1130-EMARO-MSA-1006 # Computer Vision

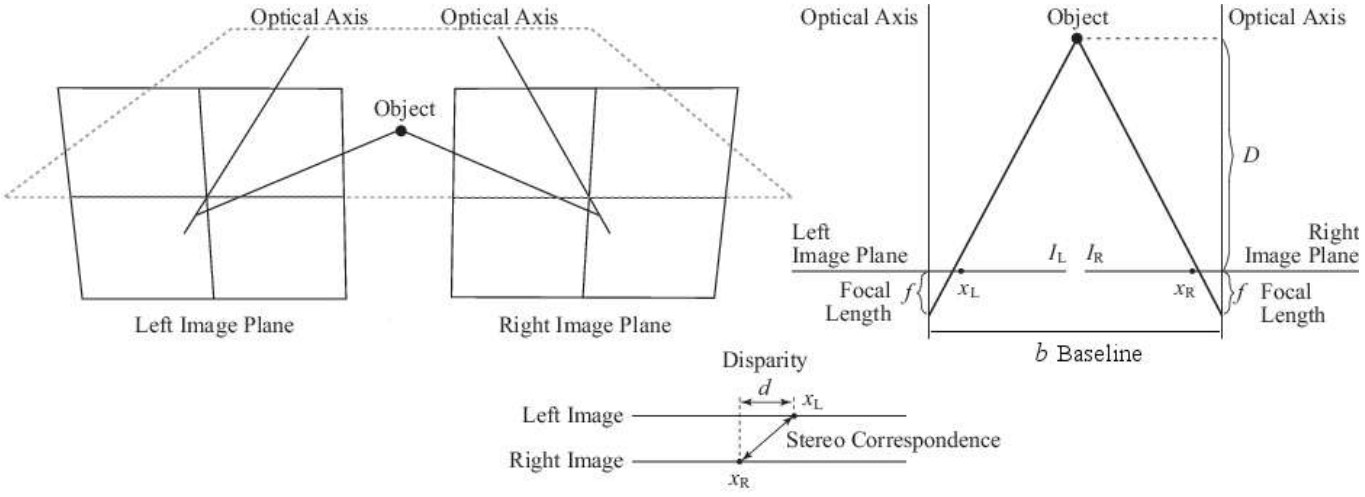
T3: Stereovision

Stereo geometry

From Wikipedia, the free encyclopedia: A **stereo camera** is a type of camera with two or more lenses with a separate image sensor or film frame for each lens. This allows the camera to simulate human binocular vision and therefore gives it the ability to capture three-dimensional images, a process known as stereo photography.

The basic phenomenon, which allows for the calculation of distance based on two shifted pictures is parallax - the same point in space, observed from different points of view, will have different positions on the picture. To make the calculations easier, we may assume, that both imaging planes are parallel (optical axes are parallel) and both cameras have the same orientation. In this case point in space is mapped on both images on the same horizontal line (in general case those are called epipolar lines).

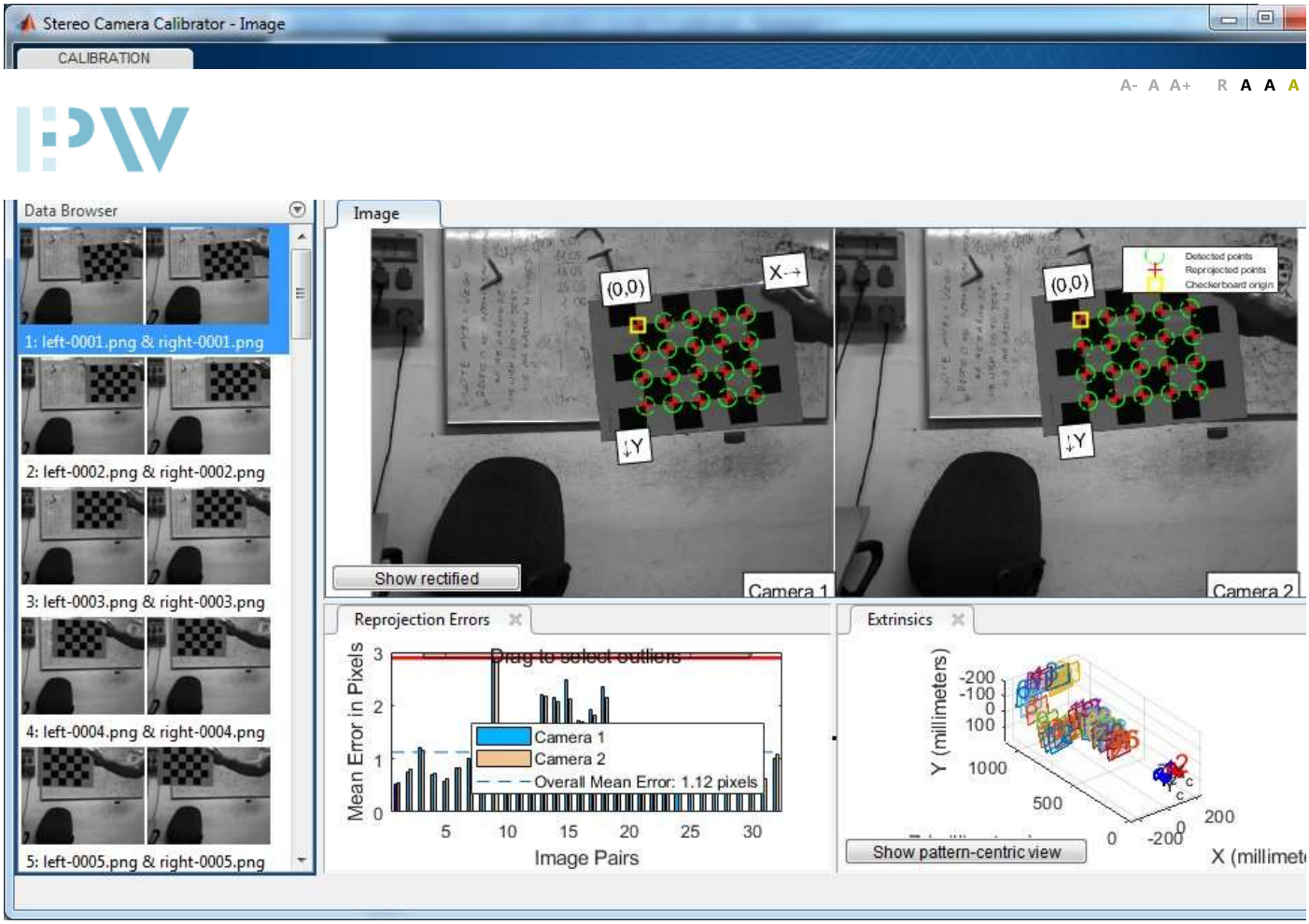
The important parameters of the stereo system are presented on the picture below.



Stereo calibration

In order to be able to retrieve 3D information from pairs of stereo images, it is necessary to know the intrinsic parameters of each camera (i.e. *f* and *c*) as well as extrinsic parameters of the system (i.e. position of one camera relative to the other). In real-world applications, it is impossible to have perfectly aligned cameras, so extrinsic parameters contain both translation and rotation between cameras.

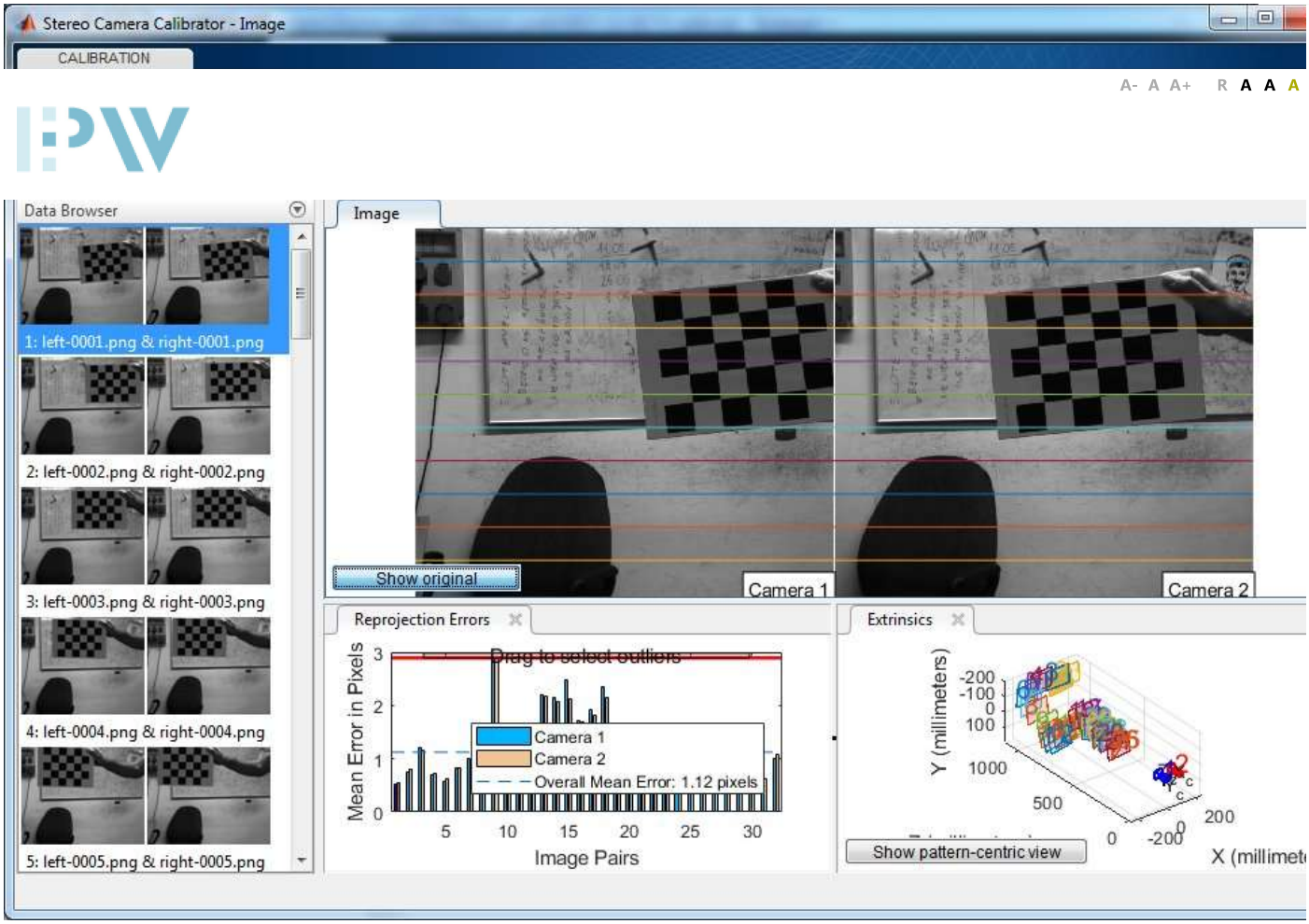
Use Stereo Calibrator App to calibrate both intrinsic and extrinsic parameters. Take the images from this dataset.



Stereo rectification

When two cameras are calibrated, it is possible to simulate perfect parallel cameras, using the process called rectification. Given two images and stereo parameters, pictures are transformed in such a way, that for a given point in space, its *y* coordinate is the same in both pictures.

You can view the rectified pictures after calibrating the stereo camera by clicking on the "Show rectified" button. Horizontal lines can be treated as guides to check, whether the calibration was successful and the corresponding points are really on the same line.



Export parameters

After calibration export the parameters to Matlab. The following tasks require those parameters to be known.

Image rectification and disparity range estimation

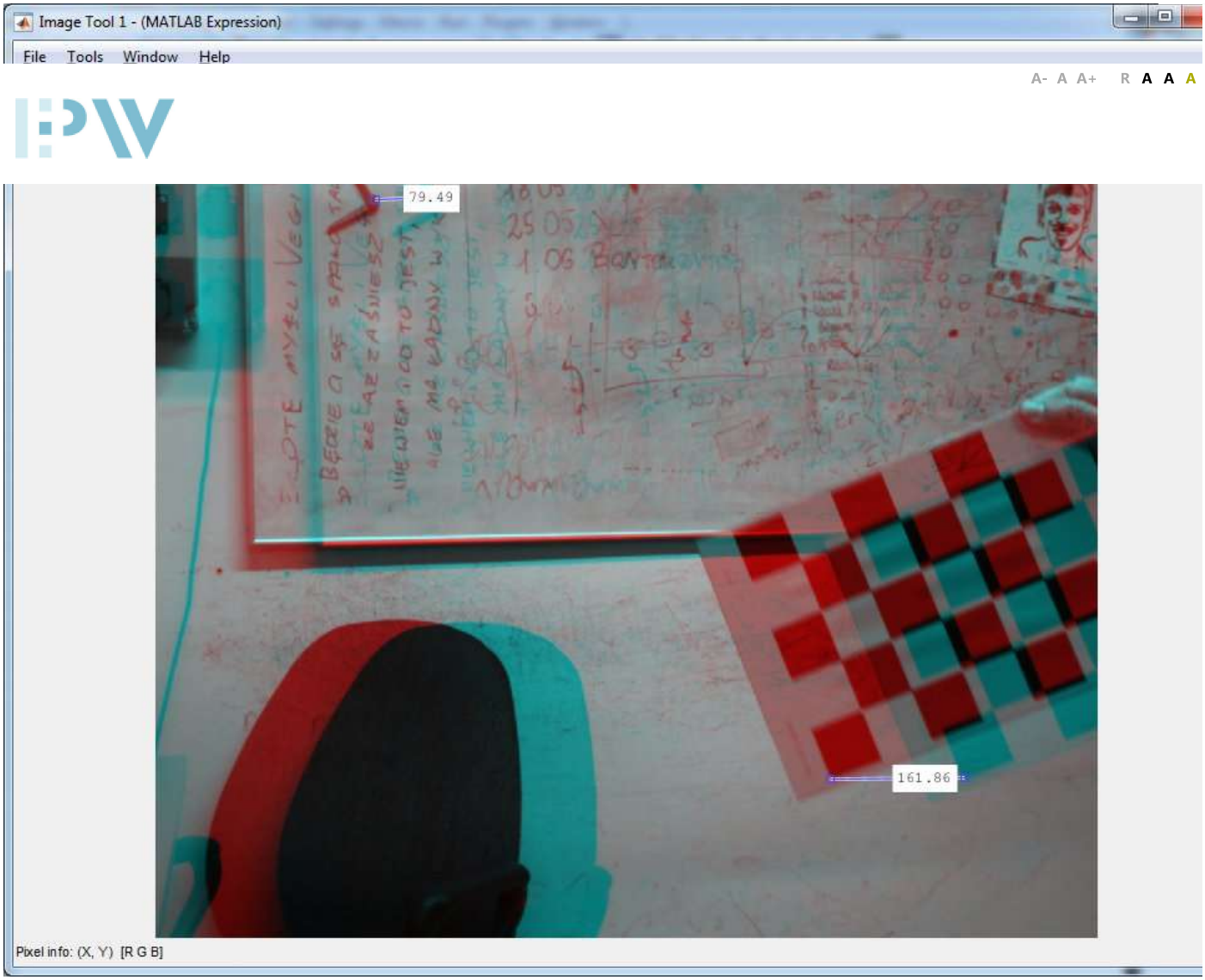
As has been said before - the necessary step for stereo calculations is the rectification of input images. This can be done using the rectifyStereoImages function:

```
% read pair of images
I1 = imread('left/left-0000.png');
I2 = imread('right/right-0000.png');

% rectify images using calculated stereo parameters
[J1, J2] = rectifyStereoImages(I1, I2, stereoParams);
```

Now you can display a mix of left and right images as anaglyph (red-cyan) picture:

```
imtool(cat(3, J1, J2, J2));
```

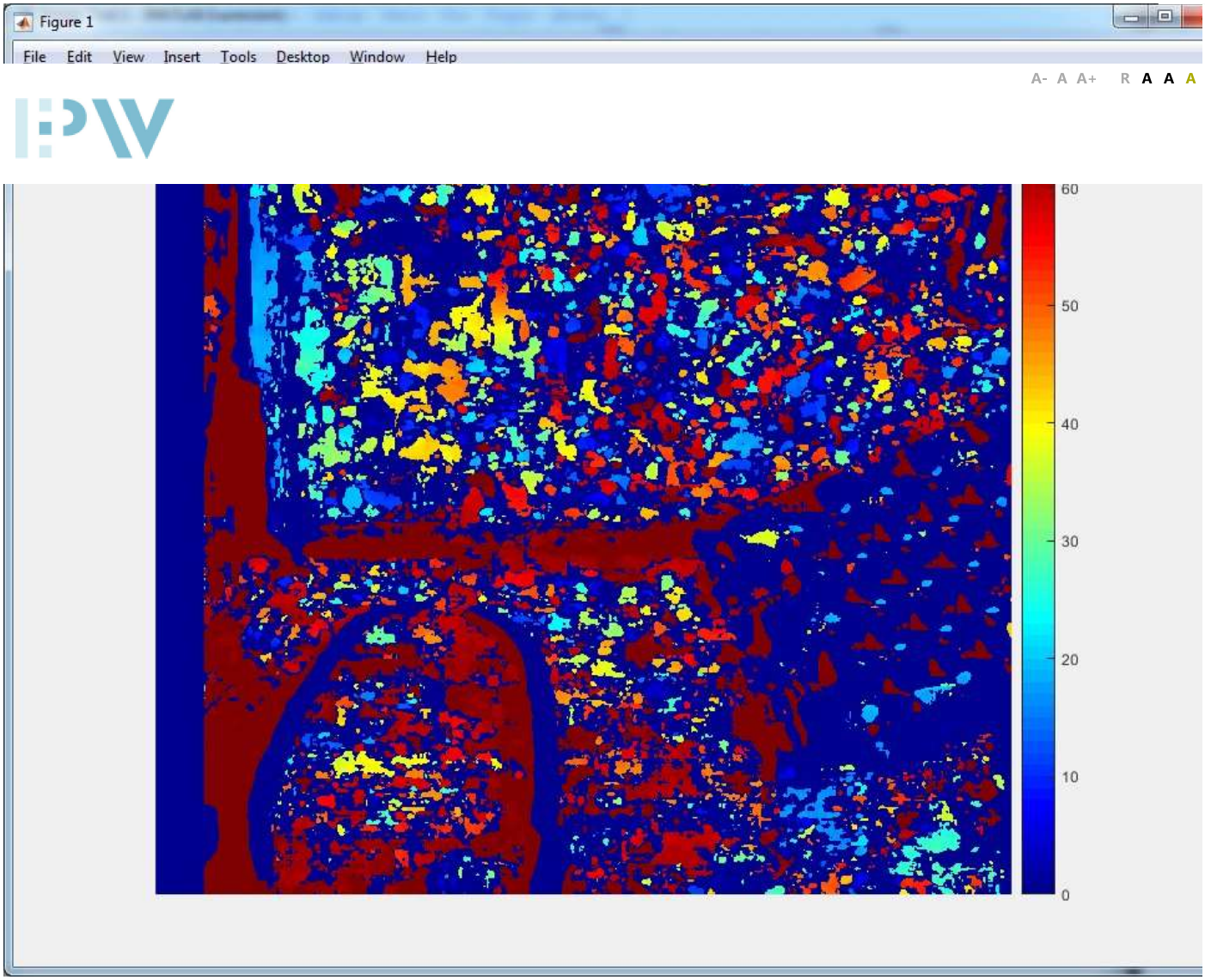
Using the imtool you can measure disparity (distance) between points that are close to and far from the camera to get the range for the disparity.



Calculating disparity map

In Matlab, the disparity function is used to calculate the disparity map from the two rectified images. There are few parameters to be tuned, mainly the disparity range. For the default parameters shown below, the result is not very nice (a lot of noise and wrong measurements).

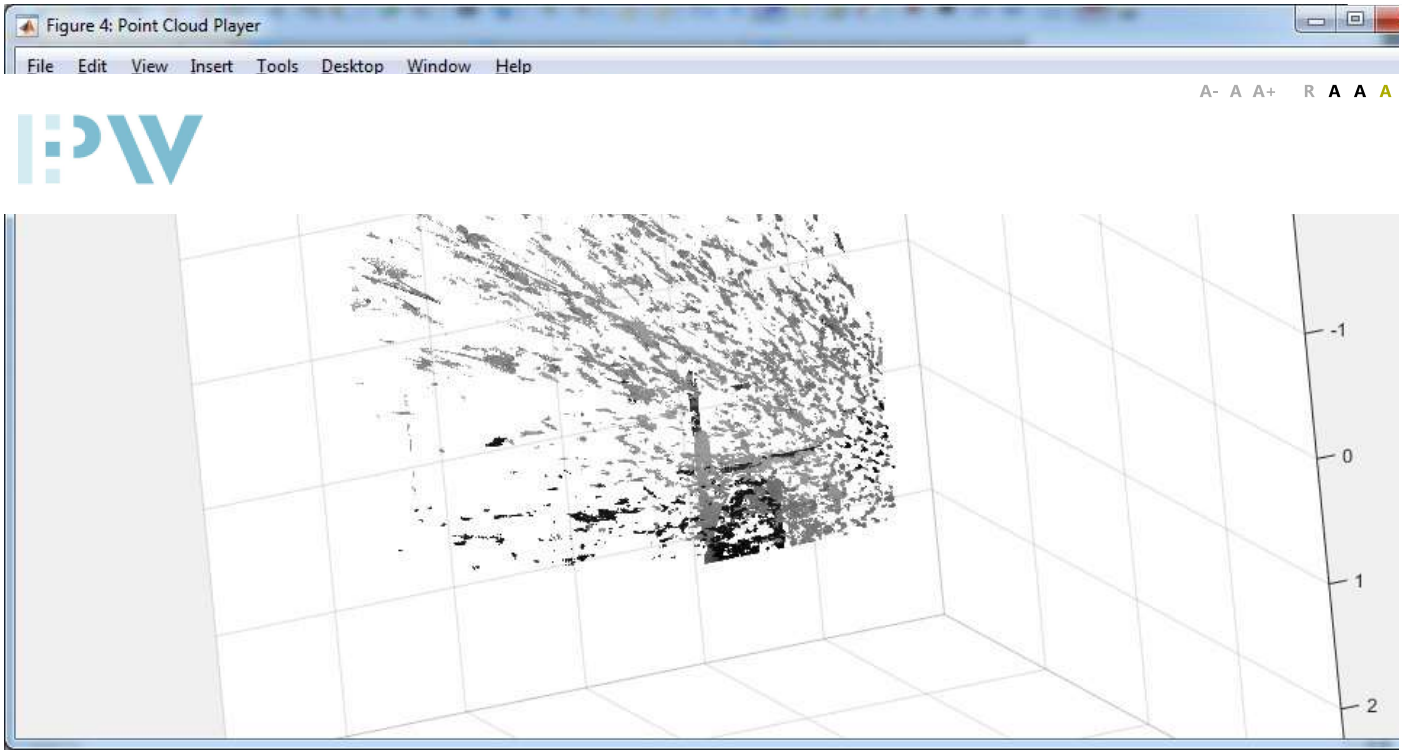
```
% disparity range
dispRange = [0, 64];
disparityMap = disparity(J1, J2, ...
    'DisparityRange', dispRange, ...
    'BlockSize', 15, ...
    'ContrastThreshold', 0.5, ...
    'UniquenessThreshold', 15 );

% show disparity map
figure
imshow(disparityMap, dispRange);
colormap(gca,jet)
colorbar
```



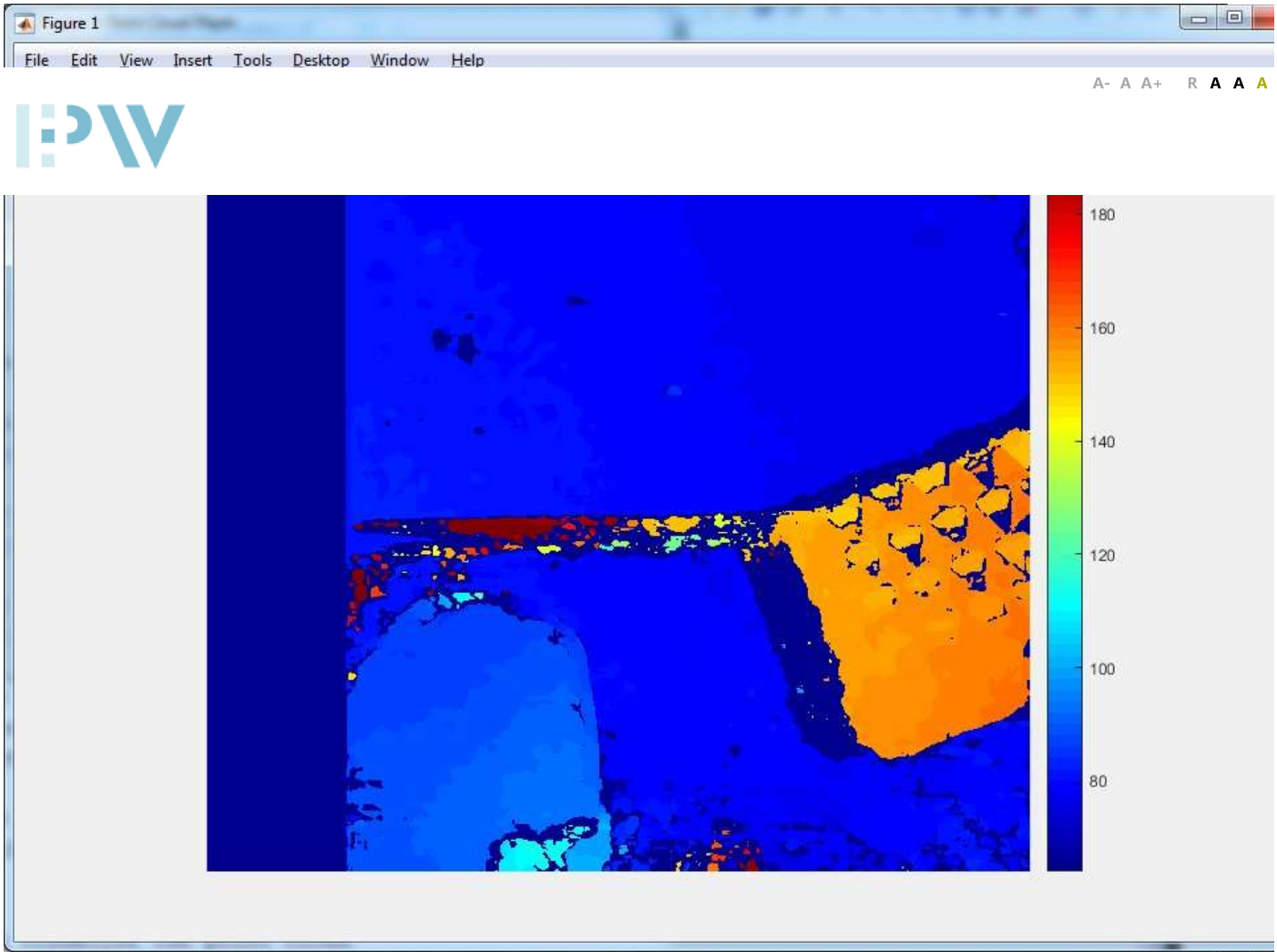
Calculating 3D points from the disparity map

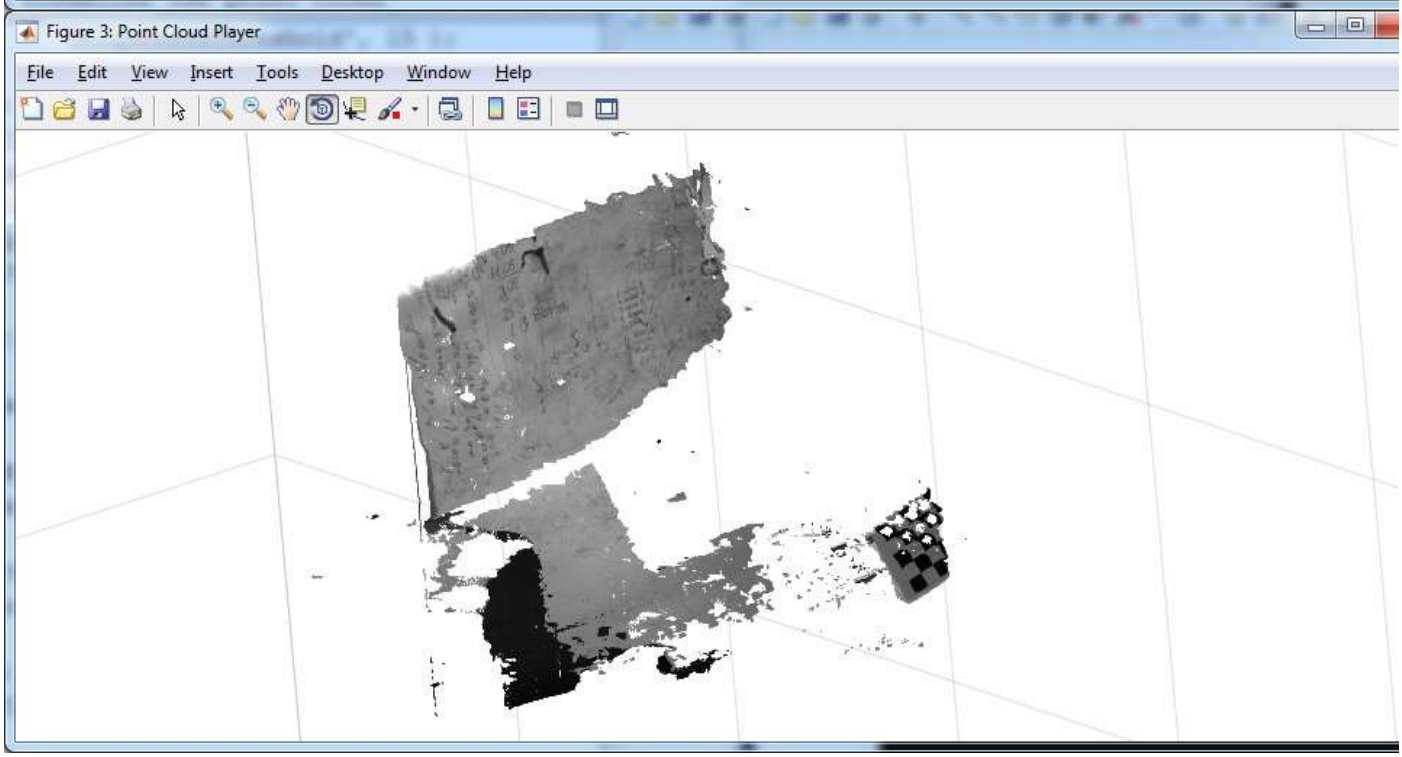
When a disparity map is calculated, you can convert it to full 3D coordinates of points. Based on the baseline, focal length, and disparity value X, Y, and Z is calculated.

```
points3D = reconstructScene(disparityMap, stereoParams);

% expand gray image to three channels (simulate RGB)
J1_col = cat(3, J1, J1, J1);

% Convert to meters and create a pointCloud object
points3D = points3D ./ 1000;
ptCloud = pointCloud(points3D, 'Color', J1_col);

% 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);
```



Expected results for good parameters





Status przesłanego zadania

Status przesłanego zadania	Przesłane do oceny