3D RECONSTRUCTION OF A SCENE FROM 2D MULTIPLE STEREO IMAGES

By : KAVEESHA WELIWATHTHA

Task of the mini project

The task of this mini project is to identify the feature points from 2D image acquired from calibrated camera and reconstructing those features using depth analysis to obtain the 3D view of the object or image. And the final obtained 3D view should be a compressed reconstruction of the input 2D Images.

Introduction

3D imaging is otherwise widely called as stereoscopy. This technique is widely used for creation or enhancement of the 2D image by increasing the illusion of depth with the help of binocular vision. Almost all kinds of stereoscopic methods are based on 2 images that is one from the left view and the other one from the right view. These 2 images are then joined together to give an illusion of 3D view along with inclusion of the depth. As a popular research topic, image-based 3D scene modelling has attracted much attention in recent years. It has wide range of applications in engineering field. Some are,

- Automobile engineers use to function the options like lane filtering, obstacle detection etc.
- Civil engineers use to generate 3D views of their construction before start.
- It is used in developing games and entertainment like virtual reality.
- Engineers use stereoscopy to navigate robots inside the building.
- Buildings that have disappeared can be modelled using photograph or a painting.
- Valuable ancient ruins can be reconstructed virtually using their existing images.

The third dimension can usually be perceived only by human vision. The eyes visualize the depth and the brain reconstructs the third dimension with the help of various views visualized by the eyes. The researchers used this strategy for reconstructing the 3D model from various views with the help of certain parameters of disparity and calibration. The main objective of 3D modelling is to computationally understand and model the 3D scene from the captured images, and provide human-like visual system for machines. The approach begins by capturing the 2D image from camera. Captured images are usually prospectively distorted. These perspective distortions are eliminated to get true dimensions of the image.

There are some special cameras like stereoscopic dual camera, depth-range camera etc. which help in capturing the 3D model of the view directly. These cameras usually capture the RGB component of the image and its corresponding depth map. The depth map is defined as the function which helps in evaluating the depth of an object at pixel positio. Usually the intensity is considered to be the depth.

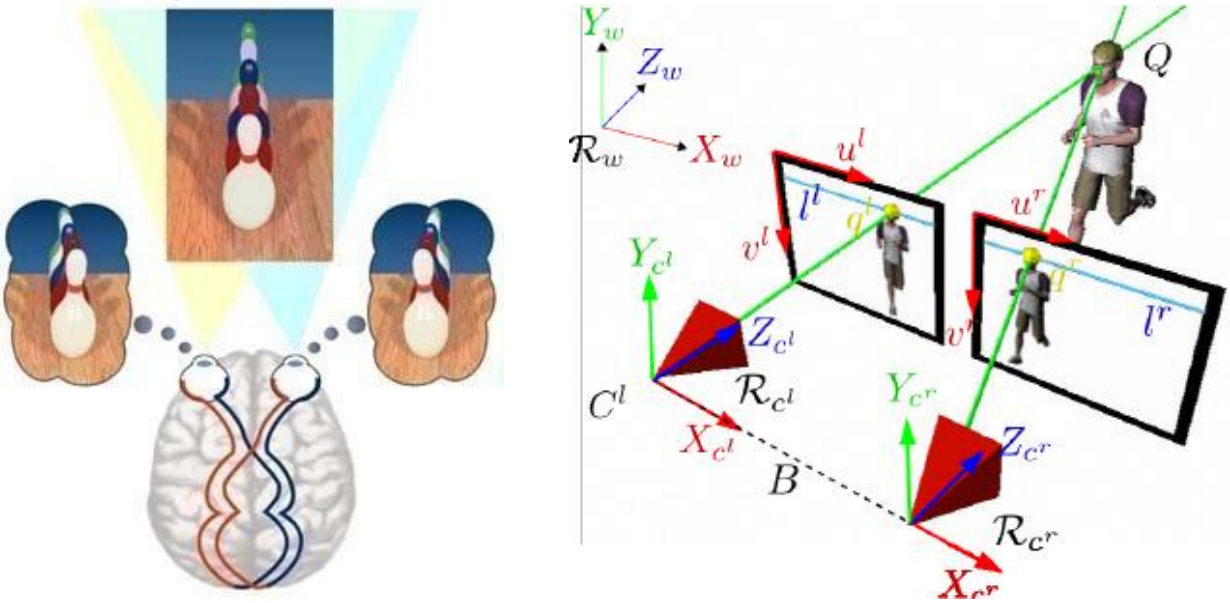
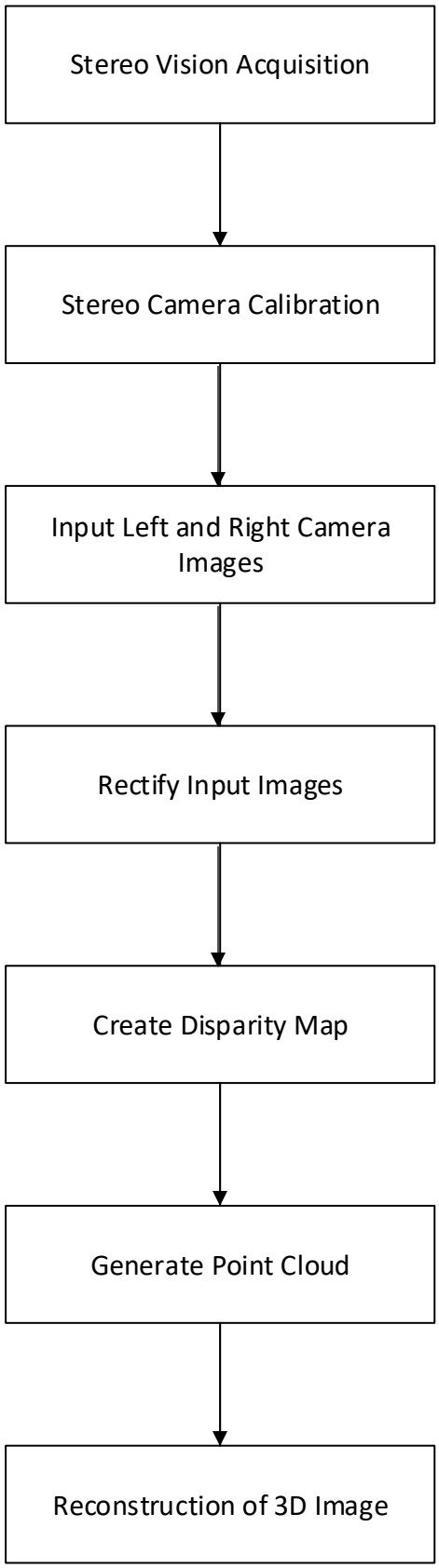


Figure 01: Human seteroscopic vision Vs. Stereoscopic vision using cameras

Methodology

Below shows the flow chart of the process we followed to complete the task.



Experimental Results

Stereo Vision Acquisition

The stereo vision acquisition is very important step in 3D reconstruction of images. To acquire stereo vision images, we can either use an integrated stereo vision camera or we can use multiple camera. The advantage of using an integrated stereo vision camera is that it offers less error than a custom multiple camera setup. An integrated stereo vision camera provides hardware triggering such that two images are taken at the exact same moment. With multiple camera setups images must be taken one at a time from each camera, if not it gives big reprojection error. The lenses in an integrated stereo vision camera do not move with respect to each other, so it eliminates the motion artifacts that occur if we use individual cameras or if an object moves between captures. This means that the camera calibration always remains accurate. A rig needs to be designed to ensure that in a multiple camera setup the cameras do not moves that a shift oscillate or shake with respect to each other. But for this project we have used multiple cameras. Which the identical two mobile cameras are used.

Figure 02: Intergrated Vs. multiple stereo vision cameras

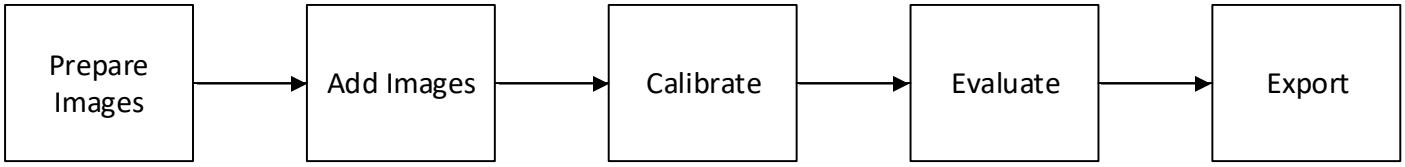
Stereo Camera Calibration

The concept of camera calibration is an important one. An image is a 2d representation of a 3d real world scene. The mapping from world to image can be described mathematically by a series of transformations. The goal of camera calibration is to estimate the parameters for these 3D reconstruction transformations. There are two types parameters to be identify. They are,

- The extrinsic parameters describe the location of the camera coordinate system within the world coordinate system (basically where the camera is placed in the world).
- The intrinsic parameters are required for the mapping from 3D camera coordinates to 2D image coordinates.

For stereo vision cameras, an accurate camera calibration becomes more essential because we can then use it to recover depth from images. A stereo system consists of two cameras. So, in addition to traditional calibration we also need to calibrate the orientation of the camera 2 relative to camera 1.

In MATLAB, there is a stereo camera calibrator app which estimates the parameters of each of the two cameras. There are 6 steps in the process of stereo camera calibration.



- First, we have to make or download a checkerboard which has even number of squares in pattern while other side has odd number of squares in the pattern. Then we have to calibrate the camera properly before tacking pictures. Without using autofocus, we have to focus on the squares of the checker board. Before capturing the images, we have to place checkerboard at a distance roughly equal to the distance from camera to object of interest in an angle of 45 degrees. Have to capture images between 10-20 or more to get a better calibration.
- Then have to add images to the MATLAB software using Stereo Calibrator App in it. After adding images, the app will compare and detect the acceptable pairs and other pairs will be rejected and removed.
- Then the app will calibrate the stereo parameters for us.
- Then we have to evaluate the results by looking at the reprojection errors which are the distances and pixels between the detected and the reprojection points. The app first detect the checkerboard points from the image pairs and then calculates the reprojection errors by projecting the checkerboard points from world coordinates defined by the checkerboard into image coordinates it then compares the reprojected points to the corresponding detected points as a general rule. Reprojection errors of less than one pixel are acceptable.
- 3D extrinsic parameters plot which provides a camera centric view of the patterns to evaluate calibration accuracy. The camera centric view is helpful if the camera was stationary when the images were captured. This view helps us examine the position of the patterns and the camera to see if they match we except. We can remove outlier images and recalibrate to improve results.
- Finally export the calibrated stereo parameters.

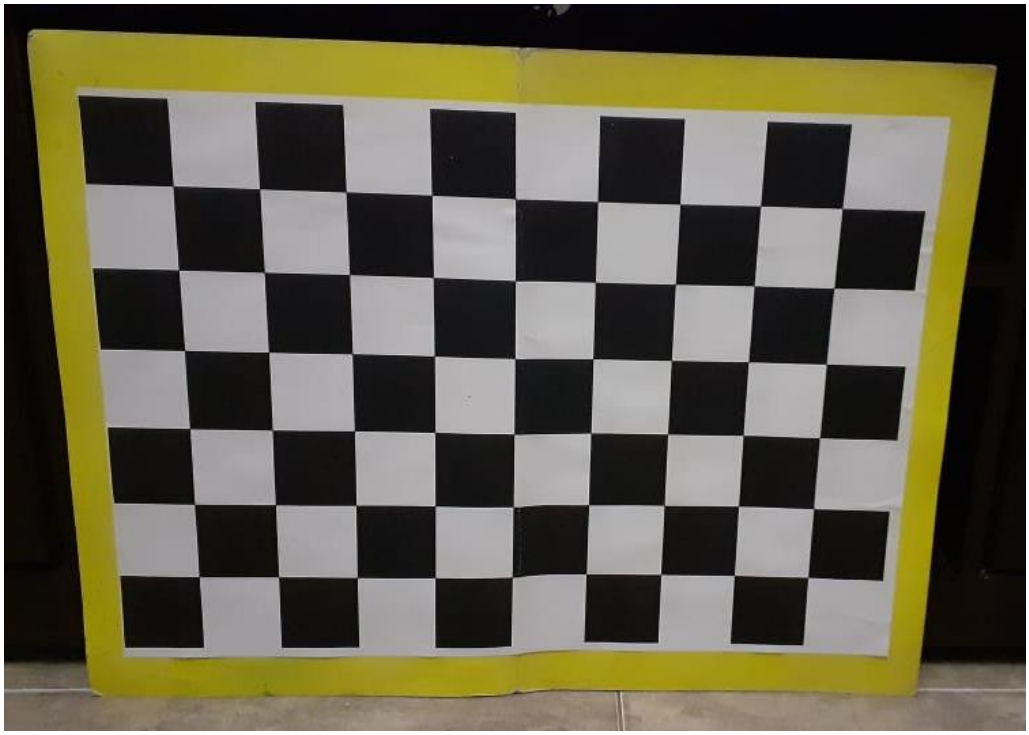


Figure 03: The checkerboard used for stereo vision calibration

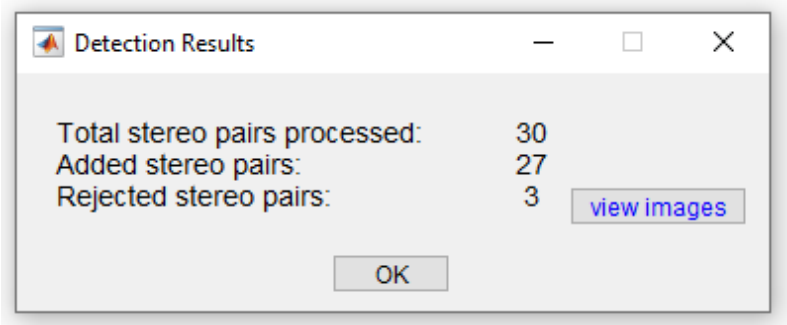


Figure 04: Detection results for added images

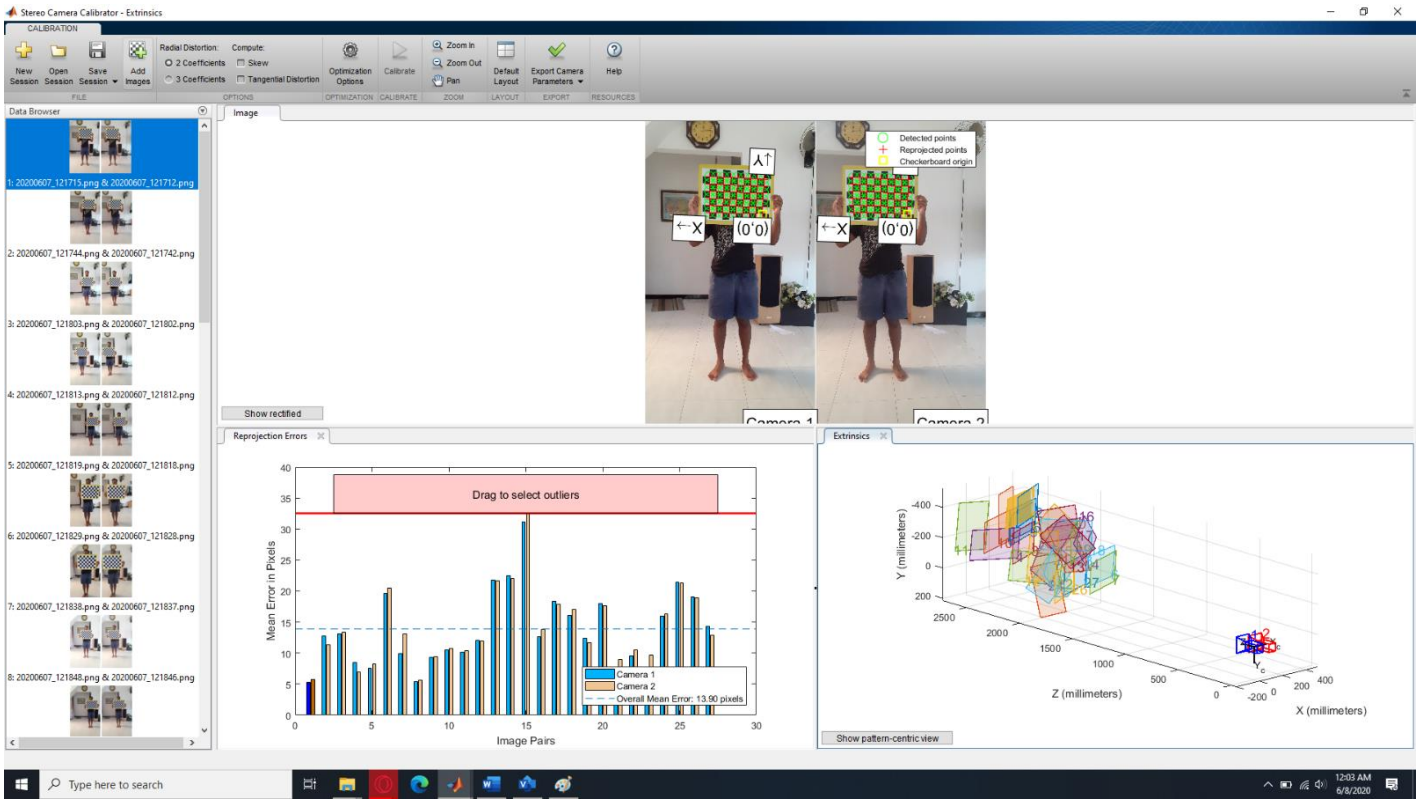


Figure 05: 1st stereo vision calibration with 13.9 overall mean error

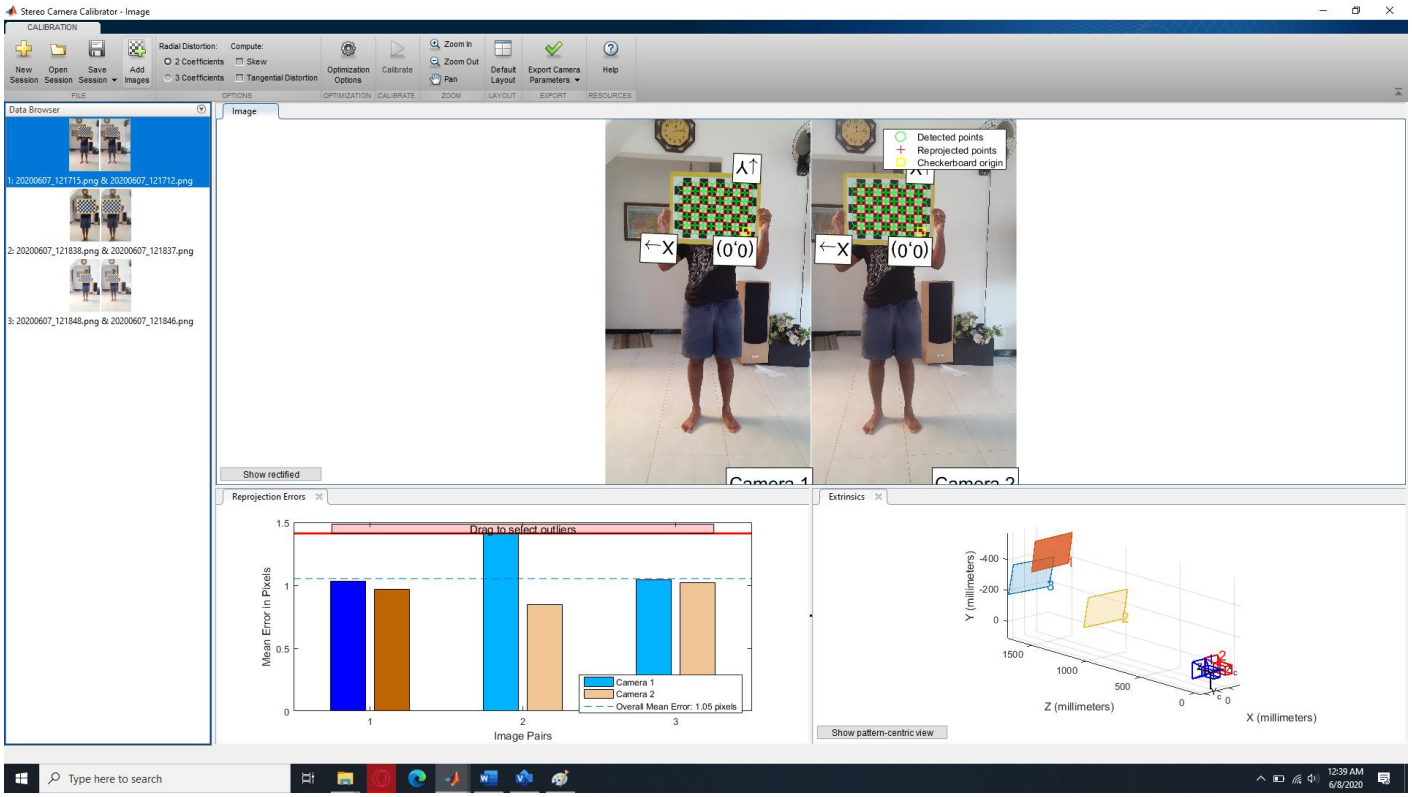


Figure 06: Final stereo vision recalibration with 1.05 overall mean error

Then we saved the stereo vision parameters and estimated errors to the MATLAB. So we got the following details.

```
stereoParams =

  stereoParameters with properties:

   Parameters of Two Cameras
        CameraParameters1: [1×1 cameraParameters]
        CameraParameters2: [1×1 cameraParameters]

   Inter-camera Geometry
        RotationOfCamera2: [3×3 double]
     TranslationOfCamera2: [-121.1586 -0.5978 -22.5779]
        FundamentalMatrix: [3×3 double]
          EssentialMatrix: [3×3 double]

   Accuracy of Estimation
    MeanReprojectionError: 1.0534

   Calibration Settings
              NumPatterns: 3
              WorldPoints: [54×2 double]
               WorldUnits: 'millimeters'

 estimationErrors =

  stereoCalibrationErrors with properties:

      Camera1IntrinsicsErrors: [1×1 intrinsicsEstimationErrors]
      Camera1ExtrinsicsErrors: [1×1 extrinsicsEstimationErrors]
      Camera2IntrinsicsErrors: [1×1 intrinsicsEstimationErrors]
       RotationOfCamera2Error: [0.0080 0.0020 3.3028e-04]
    TranslationOfCamera2Error: [0.2159 0.4578 2.0783]
```

So we used the code given below to load the parameters before start coding the program.

```
% Load stereo parameters
load stereoParams
```

Input Left and Right Camera Images

In this step we input the images of the object we interest. These left and right view of the object should pictured using the same camera used to do the stereo vision calibration. Images we used are in the resolution of 640*480. We had to take images of the same object many time as we use multiple cameras. So by comparing the image pairs we could suggest the matching pair. Input images are not rectified or modified. Below shows the coding to input and display the images.

```
% Load and read stereo images
LeftCam = imread('1.jpg');
RightCam = imread('3.jpg');

% Display the images before rectification
imshowpair(LeftCam, RightCam);
title('Unrectified Original Images');
```

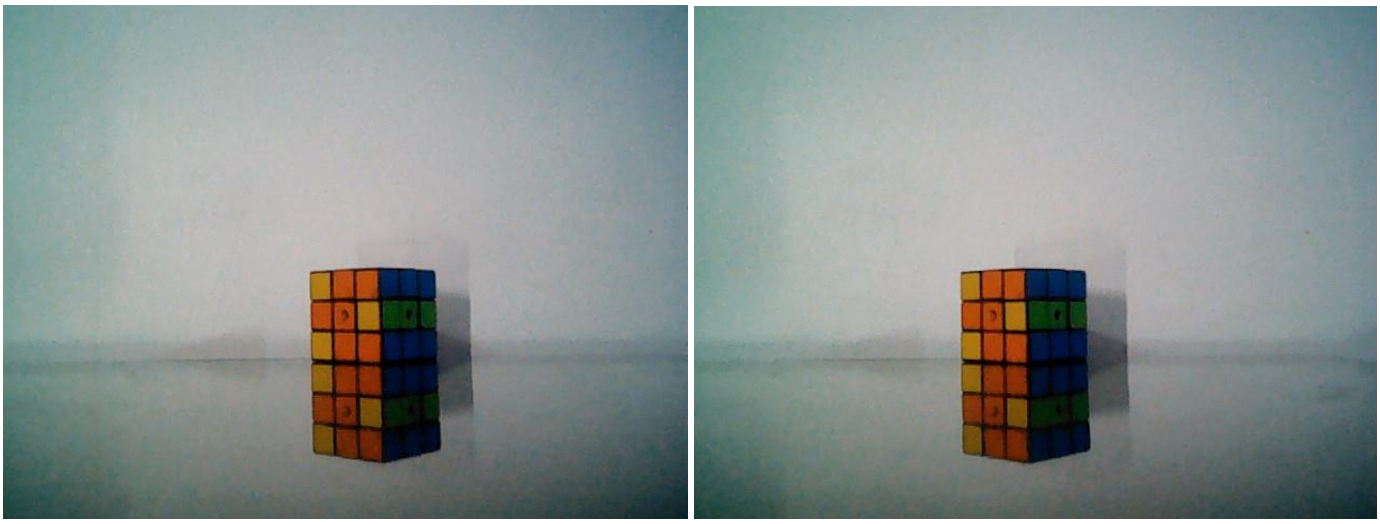


Figure 07: Left and Right stereo camera 2D images which used to input

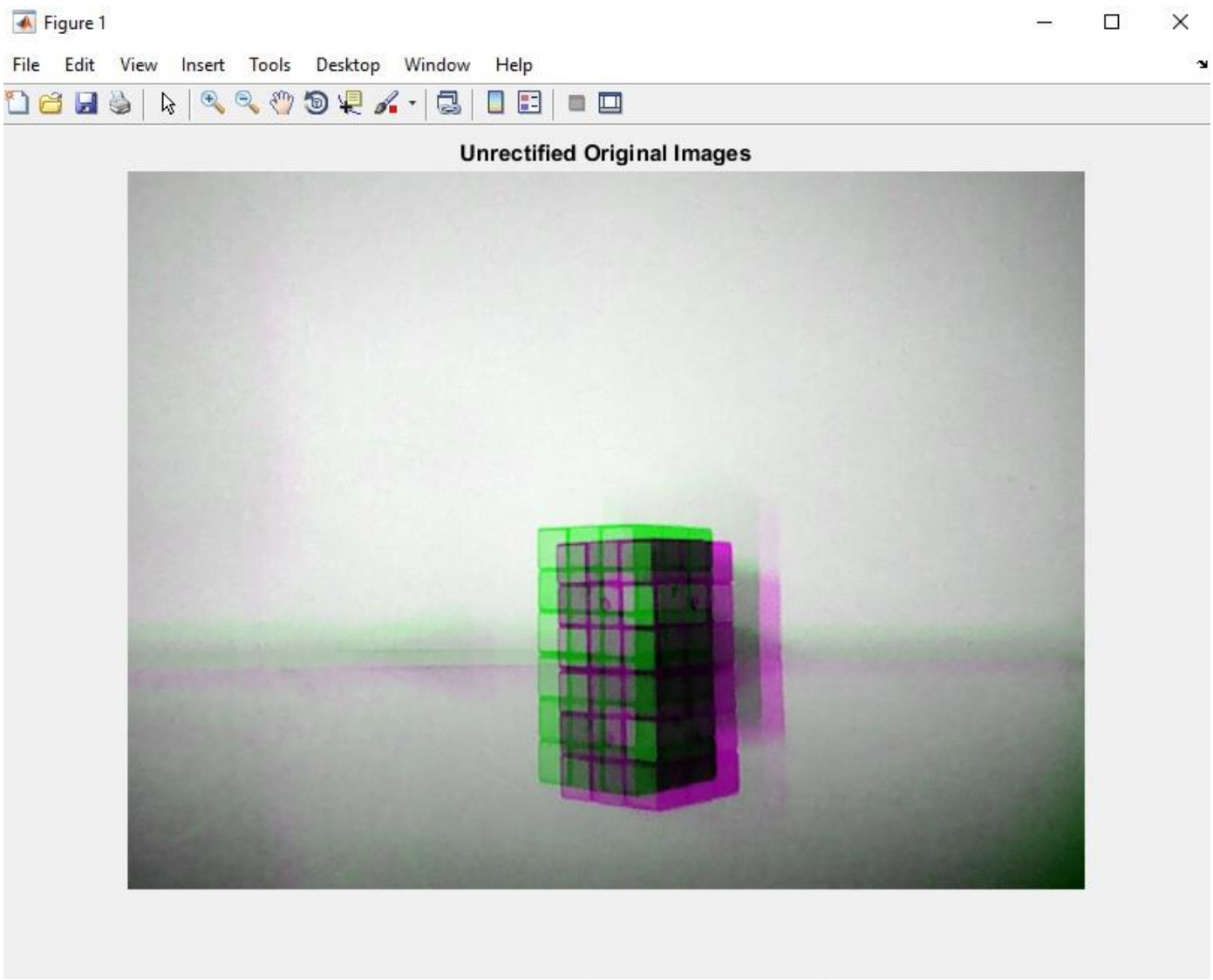


Figure 08: “Unrectified original images” MATLAB output

Rectify Input Images

The rectify stereo images function uses the input images and the stereo parameters to return undistorted and rectified images. After rectification, the two left and right input images have become aligned horizontally. We do this step because we need to create a disparity map with knees rectified images as inputs. If we use un rectified images to the disparity map, then these images will not horizontally be aligned and distorted. Matching pixels in the left and right images in this case is difficult and doesn't give very meaningful results.

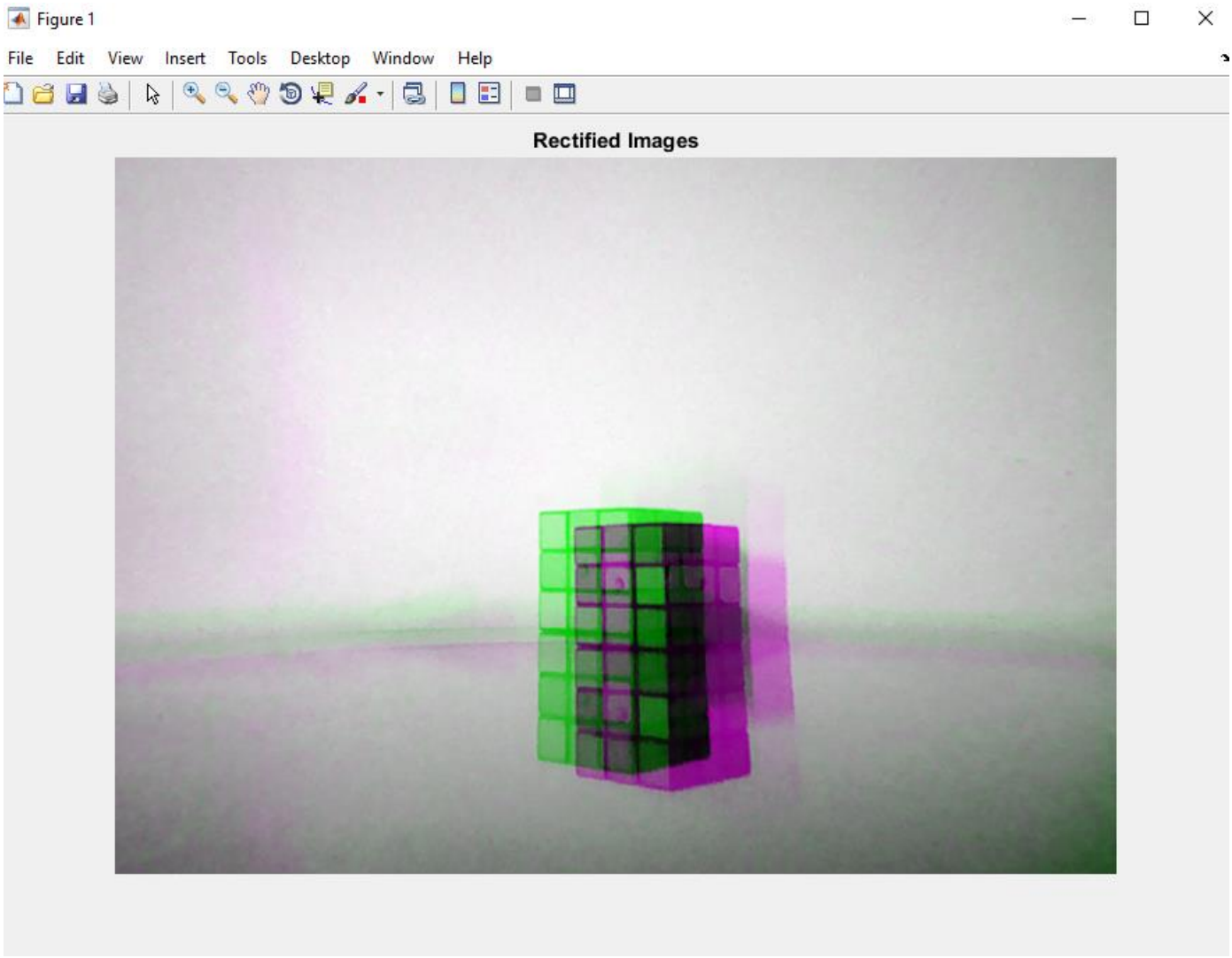


Figure 09: "Rectified Images" MATLAB output

Below shows the MATLAB coding used to rectify unrectified original images.

```
% Rectifing the original
[RecLeft, RecRight] = rectifyStereoImages(LeftCam, RightCam, stereoParams);

% Display the images after rectification
imshowpair(RecLeft, RecRight);
title('Rectified Images')
```

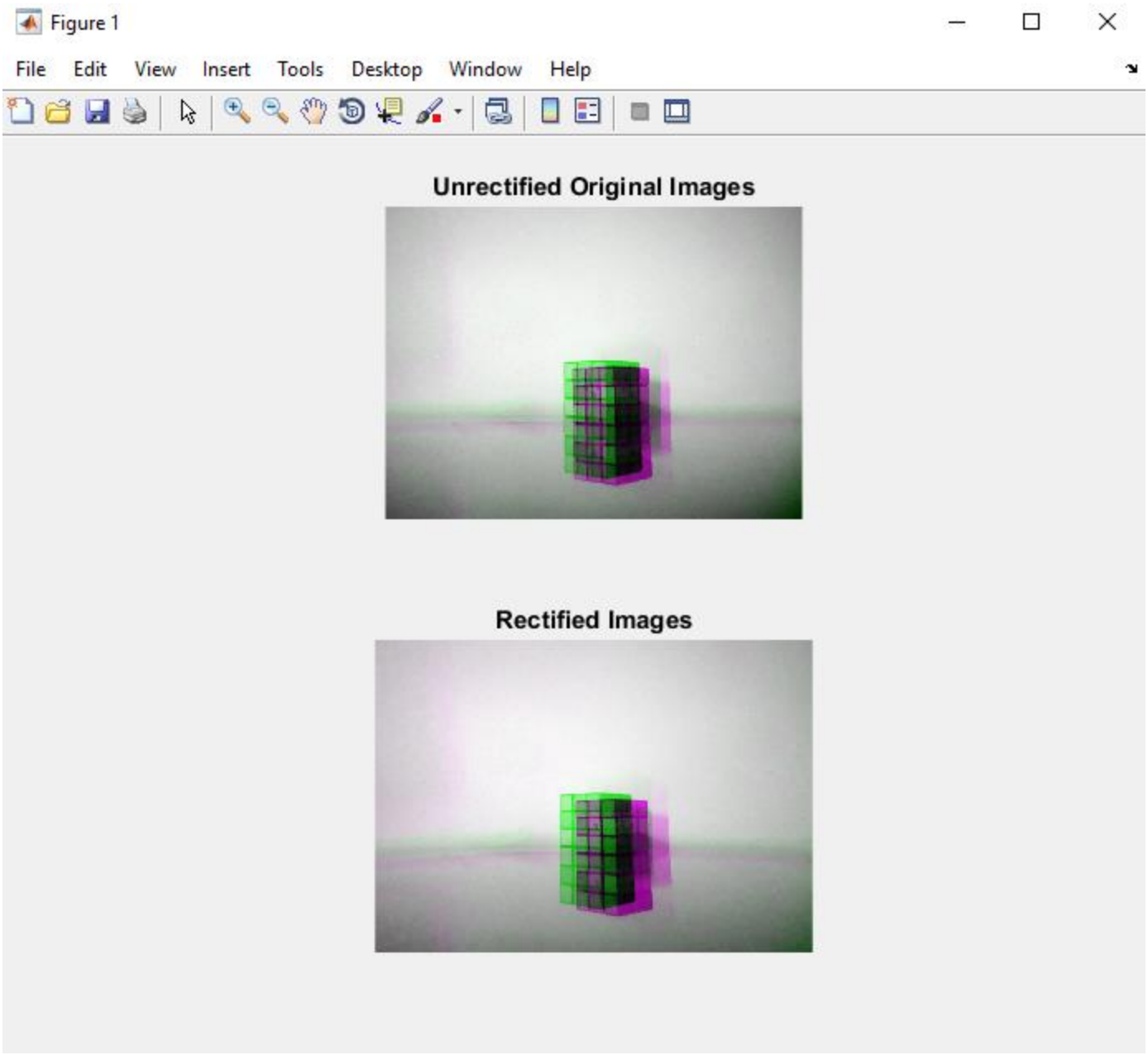


Figure 10: Difference between unrectified and rectified images

By using the subplot(); function we have shown the rectified and un rectified images on one figure. We cannot clearly see the horizontal align in rectified image as we have used multiple cameras. But there is some requirable horizontal align is possible. Below shows the code we used plot above figure.

```
% Display the images before rectification
subplot(2,1,1);
imshowpair(LeftCam, RightCam);
title('Unrectified Original Images');

% Display the images after rectification
subplot(2,1,2);
imshowpair(RecLeft, RecRight);
title('Rectified Images')
```

Create Disparity Map

The disparity map is simply a map of pixel displacements between the left and right images. Rectified images makes the process of matching pixels in the left and right images considerably faster as the search will be horizontal or one dimensional. We can create the disparity map using the disparity function in MATLAB which takes the gray scale rectified images as inputs. Below shows the coding for grayscale conversion and display disparity map.

```
% RGB to GrayScale conversion
LeftG = rgb2gray(RecLeft);
RightG = rgb2gray(RecRight);

% Disparity map generate
disparityMap = disparity(LeftG, RightG);

% Visualize the disparity map
figure, imshow(disparityMap);
title('Disparity Map');
```

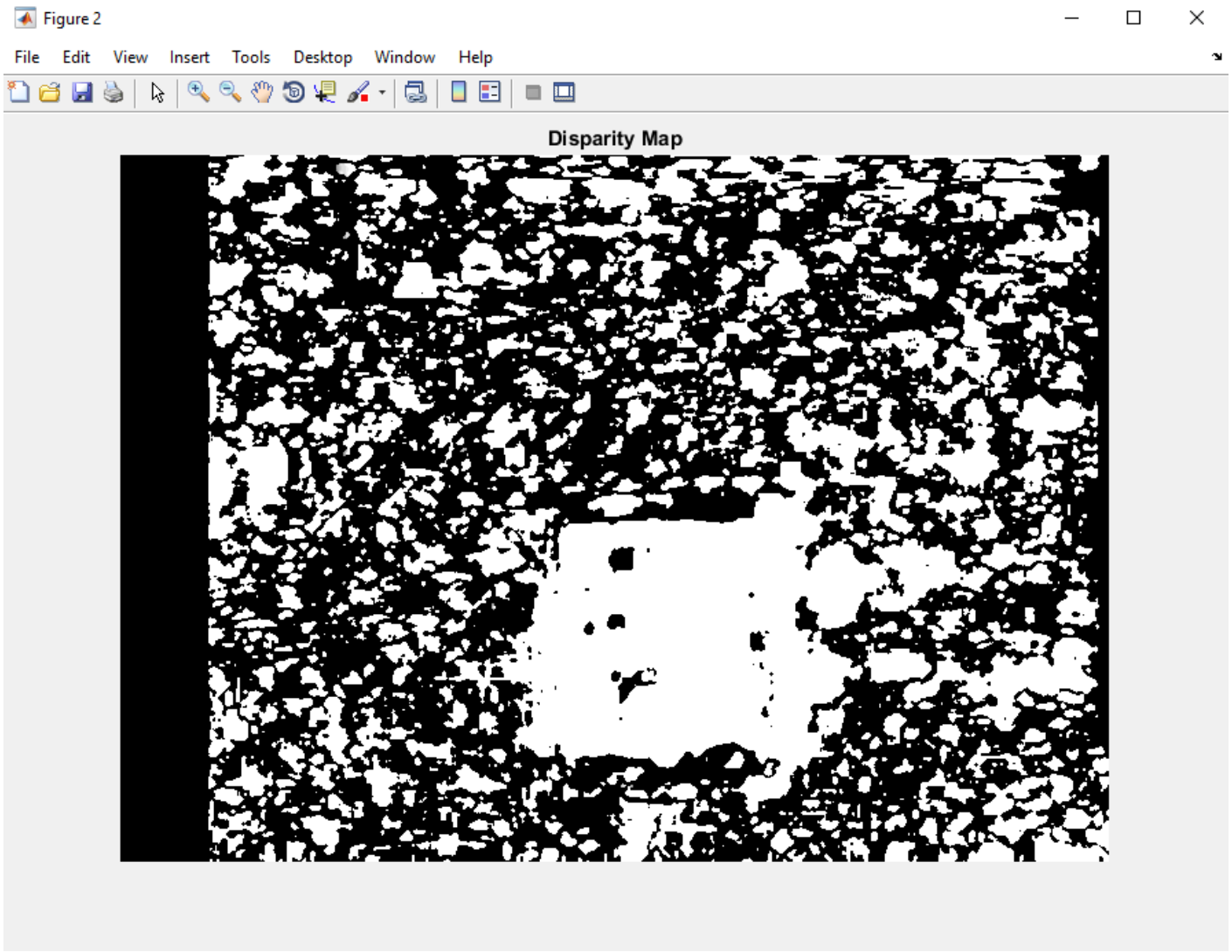


Figure 11: “Disparity Map” MATLAB output

From this image we got, it seems binary. A standard disparity map tends to have high contrast which means gradients less apparent. We checked the maximum and minimum value in the disparity map using the command window.

```
max(disparityMap(:));
ans =
    63

min(disparityMap(:));
ans =
    -3.4028e+38
```

Above are the values we got for maximum and minimum. So, we observed that the contrast is extremely high due to the minimum value being extremely low. Then try to modify and adjust the image contrast by adding a lower and upper bound display function. This forces the intensity values less than or equal to low as black and values more than or equal to high as white. The coding and the result is given as below.

```
% Make adjustments to existing Disparity Map
figure, imshow(disparityMap, [0, 63]);
title('Disparity Map with modified image contrast');
```

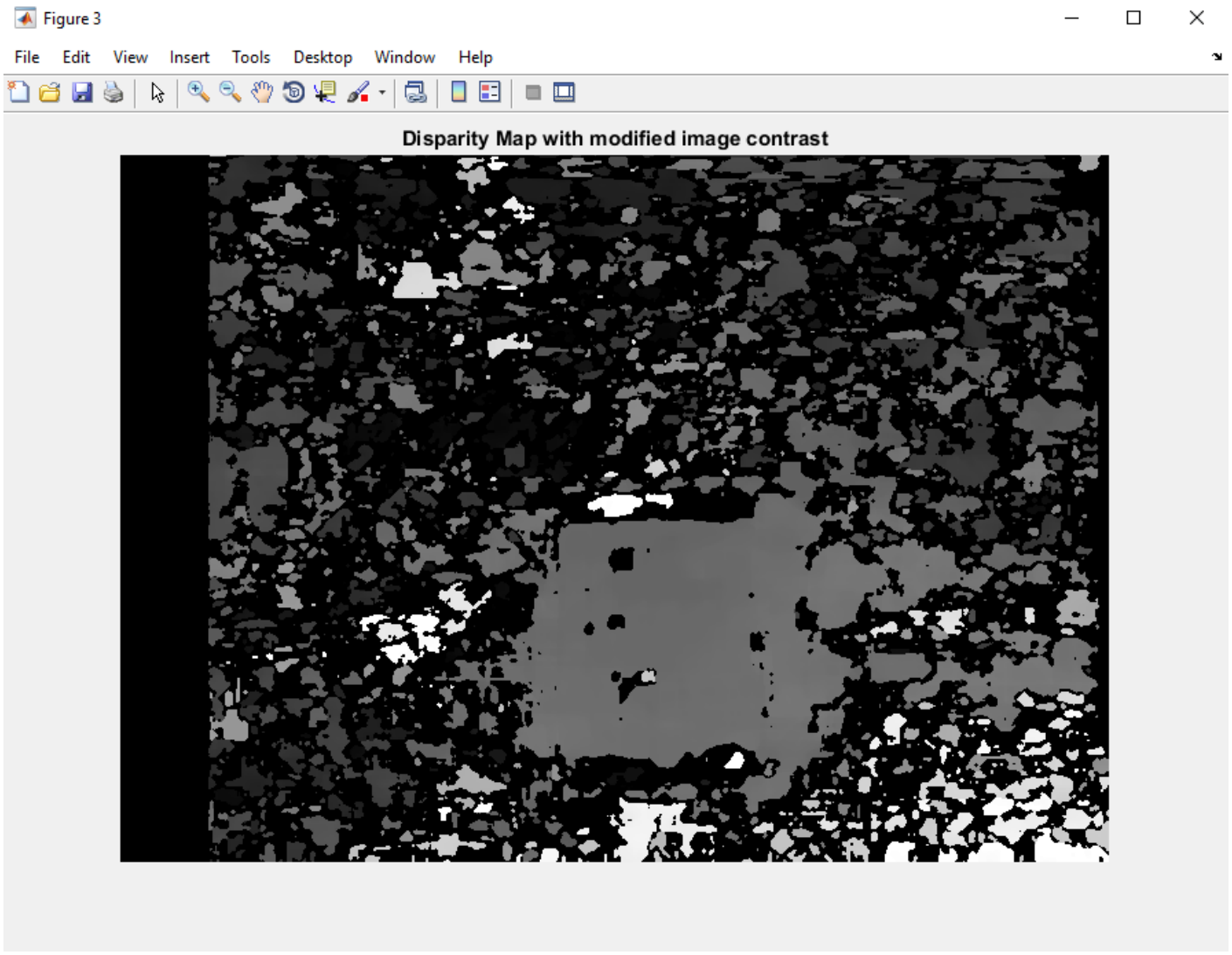


Figure 12: “Disparity Map with modified image contrast” MATLAB output

We can also change the color map from grayscale to some colorful by using the colormap function. And also, we can add a color bar to get some perspective on the color intensities. In this case the pixels closest to us have highest pixel displacement or disparity, therefor it appears in Red. And objects furthest away appear in Blue. The generated code and result are given as below.

```
% Changing the color map to add colors
colormap(gca, jet)

% Add colorbar to get some perspective
colorbar
```

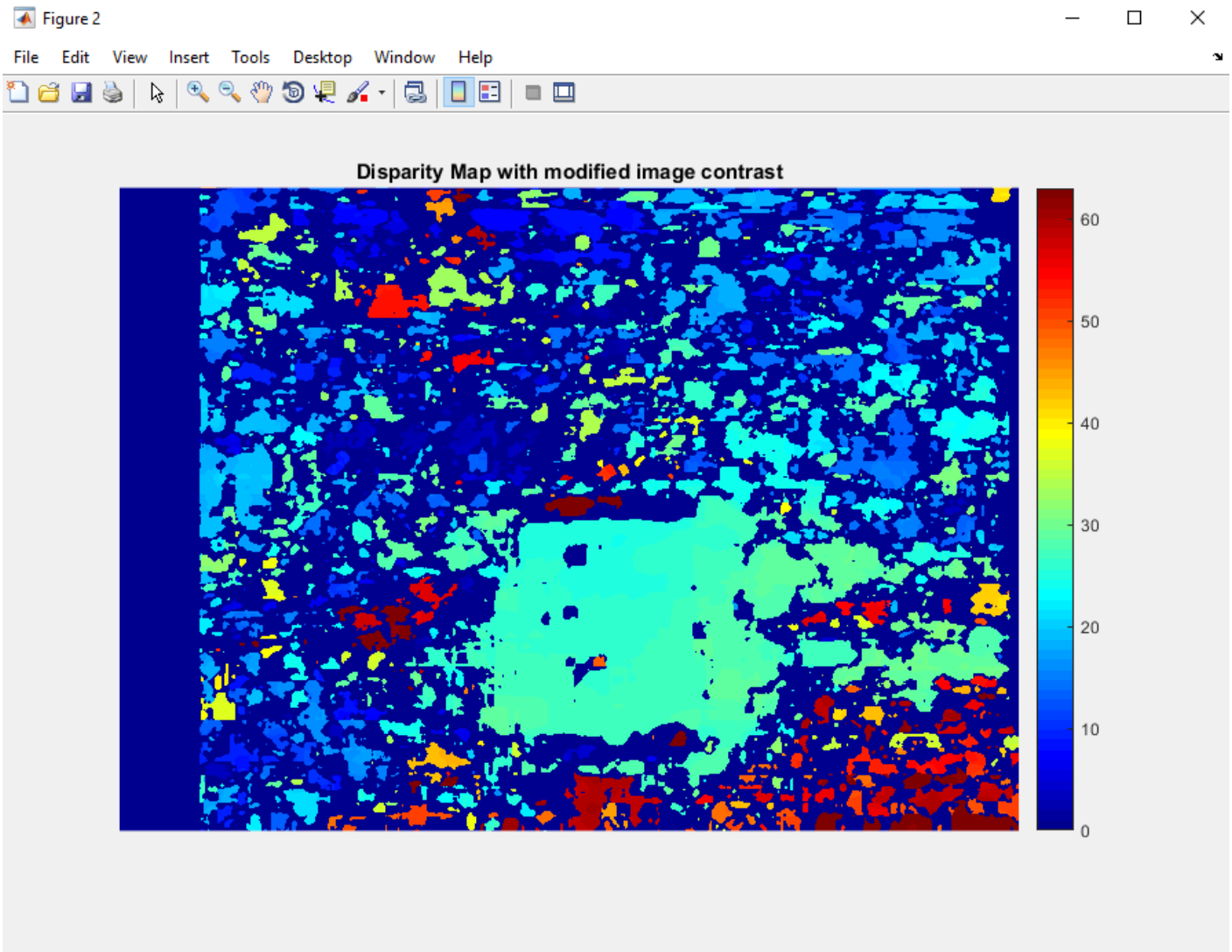


Figure 13: Modified disparity map with added colormap

Below figure 14 shows the difference between disparity maps made with unrectified and rectified images. Disparity map with unrectified images are not horizontally aligned and are still distorted. Matching pixels in the left and right images in this case is difficult and doesn't give very meaningful result.

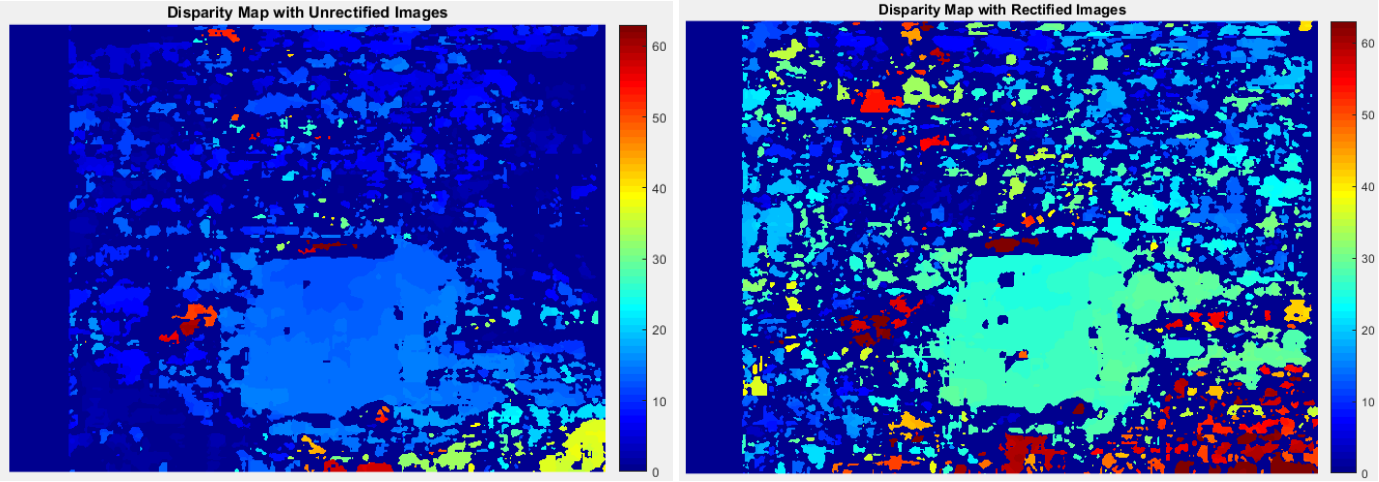


Figure 14: Differnce between disparity map made with Unrectified and Rectified images

Generate Point Cloud

We can generate the point clouds using the reconstructionScenes(); function. This estimate the exact depth for each image point in the rectified image and returns an array of 3D world point coordinates that reconstruct a scene from the disparity map and the stereo parameters. The coding and results are shown as below.

```
% Generate point cloud
points3D = reconstructScene(disparityMap, stereoParams);

% Display point cloud
pcshow(points3D);
xlabel('x'); ylabel('y'); zlabel('z');
```

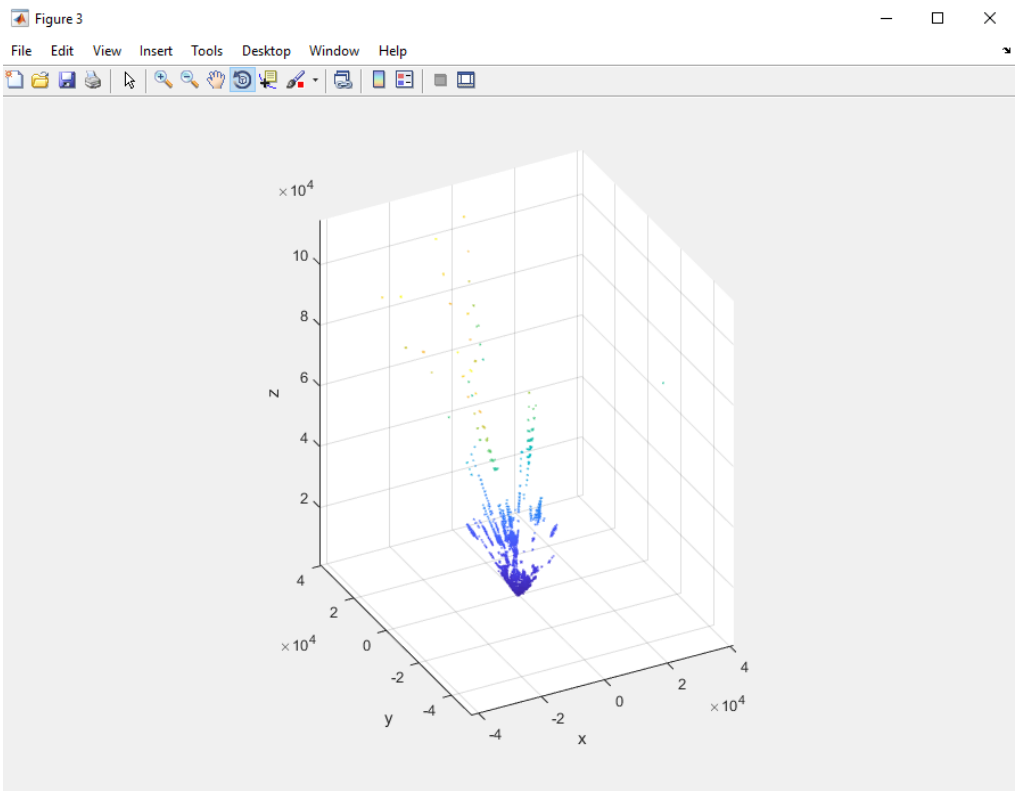


Figure 15: Point Cloud MATLAB output

Reconstruction of 3D Image

The colors of the point cloud don't seem to match because it is using the colors from the disparity map and not the colors from the original image. To add color to the point cloud visualization we can add one of the rectified images as the second argument to the pointCloud(); function. And also, we the x, y, z measures to meters for our convenience. Coding and results are as below.

```
% Convert to meters
points3D = points3D ./ 1000;

% Create a pointCloud object
ptCloud = pointCloud(points3D, 'Color', RecLeft);

% Display point cloud
pcshow(ptCloud);
xlabel('x'); ylabel('y'); zlabel('z');
```

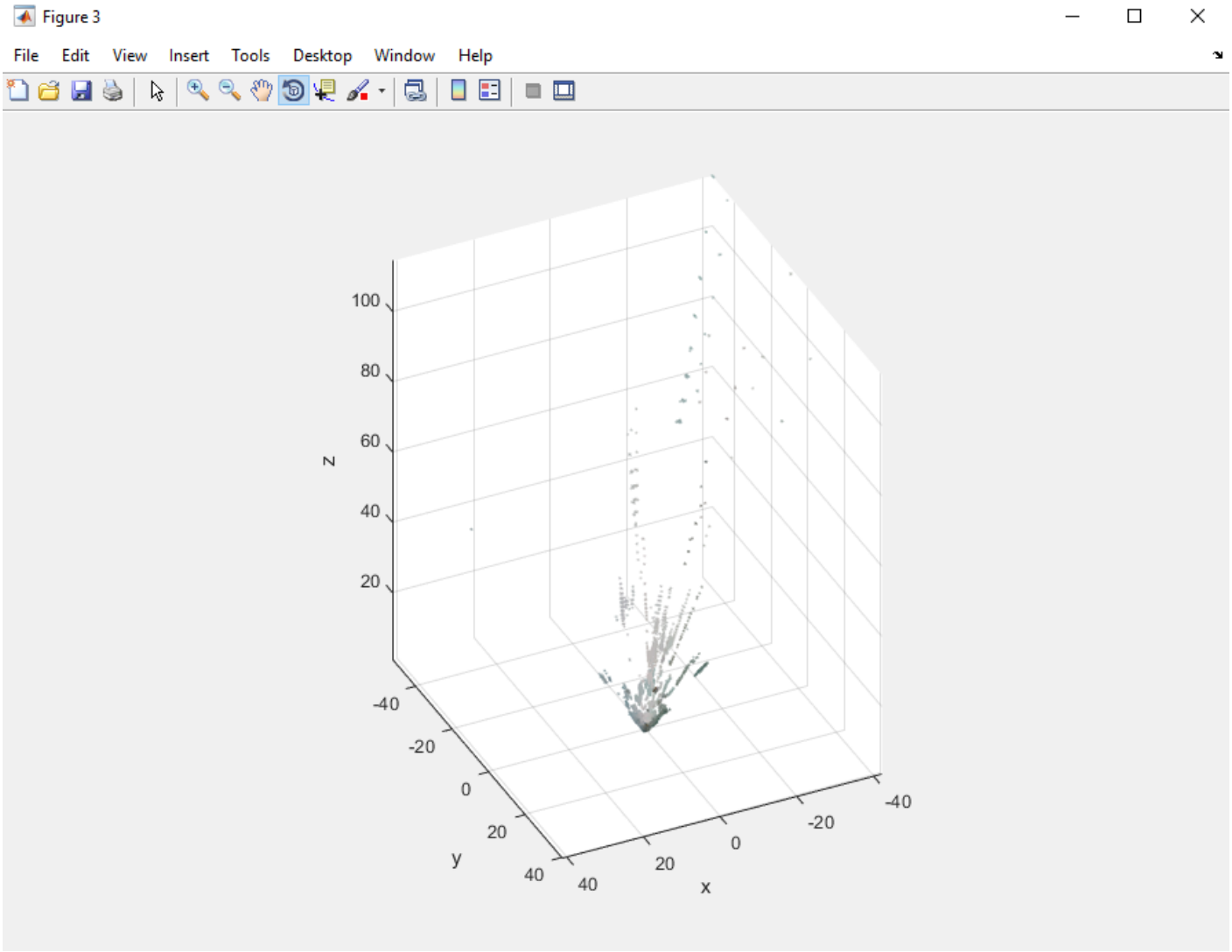


Figure 16: Real colors added Point Cloud MATLAB output

Often, we have to remove outline data from stereo vision generated point cloud. And also have to create a streaming point cloud viewer. Here we got dimensions for x, y, z axis as [-3, 3], [-3, 3], [0, 8] respectively because it gives minimum noise effect. So finally, we got the 3D reconstruction of our captured 2D image using stereo vision.

```
% Create a streaming point cloud viewer
player3D = pcplayer([-3, 3], [-3, 3], [0, 8], 'VerticalAxis', 'y', 'VerticalAxisDir',
'down');

% Visualize the point cloud
view(player3D, ptCloud);
```

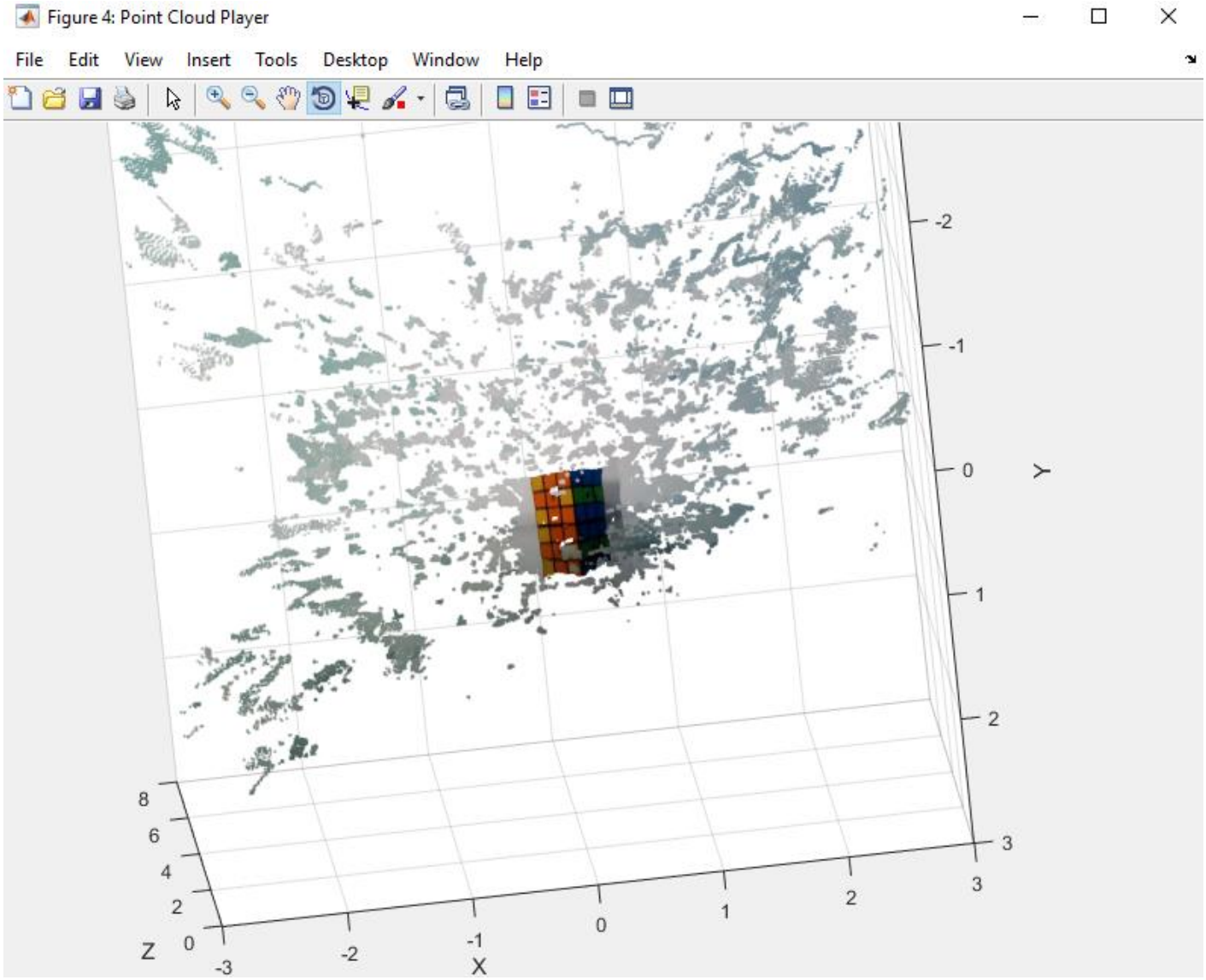


Figure 17: 3D Reconstruction of the input 2D image MATLAB output

The complete code of the whole process is given as below.

```
% Load stereo parameters
load stereoParams

% Load and read stereo images
LeftCam = imread('1.jpg');
RightCam = imread('3.jpg');

% Display the images before rectification
imshowpair(LeftCam, RightCam);
title('Unrectified Original Images');

% Rectifing the original
[RecLeft, RecRight] = rectifyStereoImages(LeftCam, RightCam, stereoParams);

% Display the images after rectification
imshowpair(RecLeft, RecRight);
title('Rectified Images')

% RGB to GrayScale conversion
LeftG = rgb2gray(RecLeft);
RightG = rgb2gray(RecRight);

% Disparity map generate
disparityMap = disparity(LeftG, RightG);

% Visualize the disparity map
figure, imshow(disparityMap);
title('Disparity Map');

% Make adjustments
figure, imshow(disparityMap, [0, 63]);
title('Disparity Map with modified images contrast');

% Changing the color map to add colors
colormap(gca, jet)

% Add colorbar to get some perspective
colorbar

% Generate point cloud
points3D = reconstructScene(disparityMap, stereoParams);

% Display point cloud
pcshow(points3D);
xlabel('x'); ylabel('y'); zlabel('z');

% Convert to meters and create a pointCloud object
points3D = points3D ./ 1000;
ptCloud = pointCloud(points3D, 'Color', RecLeft);

% Display point cloud
pcshow(ptCloud);
xlabel('x'); ylabel('y'); zlabel('z');
```

```
% Create a streaming point cloud viewer
player3D = pcplayer([-3, 3], [-3, 3], [0, 8], 'VerticalAxis', 'y',
'VerticalAxisDir', 'down');

% Visualize the point cloud
view(player3D, ptCloud);
```

Conclusion

There are many ways of 3D scene reconstruction using stereo vision. Out of that one way, I have used MATLAB R2017a as the software tool for this project. It was very hard to calibrate the stereo camera using stereo vision calibrator app because we used multiple cameras. As we used two identical mobile phone cameras (Samsung M20) to capture images. First, we constructed a base to hold phones stationary. But we got big reprojection errors. After trying many times, we reduced it to 1.05. Then we got some problems on the resolution of input images. So, we had to reduce resolution of images to 640*480. Sometimes captured pictures doesn't matched as left, right images because the distance between two cameras were incorrect. Finally, we got accurate images at the distance of 7.8cm after checking for many distances. As we use MATLAB for this modelling, it was very helpful because inbuilt libraries were there which need for this. With many difficulties we finalized the mini project task.