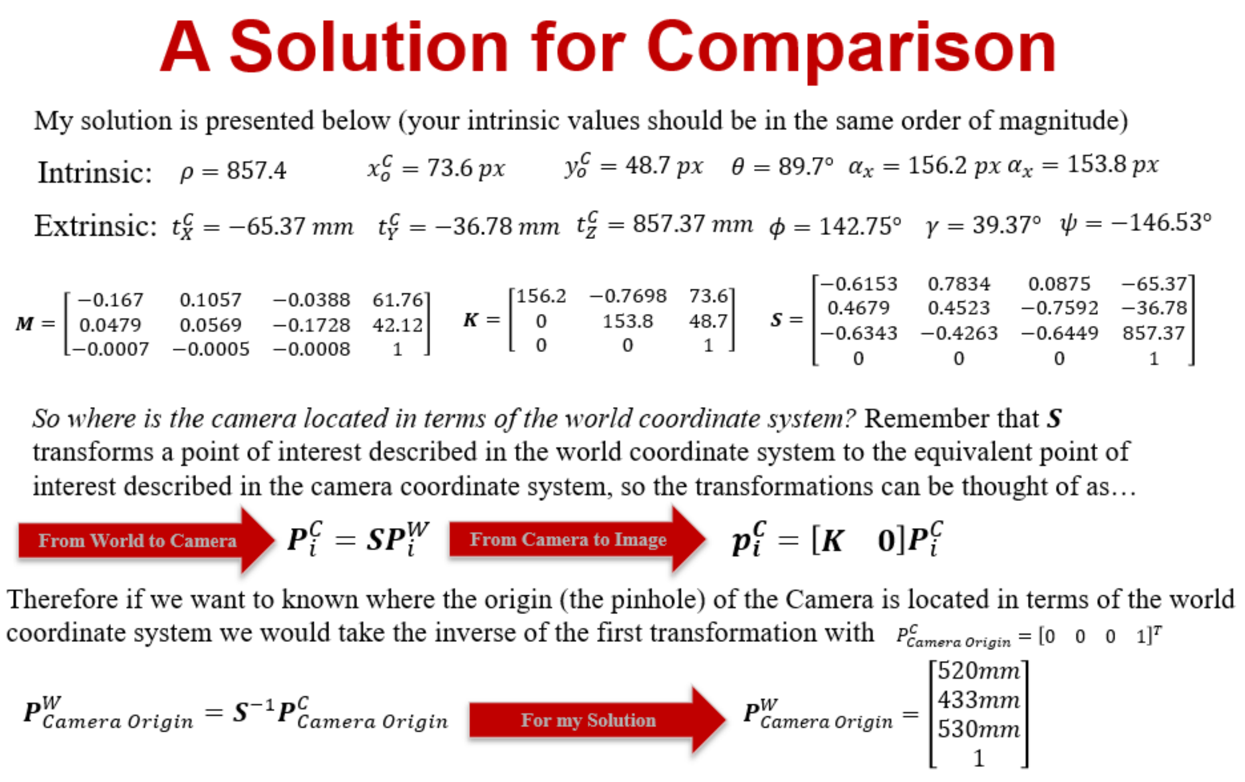


Figure 7 – Demonstration of error in the model fitting



A Solution for Comparison

My solution is presented below (your intrinsic values should be in the same order of magnitude)

Intrinsic: $\rho = 857.4$ $\quad x_o^C = 73.6\,px$ $\quad y_o^C = 48.7\,px$ $\quad \theta = 89.7°$ $\quad \alpha_x = 156.2\,px$ $\quad \alpha_x = 153.8\,px$

Extrinsic: $t_X^C = -65.37\,mm$ $\quad t_Y^C = -36.78\,mm$ $\quad t_Z^C = 857.37\,mm$ $\quad \phi = 142.75°$ $\quad \gamma = 39.37°$ $\quad \psi = -146.53°$

$$\boldsymbol{M} = \begin{bmatrix} -0.167 & 0.1057 & -0.0388 & 61.76 \\ 0.0479 & 0.0569 & -0.1728 & 42.12 \\ -0.0007 & -0.0005 & -0.0008 & 1 \end{bmatrix} \quad \boldsymbol{K} = \begin{bmatrix} 156.2 & -0.7698 & 73.6 \\ 0 & 153.8 & 48.7 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{S} = \begin{bmatrix} -0.6153 & 0.7834 & 0.0875 & -65.37 \\ 0.4679 & 0.4523 & -0.7592 & -36.78 \\ -0.6343 & -0.4263 & -0.6449 & 857.37 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So where is the camera located in terms of the world coordinate system? Remember that $\boldsymbol{S}$ transforms a point of interest described in the world coordinate system to the equivalent point of interest described in the camera coordinate system, so the transformations can be thought of as…

$$\xrightarrow{\text{From World to Camera}} \boldsymbol{P}_i^C = \boldsymbol{S}\boldsymbol{P}_i^W \xrightarrow{\text{From Camera to Image}} \boldsymbol{p}_i^C = [\boldsymbol{K} \quad \boldsymbol{0}]\boldsymbol{P}_i^C$$

Therefore if we want to known where the origin (the pinhole) of the Camera is located in terms of the world coordinate system we would take the inverse of the first transformation with $P_{Camera\ Origin}^C = [0 \quad 0 \quad 0 \quad 1]^T$

$$\boldsymbol{P}_{Camera\ Origin}^W = \boldsymbol{S}^{-1}\boldsymbol{P}_{Camera\ Origin}^C \xrightarrow{\text{For my Solution}} \boldsymbol{P}_{Camera\ Origin}^W = \begin{bmatrix} 520mm \\ 433mm \\ 530mm \\ 1 \end{bmatrix}$$

Figure 8 – Comparative solution but from a different calibration jig

3 References & Resources

3.1 References

The following references may be helpful to complete the lab.

1. http://docs.openmv.io/openmvcam/quickref.html
2. http://docs.openmv.io/openmvcam/tutorial/index.html
3. http://docs.openmv.io/library/index.html
4. http://docs.openmv.io/openmvcam/tutorial/software_setup.html
5. https://docs.python.org/3/tutorial/classes.html

4 Pre-lab

None.

5 Laboratory

Save all code, data, and solutions as you will need them to complete the final lab report.

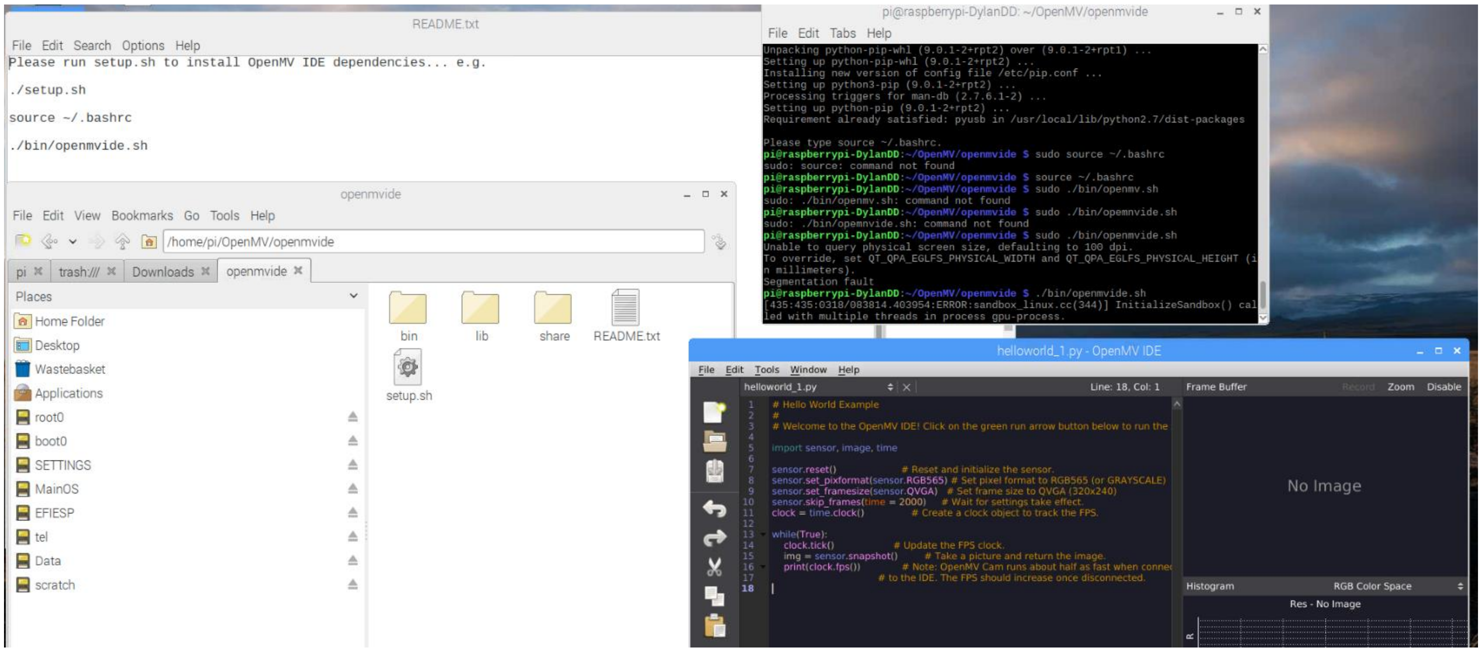
5.1 Requirements

1. Bring your raspberry pi (RPI) so that you can develop your code.
2. Grab an OpenMV Camera, make sure you label it because you will need to use this exact one for the rest of the year.
3. Micro USB data cable.

5.2 Procedure

1. Download the OpenMV software on your RPi
   a. http://github.com/openmv/openmv-ide/releases/download/v1.8.1/openmv-ide-linux-arm-1.8.1.tar.gz
2. Follow the install instructions given here
   a. http://docs.openmv.io/openmvcam/tutorial/software_setup.html
3. If you go to connect your openMV camera with the Rpi through the usb ports, make sure you have a “data” usb-micro cable, some of them are power only, this happened to me.
4. Once you have the files downloaded to your Rpi and extracted, open the README.txt and follow the instructions. (see screen shot below)
5. Once you have the OpenMV IDE up and running, plug the micro usb cable into your RPi. If everything works, the Rpi will recognize the OpenMV Cam as an external removable device and a file manager will pop up and the Cam will double-blink blue.
6. Click the Plug Icon to connect to the Cam (you may have to click twice)
7. Click area indicated by (2) in the figures below to update firmware.
8. Accept the popup boxes.
9. Play around with the camera and get to know the IDE.
10. To launch the python script to the camera, press the green play button.
11. If needed, adjust the lens to focus the picture.
12. When ready to begin the calibration process go over to the jig and follow the following instructions
    a. Have your camera plugged into your RPI.

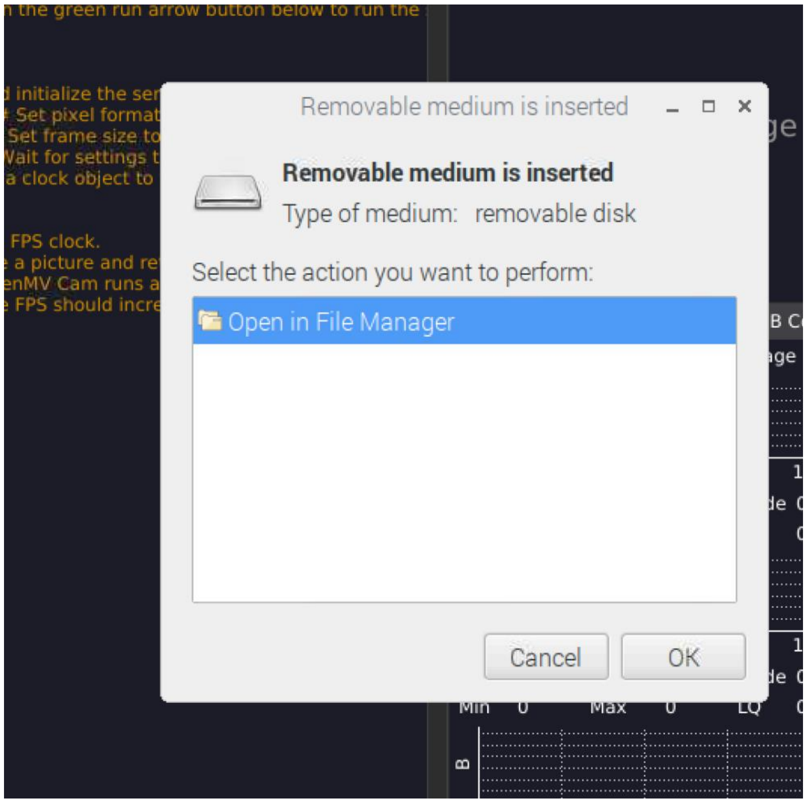
b. Place your camera on the triangle mount and adjust it so that it is looking at the origin (similar to the setup in the figures above)
c. Try to make you have enough resolution of the origin, if not, move the triangle mount closer.
d. You only want to use the area right around the origin to calibrate your camera.
e. When you think you have a good setup, go ahead and run the python script given to you.
f. This will save an image to the memory of the camera and you will need to open the camera’s memory on the RPI in the file explorer where it lists the removeable devices.
g. You will then extract the image from the camera to the RPI and delete the image from the camera’s memory (the camera has limited storage).
h. Export the image to MATLAB and display the image in a new figure.
i. Manually select the corners of the squares in your image, holding shift+click to place a new data point in the image. When finished selecting all of your data points, right-click and select export data to workspace.
j. Make sure you select your points in a methodical manner to easily generate the corresponding matrix of world coordinate points later.
k. The data will be exported as the last point selected in the image being the first row in the point matrix.
l. Using MATLAB, generate the $\mathcal{D}$ and $\wp$ data structures that were described in the beginning of this document and calculate the intrinsic and extrinsic parameters.
m. Calculate your mean squared error.
n. Save all results, calculations, code, and data for the final report. You will also need these results to be able to do the rest of the labs for this year.
o. After calibrating your camera, it is important to not adjust the lens at all for the rest of the year or else you will have to recalibrate it.
p. When finished show to GTA your results.



```
Please run setup.sh to install OpenMV IDE dependencies... e.g.

./setup.sh

source ~/.bashrc

./bin/openmvide.sh
```

```
Unpacking python-pip-whl (9.0.1-2+rpt2) over (9.0.1-2+rpt1) ...
Setting up python-pip-whl (9.0.1-2+rpt2) ...
Installing new version of config file /etc/pip.conf ...
Setting up python3-pip (9.0.1-2+rpt2) ...
Processing triggers for man-db (2.7.6.1-2) ...
Setting up python-pip (9.0.1-2+rpt2) ...
Requirement already satisfied: pyusb in /usr/local/lib/python2.7/dist-packages

Please type source ~/.bashrc.
pi@raspberrypi-DylanDD:~/OpenMV/openmvide $ sudo source ~/.bashrc
sudo: source: command not found
pi@raspberrypi-DylanDD:~/OpenMV/openmvide $ source ~/.bashrc
pi@raspberrypi-DylanDD:~/OpenMV/openmvide $ sudo ./bin/openmv.sh
sudo: ./bin/openmv.sh: command not found
pi@raspberrypi-DylanDD:~/OpenMV/openmvide $ sudo ./bin/opemnvide.sh
sudo: ./bin/opemnvide.sh: command not found
pi@raspberrypi-DylanDD:~/OpenMV/openmvide $ sudo ./bin/openmvide.sh
Unable to query physical screen size, defaulting to 100 dpi.
To override, set QT_QPA_EGLFS_PHYSICAL_WIDTH and QT_QPA_EGLFS_PHYSICAL_HEIGHT (i
n millimeters).
Segmentation fault
pi@raspberrypi-DylanDD:~/OpenMV/openmvide $ ./bin/openmvide.sh
[435:435:0318/083814.403954:ERROR:sandbox_linux.cc(344)] InitializeSandbox() cal
led with multiple threads in process gpu-process.
```

```python
# Hello World Example
#
# Welcome to the OpenMV IDE! Click on the green run arrow button below to run the

import sensor, image, time

sensor.reset()                      # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565 (or GRAYSCALE)
sensor.set_framesize(sensor.QVGA)   # Set frame size to QVGA (320x240)
sensor.skip_frames(time = 2000)     # Wait for settings take effect.
clock = time.clock()                # Create a clock object to track the FPS.

while(True):
    clock.tick()                    # Update the FPS clock.
    img = sensor.snapshot()         # Take a picture and return the image.
    print(clock.fps())              # Note: OpenMV Cam runs about half as fast when connec
                                    # to the IDE. The FPS should increase once disconnected.
```

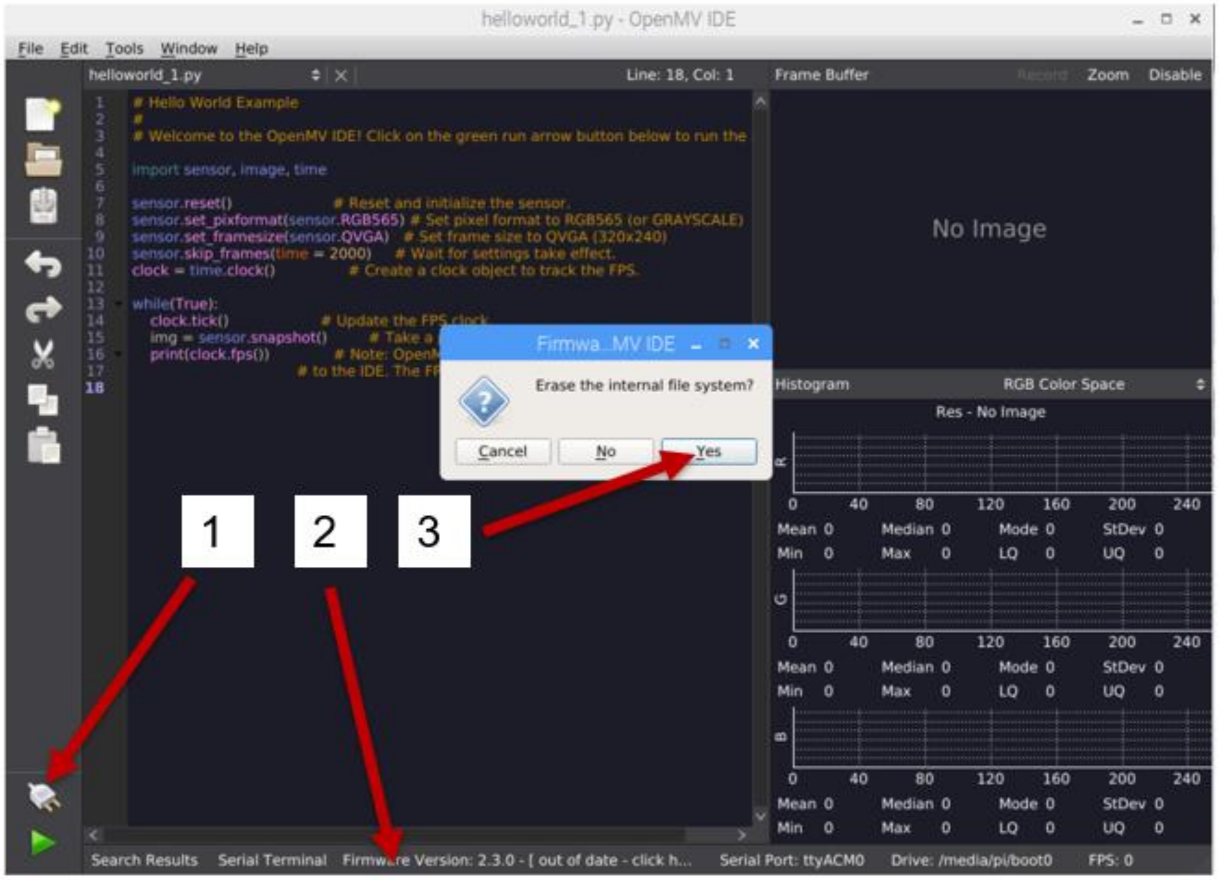


Removable medium is inserted

Type of medium: removable disk

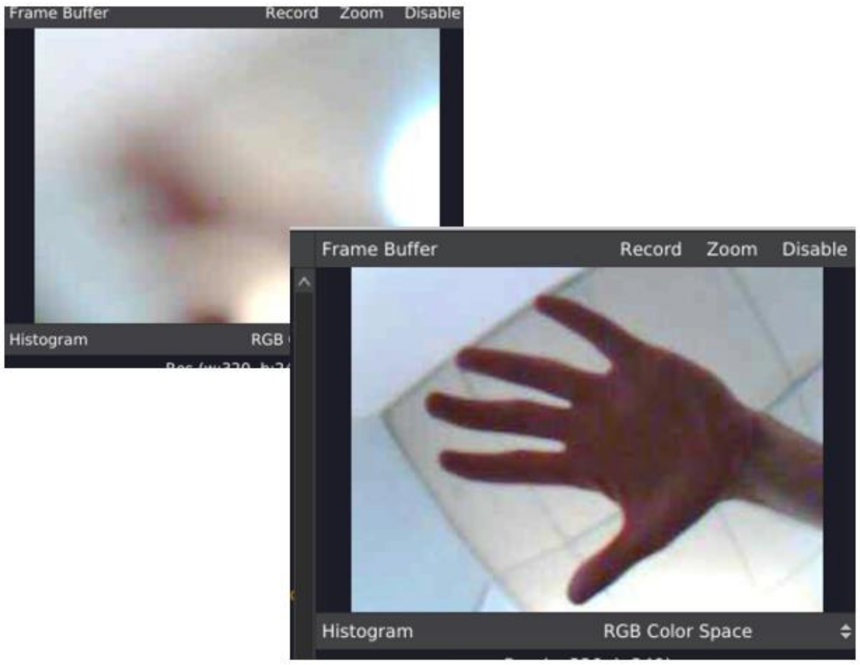
Select the action you want to perform:

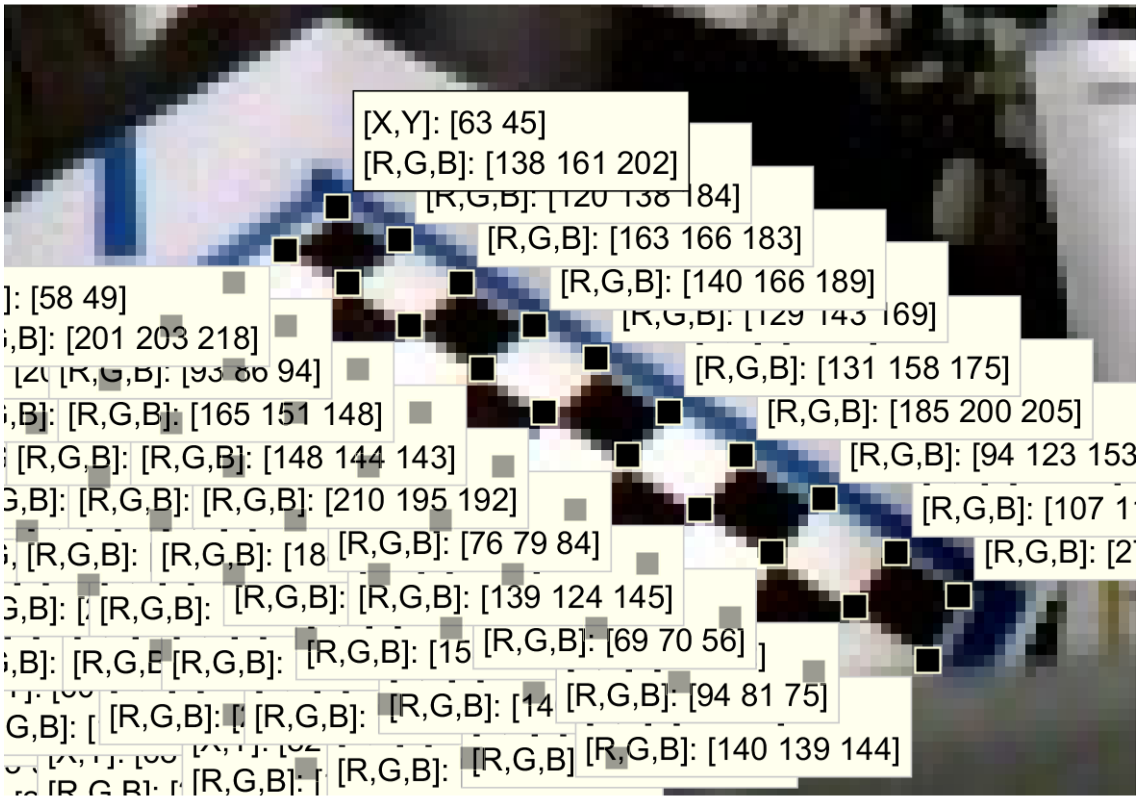
Open in File Manager

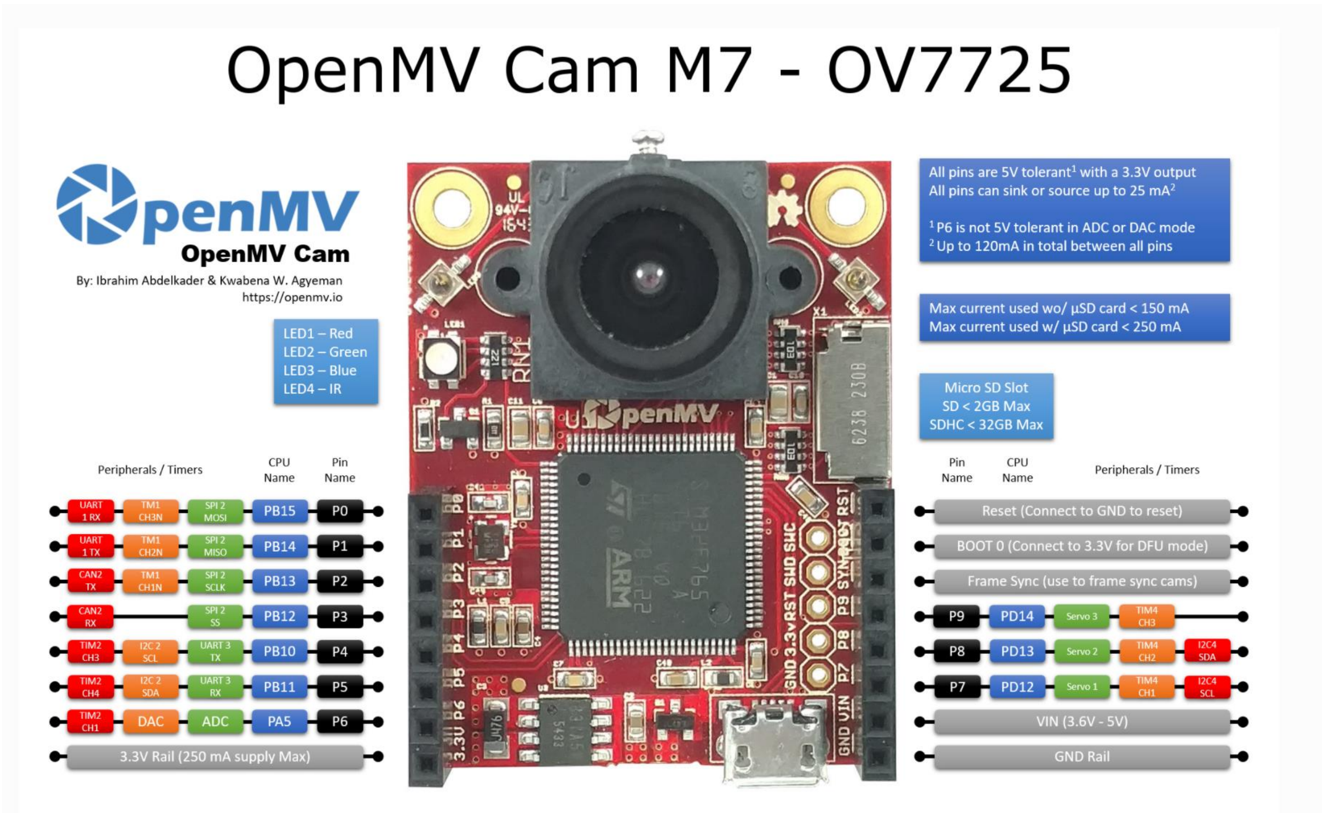


```
# Hello World Example
#
# Welcome to the OpenMV IDE! Click on the green run arrow button below to run the

import sensor, image, time

sensor.reset()                      # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565 (or GRAYSCALE)
sensor.set_framesize(sensor.QVGA)   # Set frame size to QVGA (320x240)
sensor.skip_frames(time = 2000)     # Wait for settings take effect.
clock = time.clock()                # Create a clock object to track the FPS.

while(True):
    clock.tick()                    # Update the FPS clock.
    img = sensor.snapshot()         # Take a
    print(clock.fps())              # Note: OpenM
                                    # to the IDE. The FP
```

Erase the internal file system?




[X,Y]: [63 45]
[R,G,B]: [138 161 202]
[R,G,B]: [120 138 184]
[R,G,B]: [163 166 183]
[R,G,B]: [140 166 189]
[R,G,B]: [129 143 169]
[R,G,B]: [131 158 175]
[R,G,B]: [185 200 205]
[R,G,B]: [94 123 153
[R,G,B]: [107 1
[R,G,B]: [2
]: [58 49]
,B]: [201 203 218]
[R,G,B]: [93 86 94]
[R,G,B]: [165 151 148]
[R,G,B]: [148 144 143]
[R,G,B]: [210 195 192]
[R,G,B]: [76 79 84]
[R,G,B]: [139 124 145]
[R,G,B]: [69 70 56]
[R,G,B]: [94 81 75]
[R,G,B]: [140 139 144]


OpenMV Cam M7 - OV7725
OpenMV
OpenMV Cam
By: Ibrahim Abdelkader & Kwabena W. Agyeman
https://openmv.io
LED1 – Red
LED2 – Green
LED3 – Blue
LED4 – IR
All pins are 5V tolerant¹ with a 3.3V output
All pins can sink or source up to 25 mA²
¹ P6 is not 5V tolerant in ADC or DAC mode
² Up to 120mA in total between all pins
Max current used wo/ µSD card < 150 mA
Max current used w/ µSD card < 250 mA
Micro SD Slot
SD < 2GB Max
SDHC < 32GB Max
Peripherals / Timers
CPU Name
Pin Name
UART 1 RX, TM1 CH3N, SPI 2 MOSI, PB15, P0
UART 1 TX, TM1 CH2N, SPI 2 MISO, PB14, P1
CAN2 TX, TM1 CH1N, SPI 2 SCLK, PB13, P2
CAN2 RX, SPI 2 SS, PB12, P3
TIM2 CH3, I2C 2 SCL, UART 3 TX, PB10, P4
TIM2 CH4, I2C 2 SDA, UART 3 RX, PB11, P5
TIM2 CH1, DAC, ADC, PA5, P6
3.3V Rail (250 mA supply Max)
Reset (Connect to GND to reset)
BOOT 0 (Connect to 3.3V for DFU mode)
Frame Sync (use to frame sync cams)
P9, PD14, Servo 3, TIM4 CH3
P8, PD13, Servo 2, TIM4 CH2, I2C4 SDA
P7, PD12, Servo 1, TIM4 CH1, I2C4 SCL
VIN (3.6V - 5V)
GND Rail

Specification

- Red: +5V.
- Green: Tx.
- White : Rx.
- Black: GND.