COMP.SGN.320.2022-2023 3D and Virtual Reality

Calibration of a multi-camera system and 3D data fusion

1 General Instructions

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

Tasks

The assignment consists of several tasks related to calibration and re-sampling of data captured through a multi-camera setup with a range sensor device. It includes home and laboratory assignments. The capturing of real data and camera calibration are performed in the laboratory. The remaining tasks can be solved at home.

Laboratory access

Any student enrolled in the course should have access to the laboratory **TE407**. Please, contact the teacher assistant if you do not have access to it.

Laboratory equipment

In the laboratory, students can find a checkerboard (calibration object), a multi-camera setup with a range capturing device (Microsoft Kinect 2.0) and a desktop PC. This equipment is necessary to capture real data and perform camera calibration.

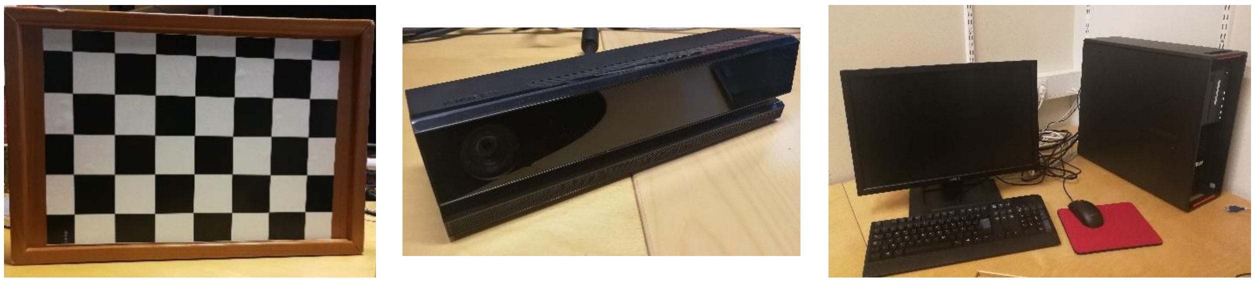


Figure 1 – Laboratory equipment: checkerboard (left), Microsoft Kinect 2.0 (center), and desktop PC (right)

Deliverables

Each group should return:

1 – **Matlab script** that does the required work. When the script is executed in the directory containing the captured data and synthetic 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. Please, **list clearly which steps you have completed** in the comments, at the beginning of the main Matlab script.

2 – **Report** with names and student numbers of the group members, a clear list of tasks the group has completed and all requested images.

The submission must contain the **PDF document**, all **Matlab scripts**, **image data**, and **calibration data** required by your scripts, everything within 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.

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 (orientation and location) of cameras in a system with respect to the real world.

For the purposes of this laboratory work, a multi-camera system with a range sensing device is used. A range sensor provides direct 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 2.0 (Figure 2), a game controller for the Xbox One gaming console.

Figure 2 – Example of a time-of-flight range sensor - Microsoft Kinect 2.0

2.1 Multi-camera systems

A multi-camera system consists of at least two (stereo) or more cameras. Such systems can also be assisted by range sensor(s), as shown in Figure 3. 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 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.

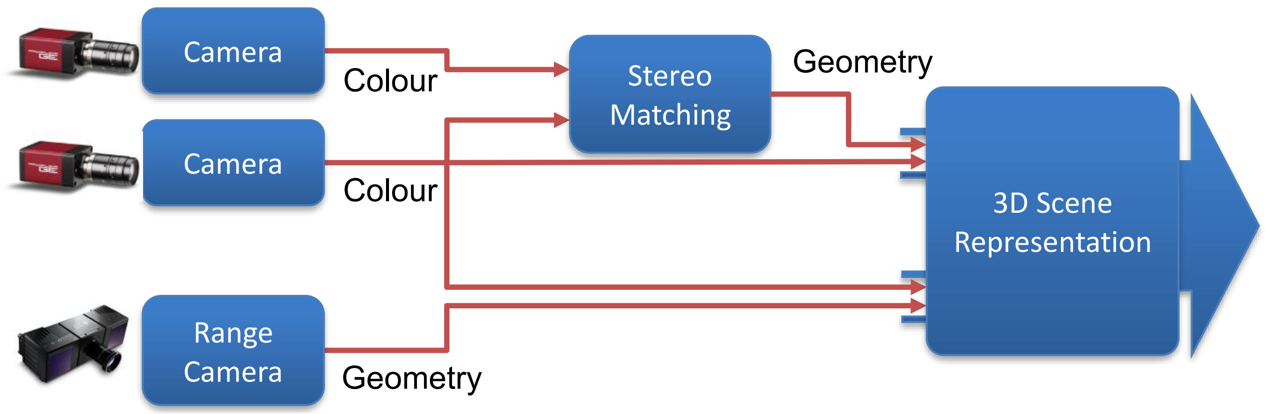


Figure 3 – Example diagram of a multi-camera system and the alternative ways of reaching a similar 3D scene representation

2.2 Pinhole camera model

The pinhole camera model describes how points in physical space 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

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

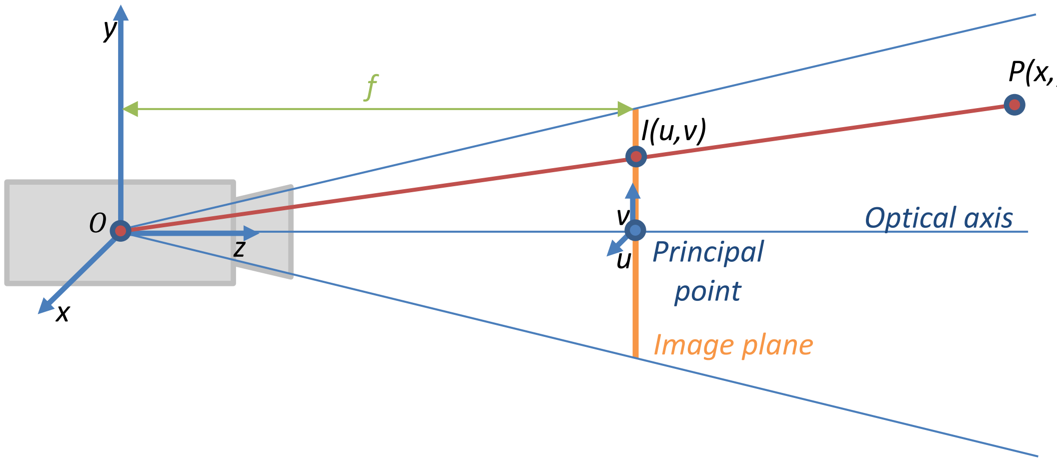


Figure 4 – Pinhole camera model

However, Eq. 1 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 rotation of the camera axes and translation 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:

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\varphi) & -sin(\varphi) \\ 0 & sin(\varphi) & cos(\varphi) \end{bmatrix}, R_y(\Psi) = \begin{bmatrix} cos(\Psi) & 0 & sin(\Psi) \\ 0 & 1 & 0 \\ -sin(\Psi) & 0 & cos(\Psi) \end{bmatrix}, R_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad 3$$

which can be combined as a single rotation matrix,

$$R(\varphi, \Psi, \theta\,) = R_x(\varphi)\, R_y(\Psi)\, R_z(\theta), \quad 4$$

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,

$$T = O_R - O_L = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_R - \begin{bmatrix} x \\ y \\ z \end{bmatrix}_L, \quad 5$$

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. It describes completely how a point with position (x, y, z) in the physical world is captured by a camera setup,

$$P = M_{3x3}[R_{3\times3} \;\; | \;\; T_{3\times1}] = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix},$$

where $[R_{3\times3} \;\; | \;\; T_{3\times1}]$ means concatenation of matrices (extrinsic parameters), and M_{3x3} is the calibration matrix (intrinsic parameters).

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

$$\begin{bmatrix} u' \\ v' \\ k \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \qquad (u, v) = (u'/k, v'/k). \tag{7}$$

3 Exercise tasks

In this assignment, you are asked to complete both, homework tasks and laboratory tasks. Both are done by the group independently. The laboratory tasks are performed in the laboratory and they are all mandatory. Completing all mandatory tasks will result in grade 1. Any additional task completed will increase the grade by one. Additional tasks do not have to be completed in numerical order.

3.1 Laboratory tasks

The following steps should be completed during the laboratory session. Before starting, get familiar with the calibration toolbox here. Specifically, try to understand the main stages behind the calibration toolbox, the (stereo) calibration procedure, and how to interpret the calibration results:

1. Estimation of the intrinsic and extrinsic parameters of each camera (here)
2. Estimation of the relative pose between cameras (stereo calibration) (here)
3. Description of the calibration parameters (here)

L1. Collecting calibration data (Mandatory)

Microsoft Kinect 2.0 provides three synchronized output images: 2D colour image, (near-)infrared (IR) image, and depth image. Only 2D colour images and IR images are used in this particular calibration procedure. In general, depth calibration is also required when dealing with range sensors. For simplicity, we will assume in this exercise that depth information obtained from the range sensor is accurate.

This specific calibration toolbox requires images of a planar checkerboard. Thus, the first goal is to capture different poses of the checkerboard. However, there are a few details to keep in mind during the capturing phase:

1. Since one of our goals is to estimate the relative pose between cameras, the checkerboard must be visible (within the same field-of-view) by the 2D colour camera and the range sensor at the same time;
2. Do not obstruct the checkerboard with hands or other objects while capturing;
3. Due to the low-resolution of the range sensor, the checkerboard should be placed as close as possible to the sensor. Avoid any light obstruction or image saturation. Placing the checkerboard closer to the sensor helps the corner detection algorithm and leads to more accurate calibration results;
4. While capturing, Kinect must stay steady. Do not touch it at any circumstance during the capturing phase since this would lead to wrong calibration parameters.

Open the file called ***CaptureCalibData.m***. Inside there are several parameters that can be changed in order to capture the calibration data. You may need to capture a dozen of pictures because not every image has a positive contribution to the estimation of the calibration parameters.

L2. Extracting the calibration parameters of the camera system (Mandatory)

Start by copying all 2D colour images and respective IR images to the same directory than the file ***Calibrate.m***. Open and run ***Calibrate.m*** for each camera sensor and only then run the stereo calibration. Please consider the following information before starting the calibration procedure:

1. For the calibration of each camera, the overall reprojection/pixel error should be as low as possible (less than 0,2 pixels for the 2D colour camera and less than 0,4 pixels for the range sensor). If it exceeds these values, you may consider discarding some calibration images (with large pixel/reprojection error) or capture new ones;
2. Do not forget to save the calibration parameters (*Save*) after reading the images, extracting the grid corners and estimating the calibration parameters;
3. Save each individual calibration file relative to 2D colour camera and range sensor with the names "***Calib_Results_right***" and "***Calib_Results_left***", respectively;
4. After the calibration, you may try to undistort the calibration images (***Undistort image*** option in ***calib_gui***) to verify whether straight lines in real world remain straight in the undistorted image.

After calibrating each individual camera sensor, run the stereo calibration. Once completed, do not forget to save the results (by default it saves with the name "***Calib_Results_stereo***"). The 2D colour camera and the range sensor are parallel relative to each other. The origin of the coordinate system in this stereo calibration is located at the center of the range sensor ($X = 0$, $Y = 0$, $Z = 0$) as shown in Figure 5. Both, 2D colour camera and IR camera have almost the same orientation and they are about 50 mm apart along X-axis. Make sure that the stereo calibration results make sense.

Finally, run the last instruction inside ***Calibrate.m*** to create a clean structure containing the intrinsics and extrinsics parameters of Kinect 2.0. This file, named ***KinectData.mat*** by default, will be used in future tasks.

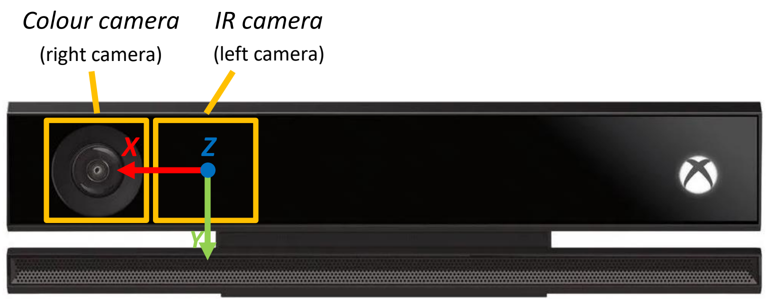


Figure 5 – Microsoft Kinect 2.0 coordinate system

L3. Capturing a set of still images (Mandatory)

After calibrating Kinect 2.0, it is time to capture new images (a 2D colour image with respective depth image) for the homework tasks. Capture something static, with simple geometry (e.g., a book) and with a flat background (e.g., a wall). Avoid transparent objects or any material that may absorb IR light greatly as this degrades the depth quality. In addition, make sure the scene contains at least two different depth layers, i.e., a close object and a far object. Figure 6 shows an example of this kind of scene.

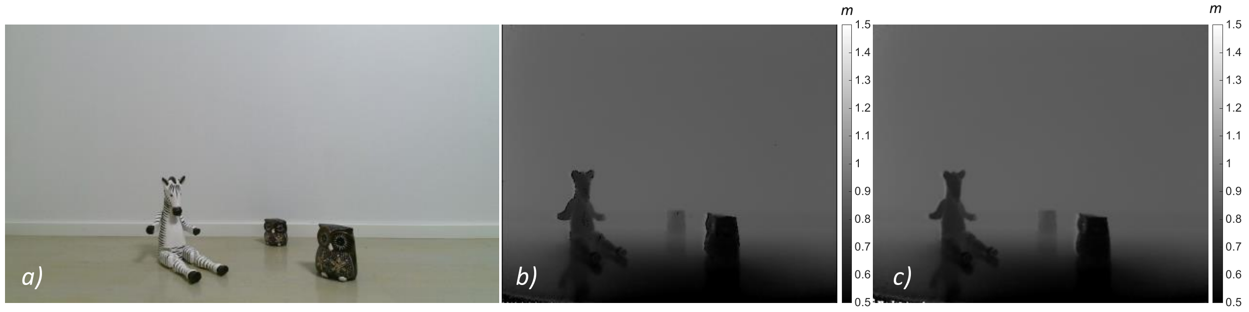


Figure 6 – Example of a captured scene with Kinect 2.0: a) colour image; b) depth image; c) inpainted depth image

Note: The depth image obtained from Kinect 2.0 contains measurement errors or unknown depth values. They are assigned with the depth value 0. Keep in mind that the effective range of Kinect 2.0 is between 0,5 m and 4,5 m. This means that your scene should be within these values. Feel free to clean up the errors by using a third-party inpainting algorithm such as: https://se.mathworks.com/matlabcentral/fileexchange/4551-inpaint_nans.

Start by running the script ***CaptureScene.m***. The script will save a 2D colour image (***Colour.tif***) and respective depth image (***Depth.tif***) every time you press "s" key.

Once you are satisfied with the captured images, run the script ***UndistortImages.m*** to undistort both 2D colour image and depth image. Here, we are using the calibration results obtained in the previous task and the calibration toolbox option ***Undistort Image***. The undistorted images are generated with the name "****_rect.tif***" by default. Use the undistorted images ***Colour_rect.tif*** and ***Depth_rect.tif***, and ***KinectData.mat*** for the homework tasks.

3.2 Homework tasks

All results here should be drawn when the Matlab script (***LW2.m***) is executed. The homework tasks should work for both synthetic data and real (Kinect 2.0) data except for the last task (H8).

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

Please, note the difference between coordinate systems. Matlab considers image coordinates to start from the corner of the image and from (1,1), whereas the equations derived from Figure 4 assume origin (0,0) at the center of the image.

1. Start by showing the 2D colour image and respective depth data.
2. Compute the global (x,y,z)-coordinates for each pixel of the depth map by using its depth values and the calibration parameters. Easiest way to do this for all pixels is to:
    a. 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.
    b. Find from Figure 7 the relation between the depth measurement *m* and the z-coordinate, and solve for *z*. Hint: form similar triangles with *z* and *f* respectively as the

matching sides. Note the units of the parameters – the *f* is given in meters and has to be converted to pixels using the pixel size, so that it matches with the other units. **This part is not done for data captured by Kinect, as the driver already returns the depth values as *z* instead of *m*.**

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

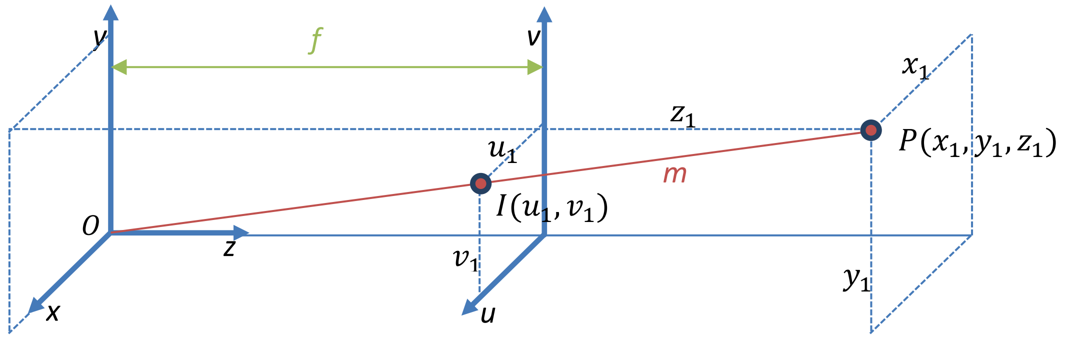


Figure 7 – Coordinate space relations in three dimensions, where *m* is the measurement given by the range camera, i.e. value of the depth map. It measures the distance starting from the optical center (entire red line), not from the image plane

3. Visualize the global coordinate point cloud by using Matlab's scatter*3* and giving the depth values as the colour 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.

H2. Projecting 3D depth data to the 2D colour camera plane (Mandatory)

1. Apply the changes to the coordinates with R and T, moving them from the coordinate system centered at the depth camera to the colour camera.
2. Project the points from 3D space (x,y,z) to the image plane (u,v) of the colour camera using **Equation 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 colour information for the scatter plot. The result should look like a depth map from the point of view of the colour camera.

H3. Re-sampling the projected data (Mandatory)

The projected data is irregularly sampled. Therefore, it needs to be resampled into a regular grid to match the pixel structure of the colour image. Use the new z-coordinates you got after applying the rotation and translation as the z-value, and the projected (irregular) sample locations u and v. Apply nearest neighbour interpolation. Hint: *scatteredInterpolant*

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

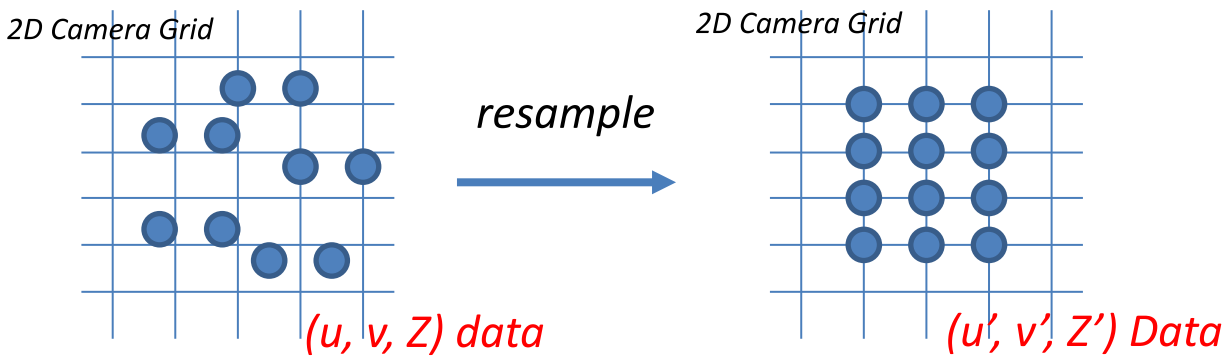


Figure 8 – Resampling process

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

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

H5. 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 due to the discontinuities in the depth. Look at the *FaceNormals* property of the 3D model you have created. Find normals which have a larger magnitude in the horizontal/vertical directions than in the third dimension. 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 will have to copy and paste it to every model drawing. Hint: *h = surf(…); h.FaceNormals = …*

H6. Mapping 2D camera colour data to depth image plane and visualizing (+1)

As an alternative to reprojecting the depth information to the colour image plane, the colour plane can be sampled to give colour 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 colour value from (u_{color}, v_{color}) Place the interpolated colour value to the corresponding *(u,v)* – location in the depth image. The result should be a regular grid of colour values, which aligns with the depth map. Hint: *interp2*

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

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.

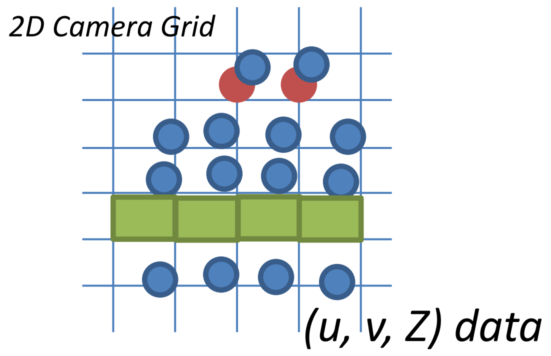
Hint: Analyse the data when you have the *(u,v)*-coordinates and the corresponding depth values in H2. Look for points that have the same (or similar) values of *(u,v)* and replace or remove the points which are behind with the values from the point in front. You may consider using e.g. the *knnsearch*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