SGN-34006 3D and Virtual Reality

Fusing depth and color camera images

1 General Instructions

The objective of this exercise is to learn how to create and calibrate multi-camera system assisted by a range sensor device, and how to fuse captured data for 2D/3D visualization.

Tasks

The exercise consists of several tasks related to calibration and re-sampling of data captured with a multi-camera setup assisted by range capturing device. It includes home assignments and a laboratory session. During the lab session, test data is captured and estimation of calibration parameters is performed.

Groups

The assignment is done in groups of two or three. All group members are expected to be present in the laboratory session and to contribute to the completion of the assignment.

Time slots

Time slots for the lab session will be allocated towards the end of the assignment. Book a time slot in the Moodle pages of this assignment. One hour per group is reserved, during which the lab tasks should be completed.

Data

The homework tasks should be completed using synthetic data provided in Moodle. During the lab session, data will be collected which will be required for finalizing homework tasks. The laboratory data should be collected during the available time slots. Each group should bring a USB storage device to take the data.

Deliverables

Each group should return:

Matlab script that does the required work. When the script is executed in the directory containing the captured data, it should execute without errors and display the requested figures for all tasks. The figures produced by the demo code, which have “Task x:” in the title should be reproduced by your script. Other figures are for guidance only, and do not need to be drawn by you.

The submission should be all Matlab codes and the image data your script needs for functioning in a single **ZIP** file. See Moodle for the exact deadline. Possible extensions to the deadline should be negotiated **before** the deadline and may be awarded at the discretion of the assistant if valid arguments are presented.

If an extension is not agreed on beforehand, submissions after the deadline will be accepted, but a penalty will be applied to the grading of the assignment. The grade will be decremented by one for being late, and by one for each additional five days it is late.

2 Multi-Camera System Calibration and 2D/3D Fusion

The objective of camera calibration is to provide an accurate description on how the observed physical objects are captured by a system of several imaging devices. Thus, camera calibration is a process which describes the parameters of how a camera captures a scene. The calibration process also estimates the pose and location of cameras in a system with respect to the real world.

For the purposes of this laboratory work, a multi-camera system augmented with a range sensing device is used. A range sensor contributes geometrical information to the system, which can then be used to enhance the captured representation of the scene. An example of such a device is Microsoft Kinect (Figure 1), a game controller for the Xbox One gaming console.

Figure 1 Example of a time-of-flight range sensor - Microsoft Kinect 2.0

2.1 Multi-camera systems

A multi-camera systems can consist e.g. of two (i.e. stereo cameras) or more 2D color cameras, assisted by a range sensor (3D). The cameras are synchronized together, which ensures data acquisition happens at the same time in all devices. This is essential if the scene contains any dynamic elements. In addition to the hardware, a suitable software component is required in order to utilize the results of the data acquisition in any way. The data must be rectified, projected, re-sampled and fused to create a valid representation of a 3D scene.

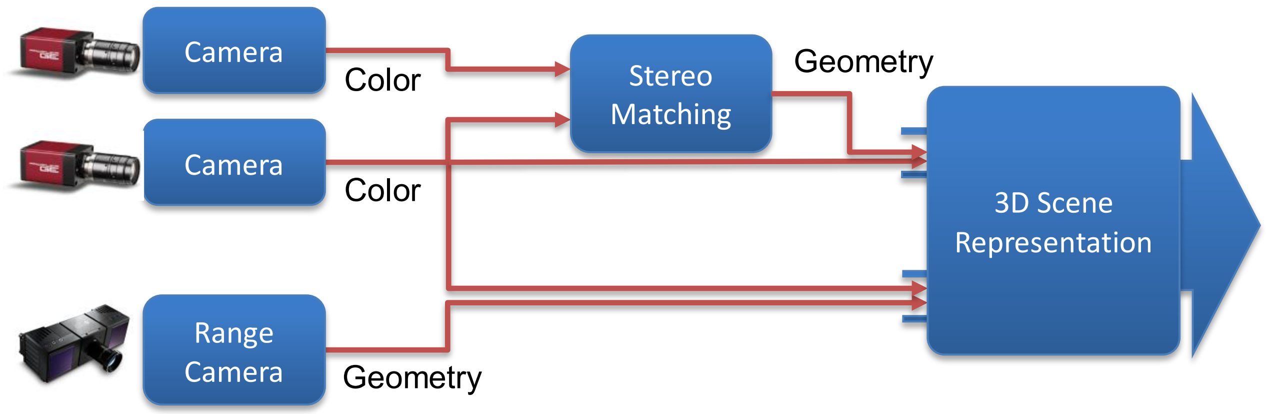


Figure 2 Example diagram of a multi-camera system and the alternative ways of reaching a similar 3D scene representation

2.2 Pinhole camera model

The pinhole camera model describes how points in physical plane appear as projections in the image plane of the camera. The pinhole model represents a camera as a system with an *optical center* ***O***(the pinhole), and an image plane. Each point $P(x,y,z)$ in the observed 3D space is mapped to the sensor as a 2D point $I(u,v)$, which is the intersection of a line going through both P and the pinhole, and the image plane. The line perpendicular to the image plane and passing through the optical center is the *optical axis*. The intersection of the optical axis and the image plane is the *principal point*. The optical axis and the principal point define the capturing direction of the camera. The distance from the optical center to the principal point is called the *focal length*, *f*.

From the rule of similar triangles, a relation between (x,y,z)-coordinates of a point in the 3D space and (u,v)-coordinates of the projection on the image plane can be formulated as

$$(u,v) = \left(f\frac{x}{z}, f\frac{y}{z}\right). \qquad 1$$

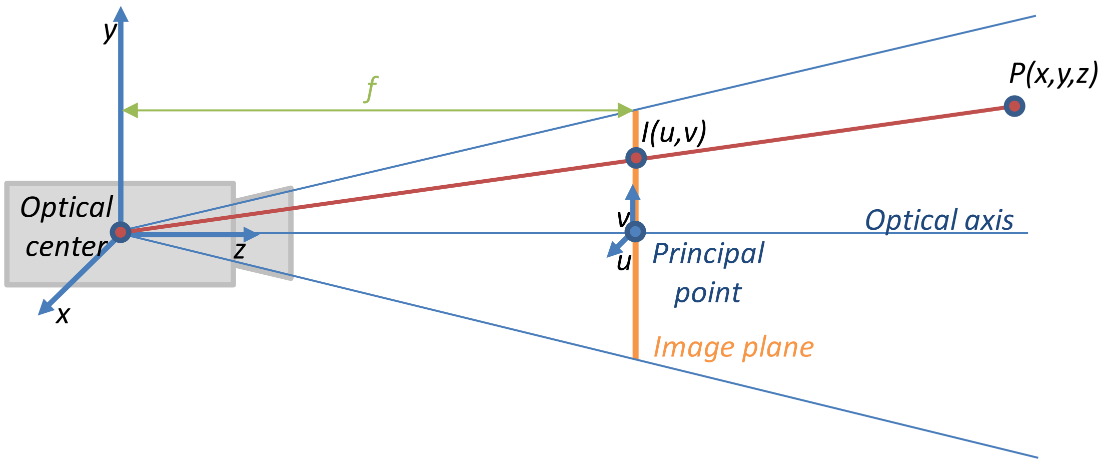


Figure 3 Pinhole camera model

However, the simple Eq. 1 only represents the ideal case. In a real camera, the pixel coordinates of the principle point deviate from the center of the image sensor. Furthermore, sensor elements are typically not perfect squares, which has implications to the conversion between the physical and pixel-wise coordinates. The calibration software used in this exercise follows the convention of including this conversion factor in the focal length, therefore giving different focal lengths for vertical and horizontal directions, f_x and f_y. This leads to a coordinate transform of the form

$$(u,v) = \left(f_x\frac{x}{z} + c_x, f_y\frac{y}{z} + c_y\right). \qquad 2$$

where c_x and c_y are the coordinates of the principal point. The parameters of Eq. 2 are called *intrinsic parameters*, or more commonly, *camera intrinsics*.

2.3 Camera pose relation in multi-camera systems: rotation matrix and translation vector

The pose of a camera with respect to global coordinates is described in terms of rotations of the camera axes and translation shifts of the optical center in relation to a reference point. Usually, in a stereo camera setup, the reference point is chosen to be the optical center of the left camera.

The physical rotation of the camera axes is implemented as consecutive 2D rotations around the optical center and with respect to the coordinate axes of the global space.

In terms of matrix operations, 2D rotations around the coordinate axes, pivoting around the optical center, are defined as:

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\varphi) & -sin(\varphi) \\ 0 & sin(\varphi) & cos(\varphi) \end{bmatrix}, R_y(\Psi) = \begin{bmatrix} cos(\Psi) & 0 & sin(\Psi) \\ 0 & 1 & 0 \\ -sin(\Psi) & 0 & cos(\Psi) \end{bmatrix}, R_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad 5$$

which can be combined as a single rotation matrix,

$$R(\varphi, \Psi, \theta\ ) = R_x(\varphi)\, R_y(\Psi)\, R_z(\theta), \quad 6$$

and which is applied by matrix multiplication of R and the point, $\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{new} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix}$.

The translation component is simple: the difference between the coordinates of the optical centers of the camera,

$$T = O_R - O_L = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_R - \begin{bmatrix} x \\ y \\ z \end{bmatrix}_L, \quad 7$$

which is applied as addition to the coordinate point, $\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{new} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + T$.

2.4 Camera projection matrix

The camera projection matrix *P* combines both camera intrinsics and extrinsics and thus describes completely how a point with position (x, y, z) in the physical world is captured by a camera setup,

$$P = M[R_{3\times3} \quad T_{3\times1}], \quad 8$$

where $[R_{3\times3} \quad T_{3\times1}]$ means concatenation of matrices.

The pixel position (u, v) of world point (x, y, z) is found in the homogeneous coordinate notation as

$$\begin{bmatrix} u' \\ v' \\ k \end{bmatrix} = \mathrm{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \qquad (u, v) = (u'/k, v'/k). \qquad 9$$

3 Exercise tasks

3.1 Summary of tasks

The tasks in the exercise are divided into two groups – *Homework* and *Lab session*. . The homework tasks are done by the group independently before the lab session. The lab tasks are performed in the laboratory and are mandatory. Completing the mandatory tasks will result in grade 1. Any additional task completed will increase the grade by one. Additional tasks don't have to be completed in numerical order.

3.1.1 Lab session tasks

During the lab session, the following steps should be completed

L1. Collecting calibration data
L2. Capturing a set of still images
L3. Extracting the calibration parameters of the camera system

The parameters of the calibrated system and the captured images are needed to finalize the homework.

3.2 Homework tasks

All results here should be drawn when the Matlab script is executed.

H1. Estimating the 3D global coordinates from captured range data (Mandatory)

Note the difference between coordinate systems. Matlab considers image coordinates to start from the corner of the image and from (1,1), whereas the eqs derived from figure 6 assume origin (0,0) at the centre of the image.

1. Open a distance map from the depth camera, and a corresponding 2D color frame from the center camera, and show them.
2. Compute the global (x,y,z)-coordinates for each pixel in the frame of the PMD distance map using values from the calibration parameters. Easiest way to do this for all pixels is to list all coordinates contained by the image using the meshgrid function. With some thought, all operations can be applied to the whole list of coordinates at once via matrix multiplication, but going through the coordinates in a for loop is also fine.

   Find from Figure 4 the relation between the PMD measurement *m* and the z-coordinate, and solve for *z*. Hint: form similar triangles with *z* and *f* respectively as the matching sides. **This part**

is not done for data captured by Kinect, as the driver already returns the depth values as Z instead of m

Use Equation 1 to convert (u,v)-coordinates to (x,y).

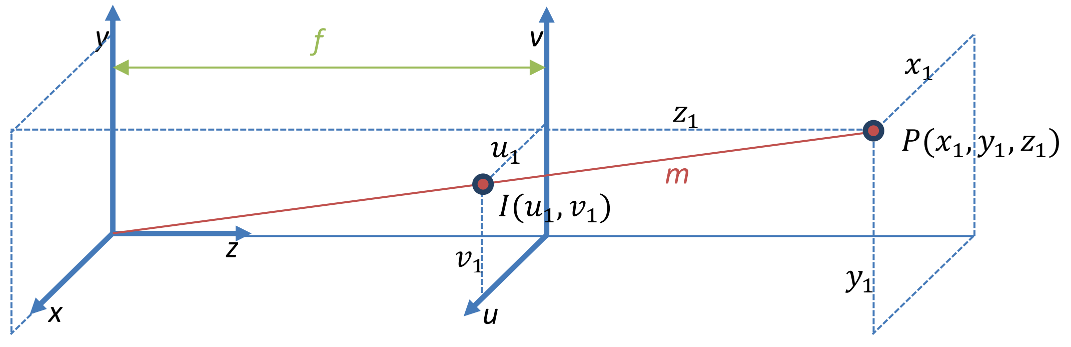


Figure 4. Coordinate space relations in three dimensions, where *m* is the measurement given by the range camera, i.e. value of the depth map

Visualize the global coordinate point cloud by using Matlab's scatter*3* and giving the depth values as the color information for scatter. Hint: `set(gca,'YDir','reverse');`

The background in the synthetic data set is a plane. It should therefore show up as a plane also in the point cloud. If it shows up curved in the visualization, please go back and check your equations. Further processing will not go as intended if this is wrong.

H2. Projecting 3D PMD data to the 2D color camera plane (Mandatory)

1. Apply the changes to the coordinates with R and T, moving them from the coordinate system centered on the PMD camera to the color camera.
2. Project the points from 3D space (x,y,z) to the image plane (u, v) of the color camera using Eq.2.
3. Visualize the projected data. Irregularly sampled data should be shown using e.g. scatter, tools such as imshow will not work. Use the new z-coordinates after rotation and translation as the color information for the scatter plot. The result should look like a depth map from the point of view of the color camera.

H3. Re-sampling the projected data (Mandatory)

The projected data is irregularly sampled, so it needs to be resampled into a regular grid so it aligns with the pixel structure in the other image. Use the new z-coordinates you got after applying the rotation and translation as the z-value, and the projected, irregular sample locations as u and v. Use nearest neighbor interpolation. Hint: *scatteredInterpolant*

Plot the resulting images (original color image, resampled depth image, both overlaid) in a single plot. Check that they actually align. Some artifacts will occur due to the occlusions on the edges. Hint: *subplot*, *imshowpair*

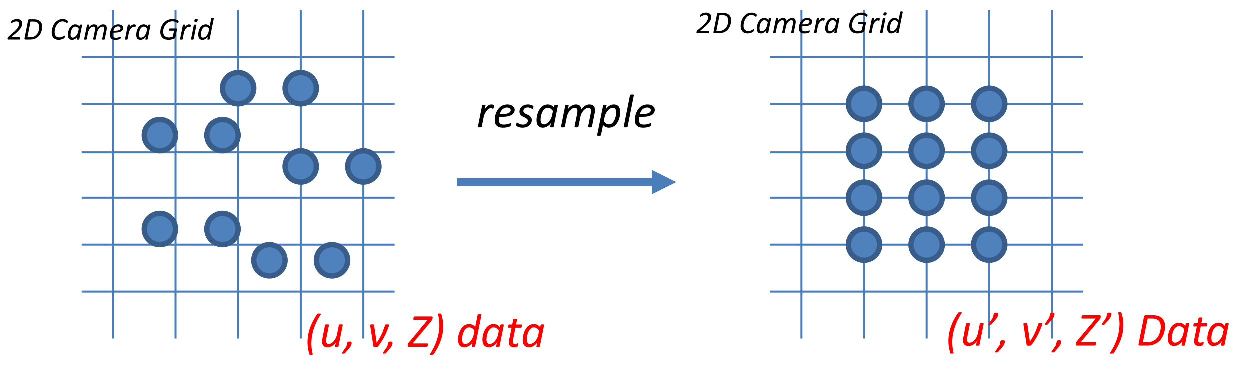


Figure 5. Resampling process

H4. Visualizing the combined depth/color data (Mandatory)

Use the *surf* tool to visualize a textured surface using the projected, re-sampled data from the previous task. Use the original color image as the texture for the model. Hide the surface mesh edges and reverse the z-direction of the plot.

H5. Mapping 2D camera color data to PMD image plane and visualizing (+1)

As an alternative to reprojecting the depth information to the color image plane, the color plane can be sampled to give color to the points of the depth map. This avoids the irregular to regular resampling, but at the cost being limited to the resolution of the depth map (which usually is of lower resolution). The mapping $(u_{depth}, v_{depth}) \rightarrow (x, y, z) \rightarrow (u_{color}, v_{color})$ between the image plane points remains the same from the previous tasks, now you have to follow the relation to the opposite direction to interpolate a color value from (u_{color}, v_{color}) Place the interpolated color value to the corresponding *(u,v)* –location in the depth image. The result should be a regular grid of color values, which aligns with the depth map. Hint: *interp2*

Plot the resulting images (resampled color image, original depth image, both overlaid) side-by-side with the corresponding images from H3.

H6. Removing 3D model edge artifacts (+1)

Continue working on the 3D model from task 4. The 3D model has long, stretched edges at the object boundaries which are due to the discontinuities in the depth. Look at the FaceNormals property of the 3D model you created. Find normals which have a larger magnitude in the horizontal/vertical directions than in the third dimension, meaning the faces are not oriented along the surface a depth map can be expected to cover. Assign the CData property of the 3D model for those faces to NaN, which means they are not drawn. Note that some edges will still remain unless Z-buffering is implemented. You may consider wrapping this into a function which takes the handle to the 3D model as input and makes the changes, otherwise you'll have to copy&paste it to every model drawing. Hint: *h* = *surf*(…); *h.FaceNormals* = …

H7. Z-buffering (+1)

When resampling the images with the projected coordinates, Matlab doesn't take into account the relative depth-wise positioning of pixels. I.e., pixels of the background might be drawn on top of the foreground objects, creating artifacts in the projected depth. Find a way to make sure pixels in the projected depth map get drawn in the correct order, so that foreground pixels are not occluded by the background. After cleaning the input coordinates from the overlaps, run the resampling of H3 again. Also draw again the output of task 4 using the z-buffered depth map.

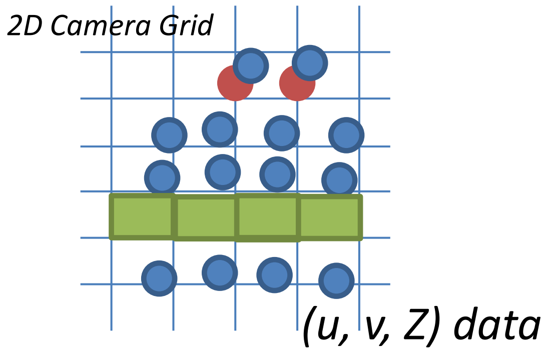
Hint: Analyze the data when you have the *(u,v)*-coordinates and the corresponding depth values in H2. Look for points that have the same (or similar) values of *(u,v)* and replace or remove the points which are behind with the values from the point in front. You may consider using e.g. the knnsearch function.

H8. Occlusion handling (+1)

Task 7 took care of the areas where data from the depth overlaps with itself. The remaining issue is the areas where there is no information at all (disocclusions).

Areas in the image without content can be identified by looking at all of the integer pixel locations (u,v) on the color camera image plane, and identifying those locations which do not have a point projected near (closer than ~1 pixel) them in H2. Hint: *knnsearch*

One possibility to fill the missing pixels is to use prior knowledge about the scene. In the synthetic data, the background is a plane. Fit a plane to the data when it is in the (u,v,z) form from H2. Fill out the disoccluded areas with depth values on the plane by applying the plane equation (au+bv+cz+d=0) by solving for z for each of the locations missing information. Hint: *pcfitplane*



Red – overlapping pixels removed by z-buffering
Green – empty pixels filled by occlusion handling