Computer Vision Course

Comp 558

Assignment Number 2

By Asa Borzabadi Farahani

Contents

Question 1 - Optical Flow

According to the slides, when we want to do image registration using the frames of a movie, we can use the **Lucus-Kanade optical flow** algorithm, which is described as below, hx and hy can build our flow vector.

Lucas-Kanade optical flow

$$\begin{bmatrix} \sum_{x,y} W(\cdot)(\frac{\partial I}{\partial x})^2 & \sum_{x,y} W(\cdot)(\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ \sum_{x,y} W(\cdot)(\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & \sum_{x,y} W(\cdot)(\frac{\partial I}{\partial y})^2 \end{bmatrix} \begin{bmatrix} h_x \\ h_y \end{bmatrix} = - \begin{bmatrix} \sum_{x,y} W(\cdot)(\frac{\partial I}{\partial t})\frac{\partial I}{\partial x} \\ \sum_{x,y} W(\cdot)(\frac{\partial I}{\partial t})\frac{\partial I}{\partial y} \end{bmatrix}$$

$$\frac{\partial I(x,y,t)}{\partial t} \approx I(x,y,t+1) - I(x,y,t)$$

In this regards, the very first step is to build the so called Structure Tensor matrix for each location (x,y) in the source image. The Structure Tensor matrix will be defined for each (x,y) in the source image as below:

$$(\nabla I) \cdot (\nabla I)^T = \begin{bmatrix} (\frac{\partial I}{\partial x})^2 & (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & (\frac{\partial I}{\partial y})^2 \end{bmatrix}$$

In the next step, we can use a 2 Dimensional convolution with to obtain the structure called the second moment matrix. The second moment matrix of each (x,y) location is a matrix just as what you can see below and it is a sum (or weighted sum) of the Structure Tensor matrices in a neighborhood, the size of the neighborhood is a parameter that we need to decide about in the next steps.

$$\begin{bmatrix} \sum(\frac{\partial I}{\partial x})^2 & \sum(\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ \sum(\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & \sum(\frac{\partial I}{\partial y})^2 \end{bmatrix}$$

Off course, it is important to emphasize that for the algorithm need the spatial and temporal derivatives to be valid, in this regard, we need to smooth the data we have both spatially and temporally, we do so by using gaussian filters, and the standard deviation of the gaussians used for smoothing are also considered as a parameter that we need to think about.

***Note:** I supposed that the weight matrix ($W(.)$) is 1 in all locations, however, in practice it is usually better to give more weight to the pixels that are closer to the central pixel

The abovementioned parameters can be manipulated to find optimal results, in the generated code, you can find the related variables in the code called “test_flow.m”:

```
smoothing_time → this is the parameter related to temporal
smoothing;
Neighborhoodsize → this is the parameter related to window size
when calculating the second moment matrix.
Sigma_spatial → this is the parameter related to spatial
smoothing
```

`Neighborhoodsize` - It is very important to select a suitable neighborhood. If the neighborhood size is too large or too small, the detection accuracy will be seriously affected. A neighborhood that is too small will not be bias to salient motion, while a neighborhood with too large a scale will introduce more noise.

`Sigma_spatial` - It is the standard deviation σ for the filter, The standard deviation indicates the degree of smoothness of gaussian filter applied. It is essential that we do the smoothing; since, as mentioned above, we are going to calculate the derivatives, for these derivatives to be meaningful, we need to have smoothed images. I am using the function called “imgaussfilt” to achieve spatial smoothing. The sigma value really depends on your application. If you want to see big object’s movements you can use big sigma.

`smoothing_time` – It is smoothing in terms of time. Actually, the frames that we have will be concatenated, and for each pixel we will do the smoothing across frames. This step seems essential, especially when there is a huge movement from a frame to the next frame. If you have really rapid movement, maybe using a bigger sigma could be better.

Note: The results are provided within some video clips. Note that I have added redundant frames to the video to make it change slowly that you can track the velocity vectors easily.

Note: I tried to find the optimal situation by trying different set of parameters, the videos are located at a folder named “optimal results”.

Note: Off course the parameters that I used to generate the videos could be different for different applications.

Figure 1 The basketball dataset is used here. – Frame number 13 and the estimated velocity vectors are shown, 4 dark blue vectors show the actual movements in the video. There is a high consistency between the actual movement and the estimated velocity vectors. (The information about the parameters is Witten in white)

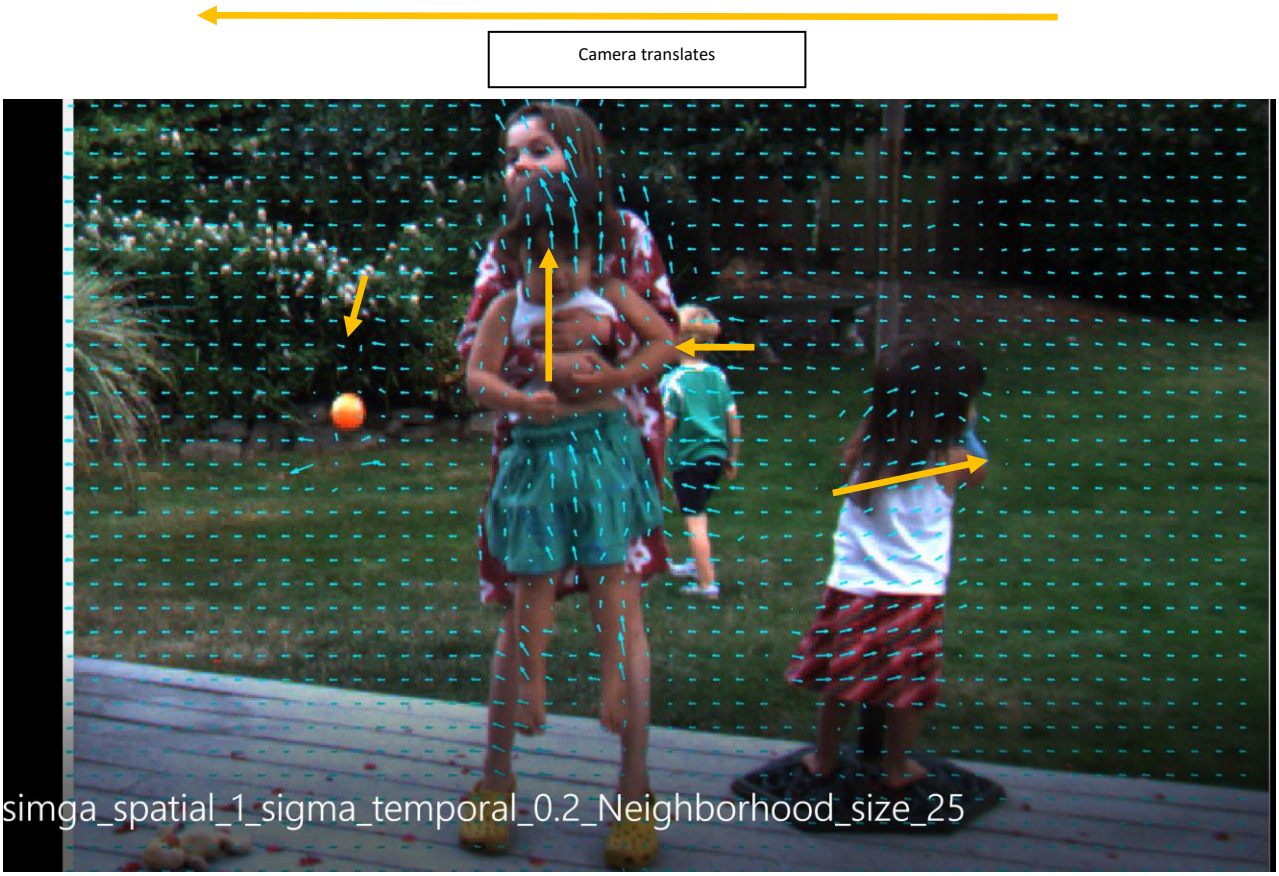


Figure 2 The backyard dataset is used here. – Frame number 13 and the estimated velocity vectors are shown, 5 yellow vectors show the actual movements in the video. There is a high consistency between the actual movement and the estimated velocity vectors. (The information about the parameters is Witten in white)

- Note that in the background, there are velocity vectors pointing to the left side of the images. The reason is that it seems like that the camera has a translation in that direction.
- The movement of the ball is not observed as clearly as movement of other objects.

Figure 3 – The Grove dataset is used here. – Frame number 13 and the estimated velocity vectors are shown, the yellow vector shows the actual movements in the video. There is a high consistency between the actual movement and the estimated velocity vectors. (The information about the parameters is Witten in white)

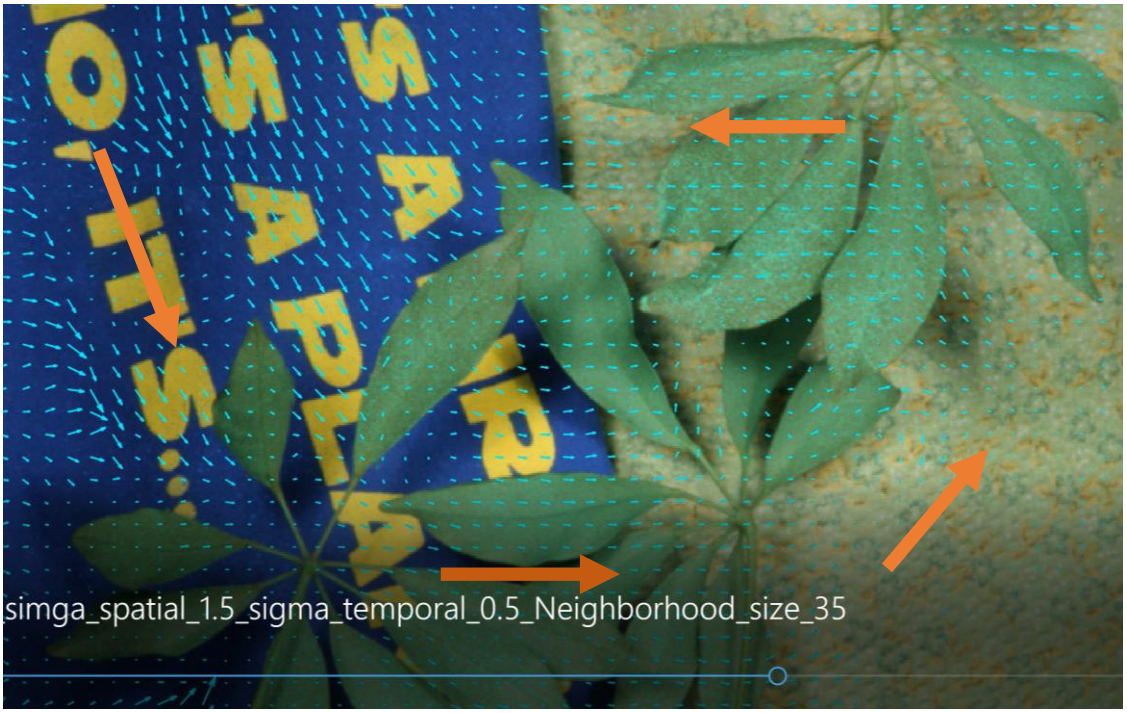


Figure 4 The Schefflera dataset is used here. – Frame number 12 and the estimated velocity vectors are shown, 4 orange vectors show the actual movements in the video. There is a high consistency between the actual movement and the estimated velocity vectors. (The information about the parameters is Witten in white)

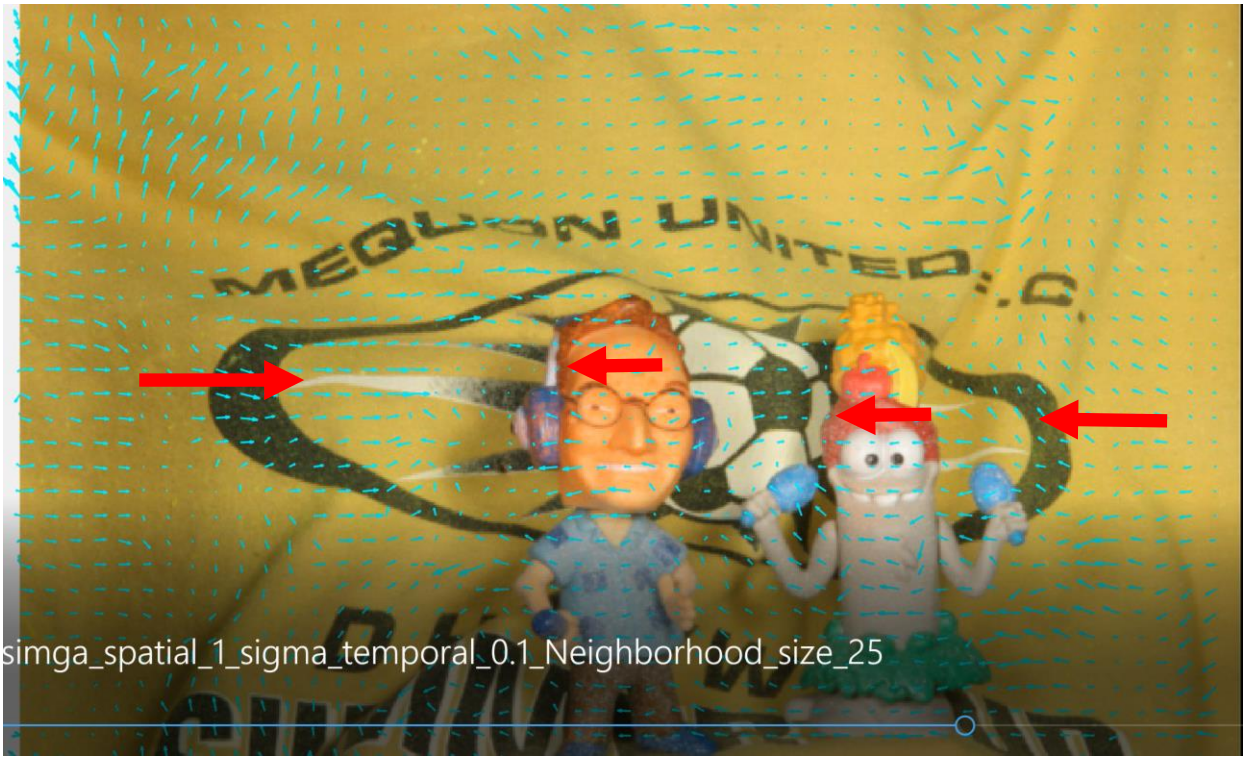


Figure 5 The Mequon dataset is used here. – Frame number 12 and the estimated velocity vectors are shown, 4 red vectors show the actual movements in the video. There is a high consistency between the actual movement and the estimated velocity vectors. (The information about the parameters is Witten in white)

Question 2 – Projection, Translation, Rotation

Part a

Roll

$$x(t) = \frac{X(t)}{Z(t)} f \qquad y(t) = \frac{Y(t)}{Z(t)} f$$

$$v_x = \frac{d}{dt} x(t) \qquad v_y = \frac{d}{dt} y(t)$$

* if "roll"

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} \cos(\Omega t) & -\sin(\Omega t) & 0 \\ \sin(\Omega t) & \cos(\Omega t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} = \begin{bmatrix} X_0 \cos(\Omega t) - Y_0 \sin(\Omega t) \\ X_0 \sin(\Omega t) + Y_0 \cos(\Omega t) \\ Z_0 \end{bmatrix}$$

$$x(t) = \frac{f}{Z_0}\left(X_0 \cos(\Omega t) - Y_0 \sin(\Omega t)\right)$$

$$\frac{dx(t)}{dt} = \frac{f}{Z_0}\left(-X_0 \Omega \sin(\Omega t) - Y_0 \Omega \cos(\Omega t)\right)$$

$$\left.\frac{dx(t)}{dt}\right|_{t=0} = \frac{f}{Z_0}\left(-X_0 \Omega \underbrace{\sin(0)}_{0} - Y_0 \Omega \cos(0)\right) = -\frac{f}{Z_0} Y_0 \Omega = -y_0 \Omega$$

$$y(t) = \frac{f}{Z_0}\left(X_0 \sin(\Omega t) + Y_0 \cos(\Omega t)\right)$$

$$\frac{dy(t)}{dt} = \frac{f}{Z_0}\left(X_0 \Omega \cos(\Omega t) + Y_0 \Omega \sin(\Omega t) \times (-1)\right)$$

$$\left.\frac{dy(t)}{dt}\right|_{t=0} = \frac{f}{Z_0} X_0 \Omega = x_0 \Omega$$

$$\begin{cases} v_x = -y_0 \Omega \\ v_y = x_0 \Omega \end{cases}$$

Pan

*if "Pan"

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} \cos(\Omega t) & 0 & \sin(\Omega t) \\ 0 & 1 & 0 \\ -\sin(\Omega t) & 0 & \cos(\Omega t) \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

$$\begin{cases} X(t) = X_0 \cos(\Omega t) + Z_0 \sin(\Omega t) \\ Y(t) = Y_0 \\ Z(t) = -X_0 \sin(\Omega t) + Z_0 \cos(\Omega t) \end{cases}$$

$$\Rightarrow x(t) = \frac{X(t)}{Z(t)} f = \frac{X_0 \cos(\Omega t) + Z_0 \sin(\Omega t)}{-X_0 \sin(\Omega t) + Z_0 \cos(\Omega t)} f$$

$$\left.\frac{dx(t)}{dt}\right|_{t=0} = \frac{-X_0 f(-X_0 \Omega) + Z_0 \Omega (Z_0) f}{Z_0^2}$$

$$= \frac{f \Omega X_0^2 + Z_0^2 \Omega f}{Z_0^2} = \Omega f \left(1 + \left(\frac{x}{f}\right)^2\right)$$

$$\Rightarrow y(t) = \frac{Y(t)}{Z(t)} f = \frac{Y_0 f}{(-X_0 \sin(\Omega t) + Z_0 \cos(\Omega t))}$$

$$\frac{dy(t)}{dt} = \frac{-Y_0 f (-X_0 \Omega \cos(\Omega t) - Z_0 \Omega \sin(\Omega t))}{(-X_0 \sin(\Omega t) + Z_0 \cos(\Omega t))^2}$$

$$\left.\frac{dy(t)}{dt}\right|_{t=0} = \frac{+Y_0 f X_0 \Omega}{Z_0^2} = f \Omega \frac{xy}{f^2} = \frac{xy}{f} \Omega$$

$$\begin{cases} v_x = \Omega f \left(1 + \left(\frac{x}{f}\right)^2\right) \\ v_y = \frac{xy}{f} \Omega \end{cases}$$

if "tilt"

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\Omega t) & -\sin(\Omega t) \\ 0 & \sin(\Omega t) & \cos(\Omega t) \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

$$X(t) = X_0$$
$$Y(t) = Y_0 \cos(\Omega t) - Z_0 \sin(\Omega t)$$
$$Z(t) = Y_0 \sin(\Omega t) + Z_0 \cos(\Omega t)$$

$$x(t) = \frac{X(t)}{Z(t)} f = f \frac{X_0}{Y_0 \sin(\Omega t) + Z_0 \cos(\Omega t)}$$

$$\frac{dx(t)}{dt} = \frac{-f X_0 \times (+Y_0 \Omega \cos(\Omega t) - Z_0 \Omega \sin(\Omega t))}{(Y_0 \sin(\Omega t) + Z_0 \cos(\Omega t))^2}$$

$$\left.\frac{dx(t)}{dt}\right|_{t=0} = \frac{-f X_0 Y_0 \Omega}{Z_0^2} = -\frac{X_0 f}{Z_0} \frac{Y_0 f}{Z_0} \frac{\Omega}{f} = -xy\frac{\Omega}{f}$$

$$y(t) = \frac{Y(t)}{Z(t)} f = f \frac{Y_0 \cos(\Omega t) - Z_0 \sin(\Omega t)}{Y_0 \sin(\Omega t) + Z_0 \cos(\Omega t)}$$

$$\left.\frac{dy(t)}{dt}\right|_{t=0} = f \frac{-Z_0 \Omega Z_0 - Y_0 \Omega Y_0}{Z_0^2} = -\Omega - \frac{Y_0^2}{Z_0^2}\Omega$$
$$= -\Omega\left(1 + \left(\frac{y}{f}\right)^2\right)$$

$$v_x = -xy\frac{\Omega}{f}$$

$$v_y = -\Omega\left(1 + \left(\frac{y}{f}\right)^2\right)$$

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\Omega t) & \sin(\Omega t) \\ 0 & -\sin(\Omega t) & \cos(\Omega t) \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

$$X(t) = X_0$$

$$Y(t) = Y_0 \cos(\Omega t) + Z_0 \sin(\Omega t)$$

$$Z(t) = -Y_0 \sin(\Omega t) + Z_0 \cos(\Omega t)$$

$$x(t) = f \frac{X(t)}{Z(t)} = f \frac{X_0}{-Y_0 \sin(\Omega t) + Z_0 \cos(\Omega t)}$$

$$\left.\frac{dx(t)}{dt}\right|_{t=0} = f \frac{-X_0(-Y_0\Omega)}{Z_0^2} = f \frac{X_0}{Z_0} \frac{Y_0}{Z_0} \Omega = f \frac{xy}{f^2} \Omega = \frac{xy}{f} \Omega$$

$$y(t) = f \frac{Y(t)}{Z(t)} = f \frac{Y_0 \cos(\Omega t) + Z_0 \sin(\Omega t)}{-Y_0 \sin(\Omega t) + Z_0 \cos(\Omega t)}$$

$$\left.\frac{dy(t)}{dt}\right|_{t=0} = \frac{(Z_0\Omega) Z_0 - Y_0(-Y_0\Omega)}{Z_0^2} = \Omega\left(1 + \frac{Y_0^2}{Z_0^2}\right) = \Omega\left(1 + \left(\frac{y}{f}\right)^2\right)$$

$$v_x = \frac{xy}{f} \Omega$$

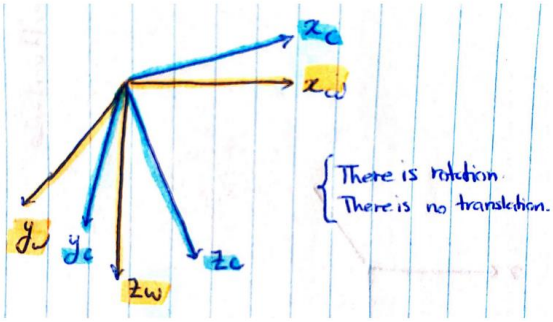
$$v_y = \Omega\left(1 + \left(\frac{y}{f}\right)^2\right)$$

Part b

There is no translation vector; as we look at the extrinsic matrix which is a 3*4 matrix, we can see that the fourth column is made of zeros.

$$Extransic\ matrix = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \end{bmatrix}$$

This means that the origin of the camera's coordinate and the origin of the world's coordinate are in located the same place, and we have a structure like this:



Now that there is no translation component, we can truncate the extrinsic matrix and make it 3*3. We can omit the fourth column which is related to translation. So, we no longer need to work with homogenous coordinate and the formulation of the problem would be simplie.

The general formulation is:

$$\begin{bmatrix} w\ x_{pixel} \\ w\ y_{pixel} \\ w \end{bmatrix} = \begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \end{bmatrix} \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} w\ x_{pixel} \\ w\ y_{pixel} \\ w \end{bmatrix} = \begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix}$$

$$\begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \end{bmatrix} \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix}$$

Where $\begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$ is the intrinsic matrix and $\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \end{bmatrix}$ form the extrinsic matrix.

However, in this case, as there is no translation, we would have a simpler approach:

$$\begin{bmatrix} w\,x_{pixel} \\ w\,y_{pixel} \\ w \end{bmatrix} = \begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \end{bmatrix}$$

$$\begin{bmatrix} w\,x_{pixel} \\ w\,y_{pixel} \\ w \end{bmatrix} = \begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix}$$

$$\begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \end{bmatrix}$$

In this question, we know about pixel coordinate, and we are given the extrinsic and intrinsic matrix as well as the depth image. Depth image tells us about the z coordinate of the camera. Having depth image we can find the w value and hence we can solve for the world coordinate.

$$\begin{bmatrix} w\,x_{pixel} \\ w\,y_{pixel} \\ w \end{bmatrix} = \begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix} = \begin{bmatrix} fm_x x_{camera} + p_x z_{camera} \\ fm_y\, y_{camera} + p_y z_{camera} \\ z_{camera} \end{bmatrix}$$

$$so \rightarrow w = z_{camera}$$

$$\begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix} = inv\left(\begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} w\,x_{pixel} \\ w\,y_{pixel} \\ w \end{bmatrix}$$

$\rightarrow$ *the right side of the equation is known so we can know about the camera's* $x, y,$ *and* z

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \end{bmatrix} = inv\left(\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \right) \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix}$$

$\rightarrow$ *knowing the camera's* $x, y,$ *and* z *and the etransic matrix, we can find the world coordinate*

So, I implemented a code that just represents the abovementioned explanations:

```
for xpixel = 1:1:size(rgbImage,1)
    for ypixel = 1:1:size(rgbImage,2)

        % find world equation
        pixelCoordinate = depthImage(xpixel,ypixel)*double([xpixel;ypixel;1]);
        CameraCoordinate = inv(Ic)*pixelCoordinate;
        WorldCoordinate(xpixel,ypixel,:) = inv(Ec(:,1:3))*CameraCoordinate;

        % define world coordinate as X - Y - Z
        X(xpixel,ypixel) = WorldCoordinate(xpixel,ypixel,1);
        Y(xpixel,ypixel) = WorldCoordinate(xpixel,ypixel,2);
        Z(xpixel,ypixel) = WorldCoordinate(xpixel,ypixel,3);

    end
end
```

Note: the MATLAB code is which is used to recover the world coordinate is "`world_coordinate.m`".

Note: To plot the 3D coordinates I have used a MATLAB function called: "`pcshow`".

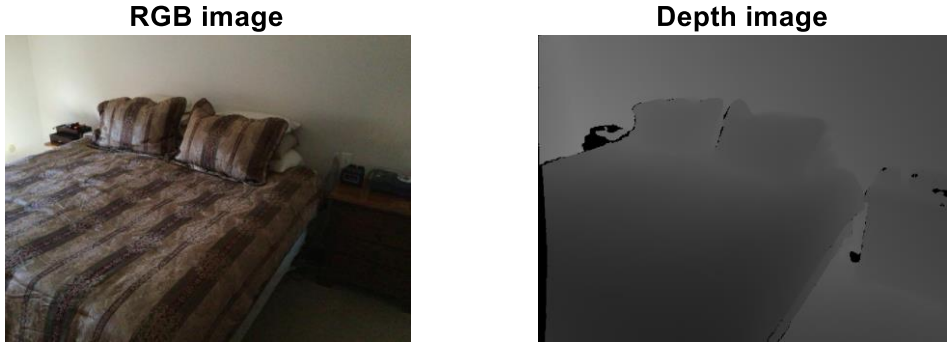


Figure 6 The first image

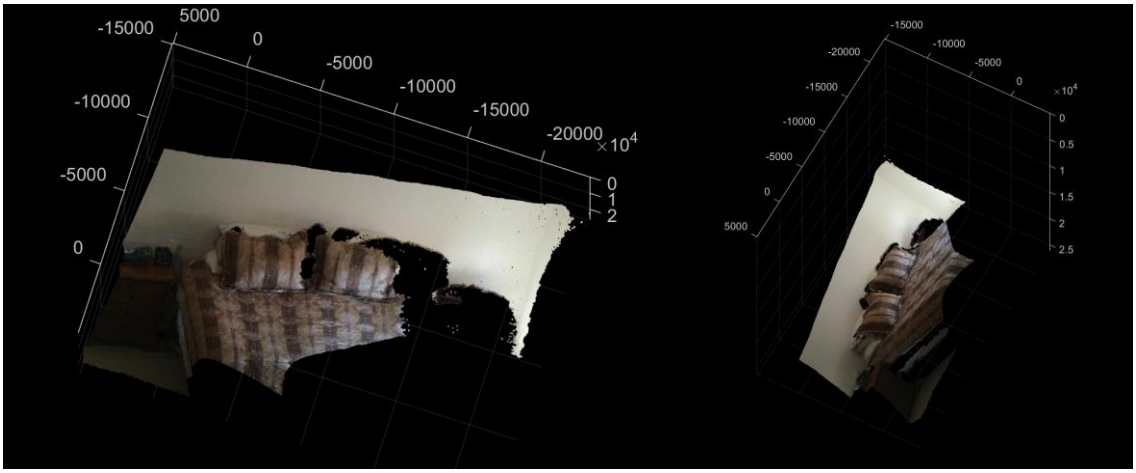


Figure 7 The reconstructed world coordinate of the first dataset

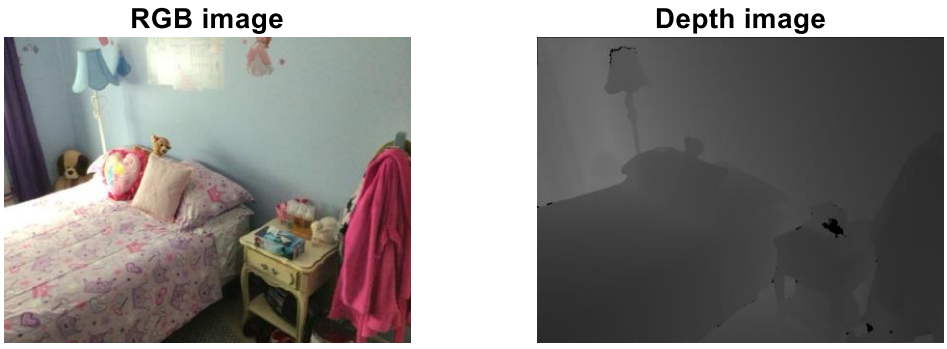


Figure 8 The second image

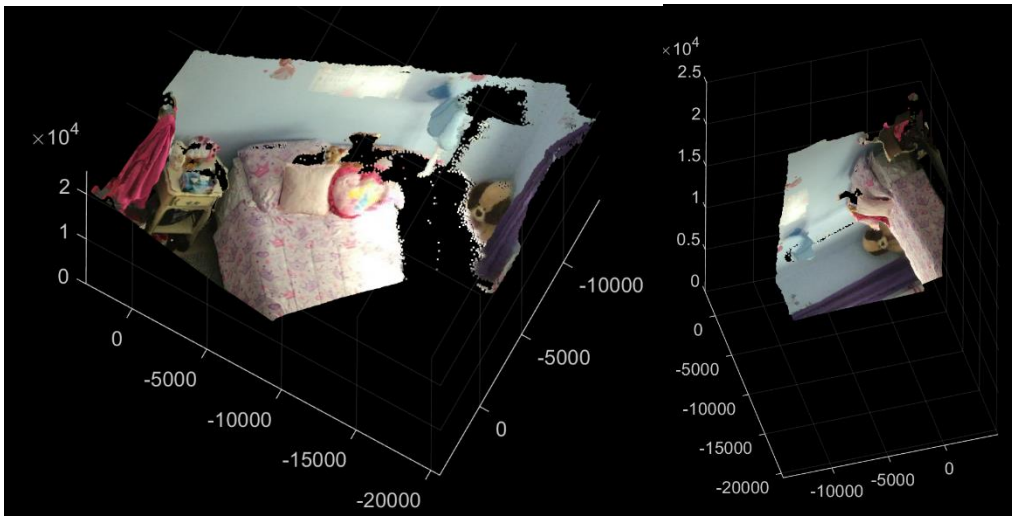


Figure 9 The reconstructed world coordinate of the second dataset

Figure 10 The third image

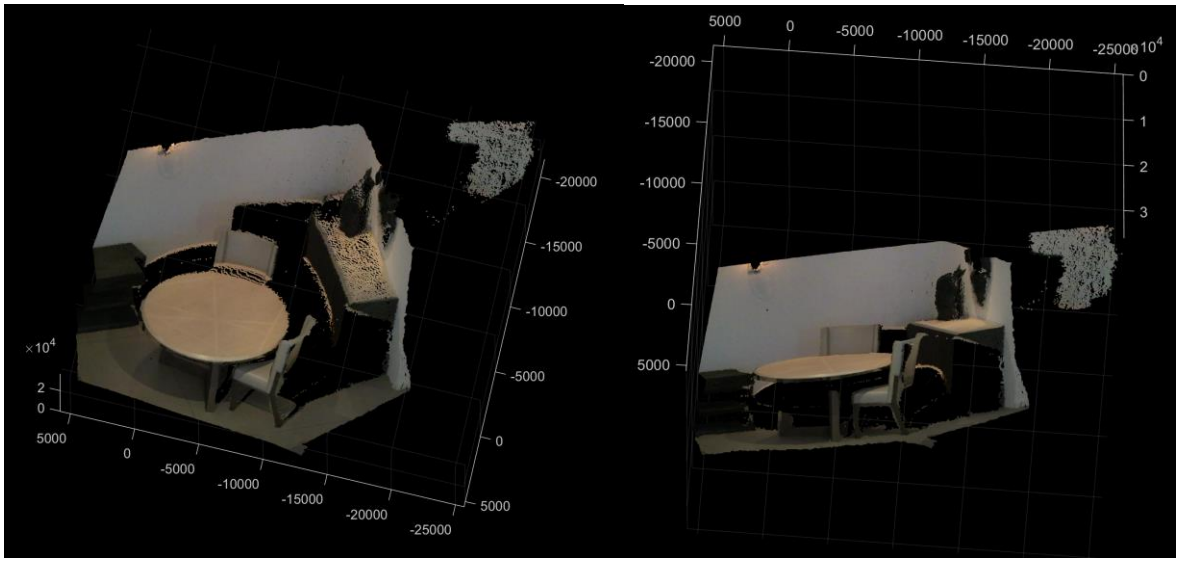


Figure 11 The reconstructed world coordinate of the third dataset

Part c

- My **assumption** for Part c and Part d: I want to move camera from its initial position. All rotations and translations are with respect to this initial position.

As discussed in part a, each rotation can be expressed by a matrix. We can create rotated versions of any 3D array by multiplying the array with the rotation matrix. The rotation matrix about x, y, or z axes is defined as follows:

a rotation about the Y -axis:

```
Xrotationmatrix = [1, 0,              0                ;...
                   0, cosd(omegax*t), -sind(omegax*t), ;...
                   0, sind(omegax*t), cosd(omegax*t)   ;];
```

a rotation about the X-axis:

```
Yrotationmatrix = [cosd(omegay*t),  0, sind(omegay*t)  ;...
                   0,               1,  0              ;...
                   -sind(omegay*t), 0, cosd(omegay*t)  ;];
```

a rotation about the Z-axis:

```
Zrotationmatrix = [cosd(omegaz*t), -sind(omegaz*t), 0  ;...
                   sind(omegaz*t),  cosd(omegaz*t), 0  ;...
                   0,               0,              1  ;];
```

Other rotation matrices can be obtained from these three by doing matrix multiplication:

```
General_rotation_matrix = Xrotationmatrix*Yrotationmatrix*Zrotationmatrix;
```

Now, if you want to do translation at the same time, it is a good idea to use homogenous coordinate; in this case, you can get the translated-rotated version of the array simply by a using a matrix multiplication.

```
TranslationVector = [Tx;Ty;Tz];

RT_manual = horzcat(General_rotation_matrix, TranslationVector);
```

Here, I have included a sample version of such a 3*4 matric, it you multiply this matrix by the homogenous coordinate of your choice, you can get the rotated and translated version of the array.

```
RT_manual =

    0.9924   -0.0868    0.0872    2.0000
    0.0944    0.9917   -0.0868   20.0000
   -0.0789    0.0944    0.9924   70.0000
```

Now, that we have this translation and rotation matrix, I just assume that I need to multiple this new RT_manual matrix with the extrinsic matrix that we are given. This is done because my assumption is that we want to translate/rotate camera from its initial position. As we have such an assumption, when we are trying to find the camera's coordinate, we do such a process:

```
    for i = 1:1: size(rgbImage,1)
        for j = 1:1: size(rgbImage,2)
            Coordinate_camera(i,j,1:3) = ((Ec(1:3,1:3))*RT_manual)*([X(i,j);Y(i,j);Z(i,j);1]);
        end
    end
```

To get the pixel coordinate it is enough to multiply the above-mentioned matrix in the intrinsic matrix of the camera.

```
for i = 1:1: size(rgbImage,1)
    for j = 1:1: size(rgbImage,2)
         PixelImg(i,j,1:3) = ((Ic*Ec(1:3,1:3))*RT_manual)*([X(i,j);Y(i,j);Z(i,j);1]);
     end
end
```

- The results are represented using some video.
- There are two versions: 1 – I have shown how the camera coordinate changes and also, I have shown how the corresponding pixel coordinate changes.
- These videos are located at: *..\Q2\Results_Part_3\Rotation* and *..\Q2\Results_Part_3\Translation*
- The names of videos say what the content is.
- Some screenshots are provided here to help elucidate the results:

Rotation – X

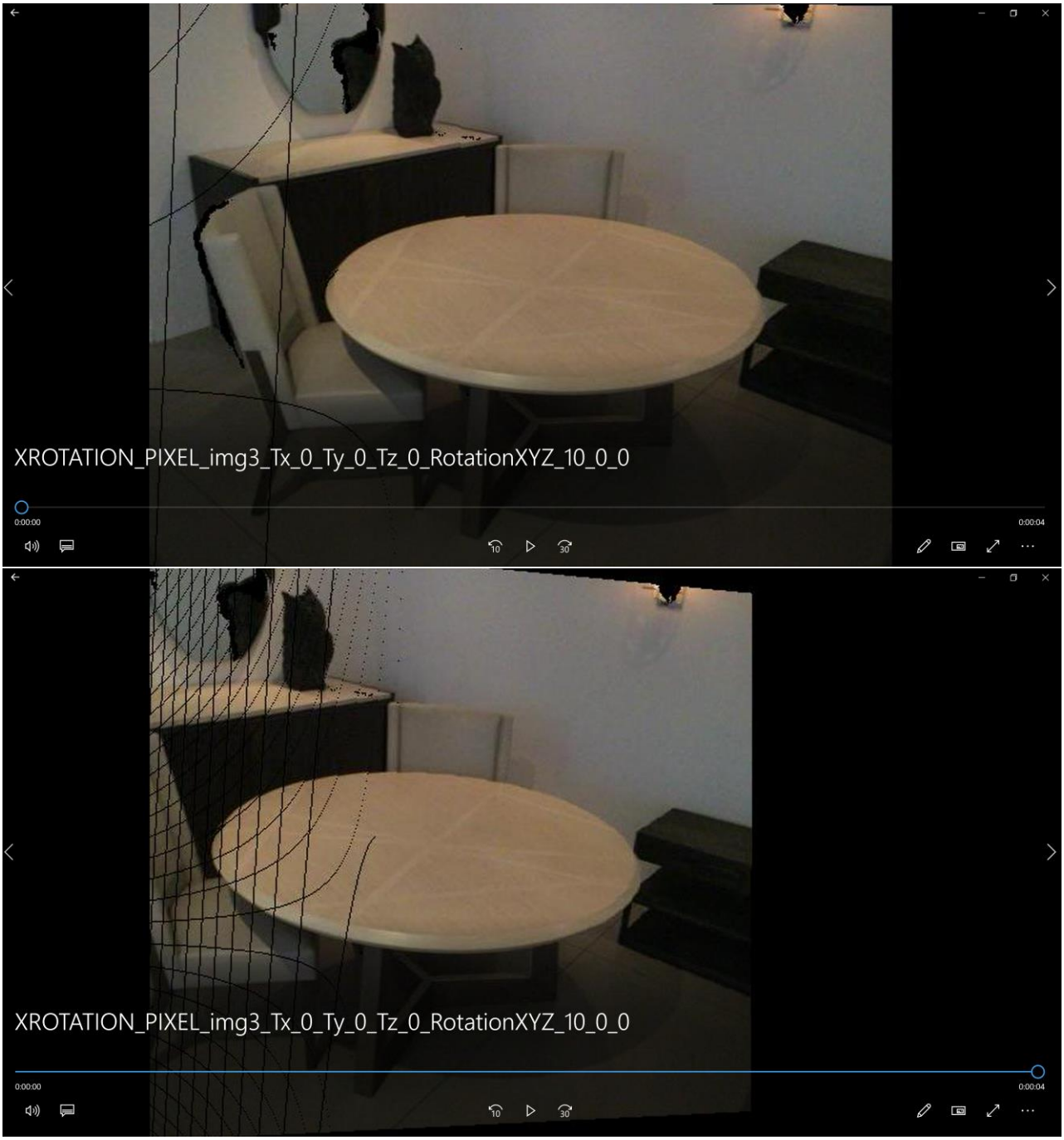


Figure 12 Pixel coordinate - The upper image is the result after the camera has rotated 1 degree with respect to its initial position. The lower image is the result after the camera has rotated 10 degrees with respect to its initial position.

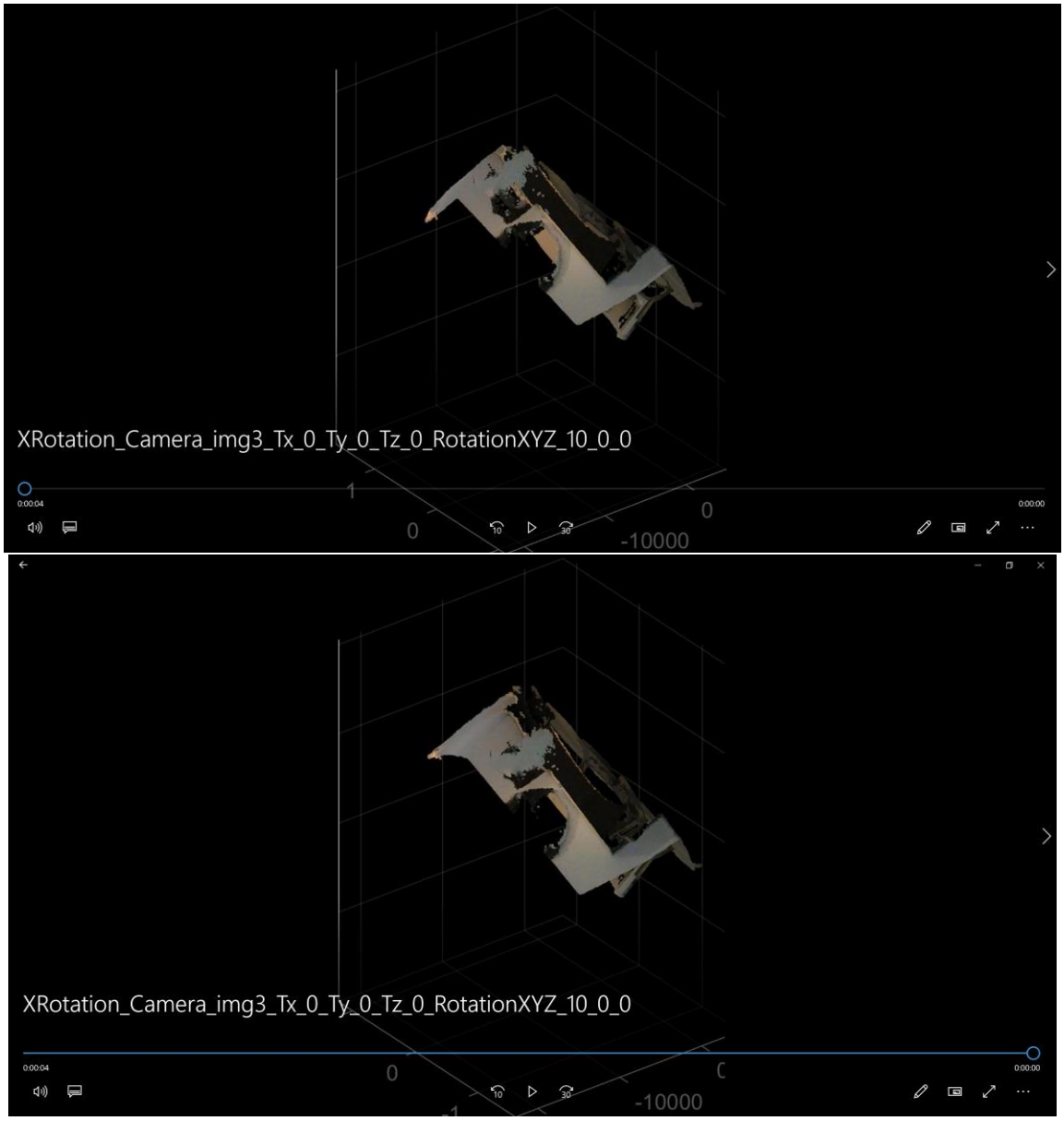


Figure 13 Camera coordinate - The upper image is the result after the camera has rotated 1 degree with respect to its initial position. The lower image is the result after the camera has rotated 10 degrees with respect to its initial position.

- In the abovementioned image you may not sense the movement, please refer to the original videos to get the sense.

Rotation – Y

Figure 14 Pixel coordinate - The upper image is the result after the camera has rotated 1 degree with respect to its initial position. The lower image is the result after the camera has rotated 10 degrees with respect to its initial position.

Figure 15 Camera coordinate - The upper image is the result after the camera has rotated 1 degree with respect to its initial position. The lower image is the result after the camera has rotated 10 degrees with respect to its initial position.

Rotation - Z

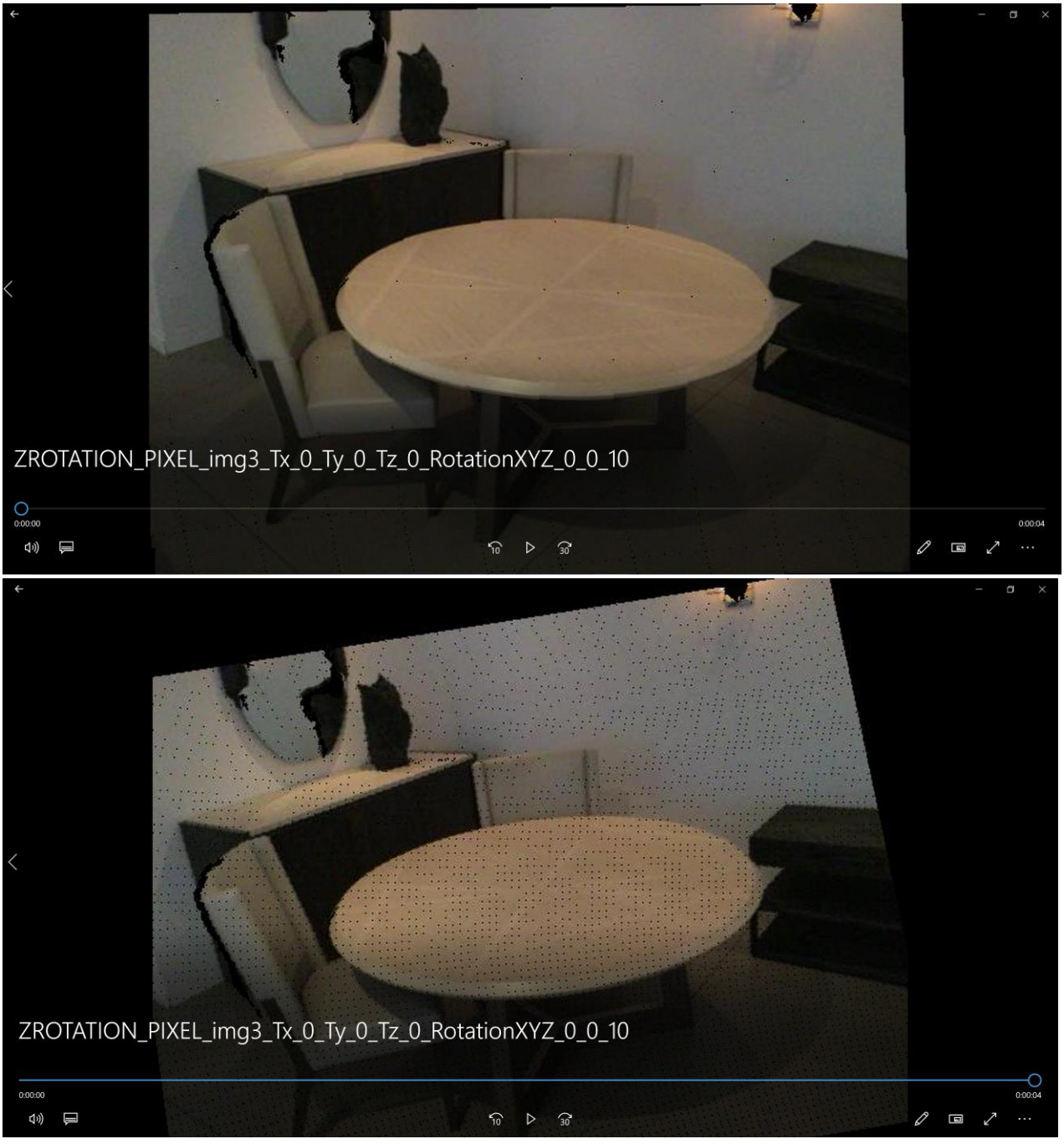


Figure 16 Pixel coordinate - The upper image is the result after the camera has rotated 1 degree with respect to its initial position. The lower image is the result after the camera has rotated 10 degrees with respect to its initial position.

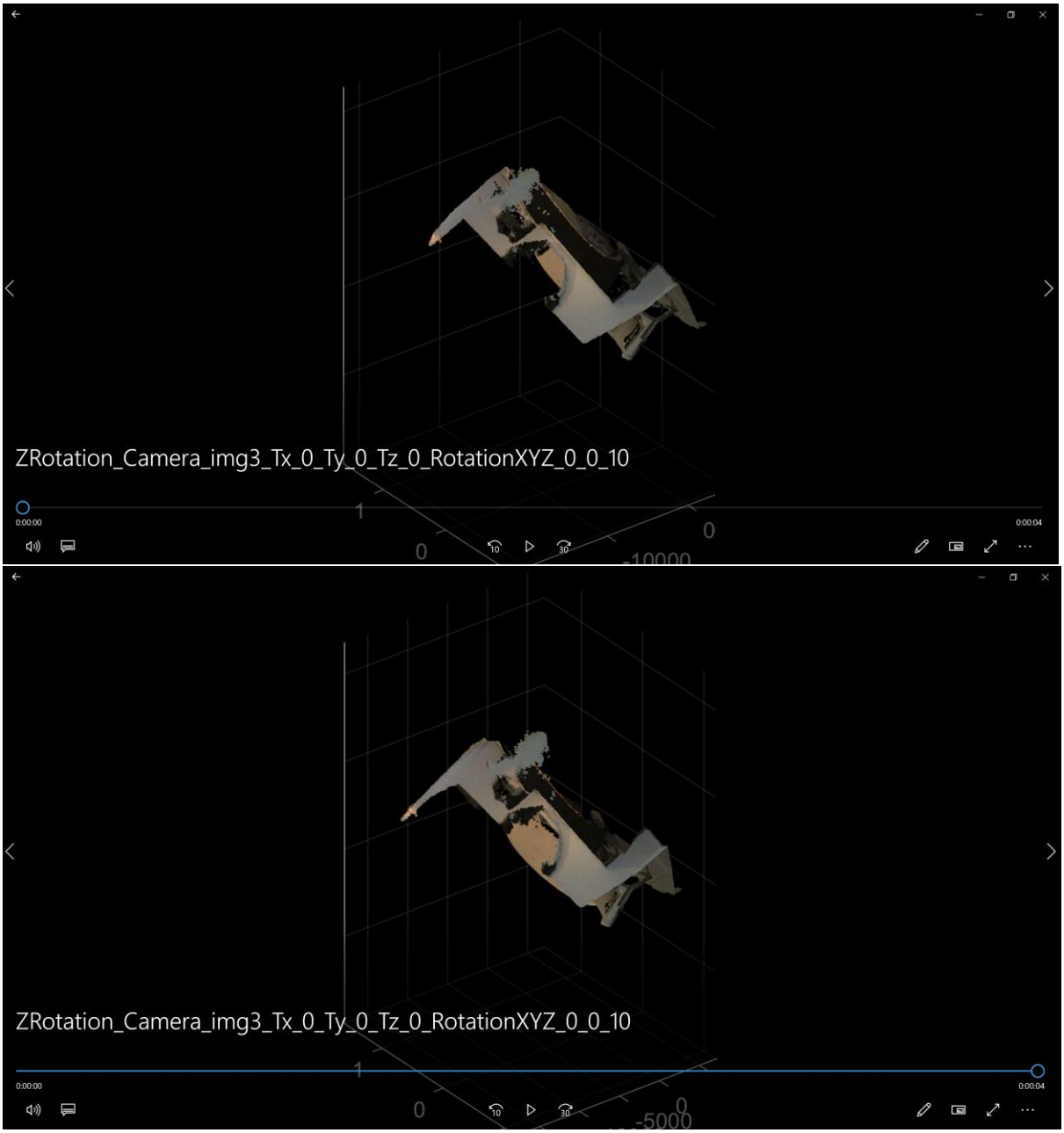


Figure 17 Camera coordinate - The upper image is the result after the camera has rotated 1 degree with respect to its initial position. The lower image is the result after the camera has rotated 10 degrees with respect to its initial position.

Translation - X

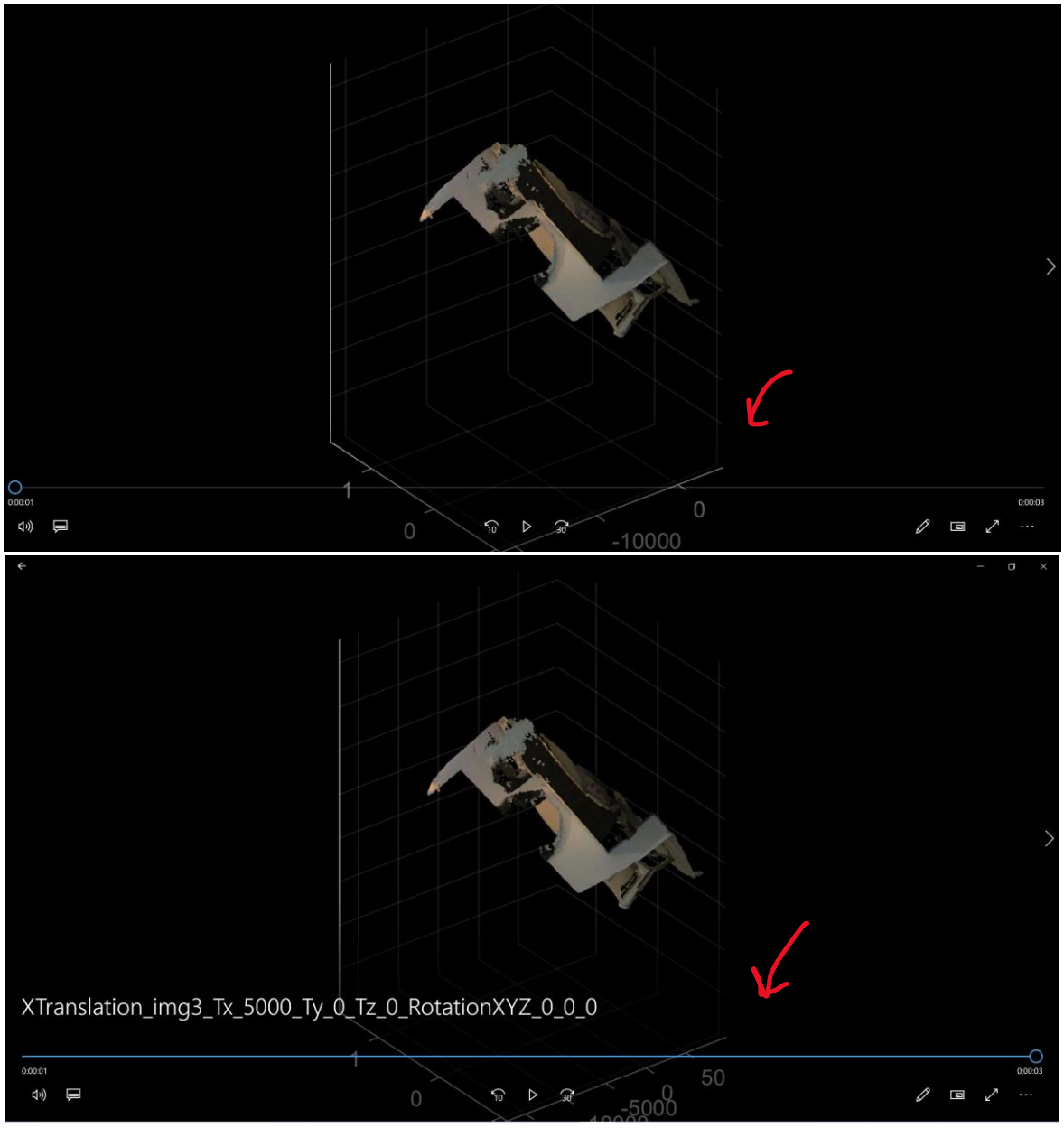


Figure 18 The camera coordinate with respect to the initial position of the camera is shown in the upper field and the camera coordinate when there was a translation of 5000 in X direction is shown below.

Figure 19 This is what we observe in the pixel coordinate when we translate the camera from its original position in X direction (for 5000).

Translation - Y

Note: To avoid redundancy, I have not included here the translation versions of the camera coordinate any more. You may refer to the videos if needed.

Figure 20 This is what we observe in the pixel coordinate when we translate the camera from its original position in Y direction (for -2000).

Translation - Z

Figure 21 This is what we observe in the pixel coordinate when we translate the camera from its original position in Z direction (for 5000).

Part d

I have provided 6 videos for each set of images. I have created 30 frames for each of the movies. In each frame the translation value and the rotation degree has changed.

The detail of each movie is provided here:

YX → Rotation in Y – Translation in X

- ➔ For frame number = i (i = 1:1:30)
  - omega_y = i*1
  - Tx = -200*t

ZY → Rotation in Z – Translation in Y

- ➔ For frame number = i (i = 1:1:30)
  - omega_z = i*1
  - Ty = 500*t

XZ → Rotation in X – Translation in Z

- ➔ For frame number = i (i = 1:1:30)
  - omega_x = i*1
  - Tz = 500*t

After each iteration, each image has been saved and finally all the images are written to a video. The sequence of projected images looks like a 3D scene, viewed from different virtual camera viewpoints.

I have included here some sample pictures, but you can observe the provided videos for more clarification.

The videos are located at:" Results_image_number_1", "Results_image_number_2", and "Results_image_number_3". In each of these folders there are 6 videos (3 depth videos and 3 RGB videos.)

Note: Mathematical concepts are the same as what I explained in the previous sections.

Figure 22 YX: Rotation in Y – Translation in X - dataset number 1

Figure 23 YX: Rotation in Y – Translation in X - dataset number 3

Figure 24 ZY - Rotation in Z – Translation in Y dataset number 1

Figure 25 ZY - Rotation in Z – Translation in Y dataset number 3

Figure 26 XZ - Rotation in X – Translation in Z dataset number 1

Figure 27 XZ - Rotation in X – Translation in Z dataset number 3