1130-EMARO-MSA-1006 # Computer Vision

T5: RGB-D registration (1)

The goal

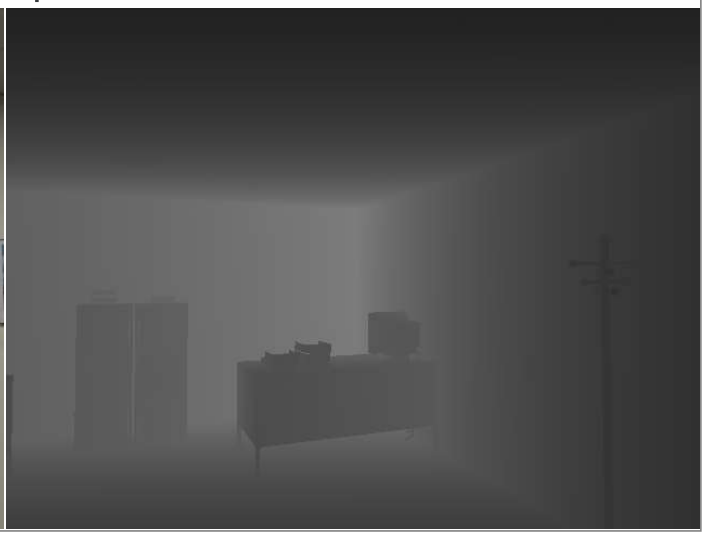
Your goal in this double-part exercise is to calculate camera trajectory having a set of RGB images with depth information added. The process should be composed of two main steps:

1. initial sparse transformation estimation between two frames (features + RanSaC),
2. dense pairwise transformation refinement (ICP).

Input data

You will be provided with a set of artificially generated image sequences. The sequence contains RGB images with aligned, perfect depth. The format of the images is the following:

- color image: RGB with 8 bits per pixel,
- depth image: 16-bit depth values, with scale 5000 per 1m (i.e. value 1000 is 0.2 m, 8000 is 1.6 m etc.).

RGB	Depth
	

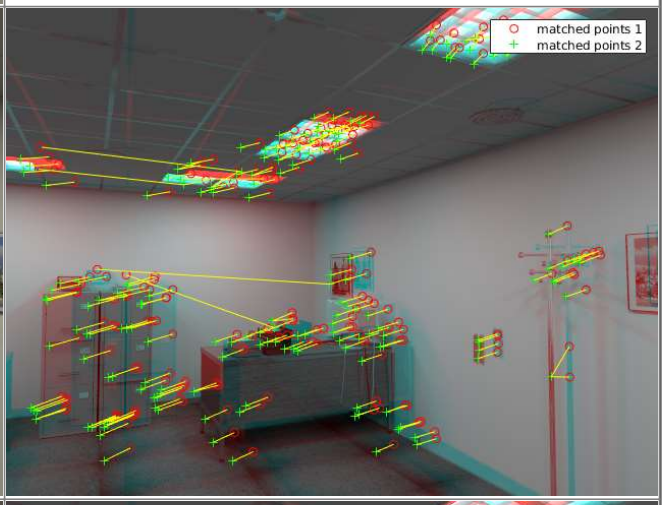
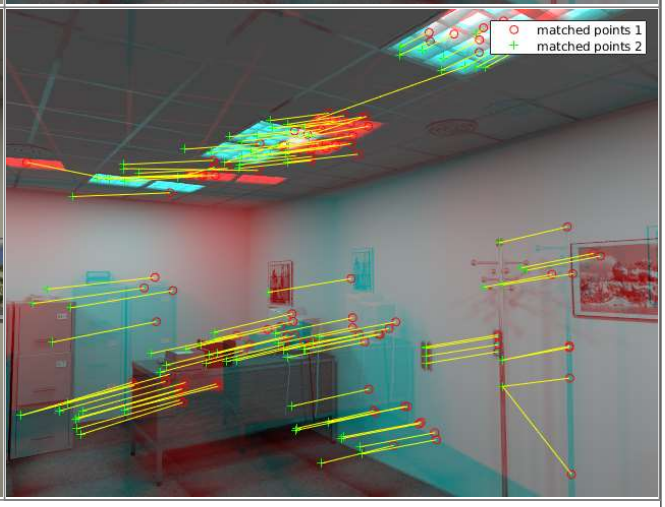
Data comes from the ICL NUIM dataset, and the selected sequence is **of-kt2**.

The original sequence was rendered with 30 frames per second, but for this task, it was downsampled to 3 Hz, i.e., only every 10-nth frame was selected.

points using the depth image and the camera parameters. When two sets of matching 3D points are available you have to calculate the transformation between them. After this step, you should have an initial camera pose for each frame calculated.

Detecting and matching the features

As the exemplary feature detection and matching SURF was selected. After calculating the features for the reference frame (100.png) features for three other frames were calculated and matched. Matches are shown as yellow lines on the picture.

Sample	Image	Matches
100		None (reference image)
110		
130		

170

Sample code to detect and match features between two frames is given below. Please remember, that in the final solution you should calculate the features for each frame once and store them so that you can use them later without recalculating.

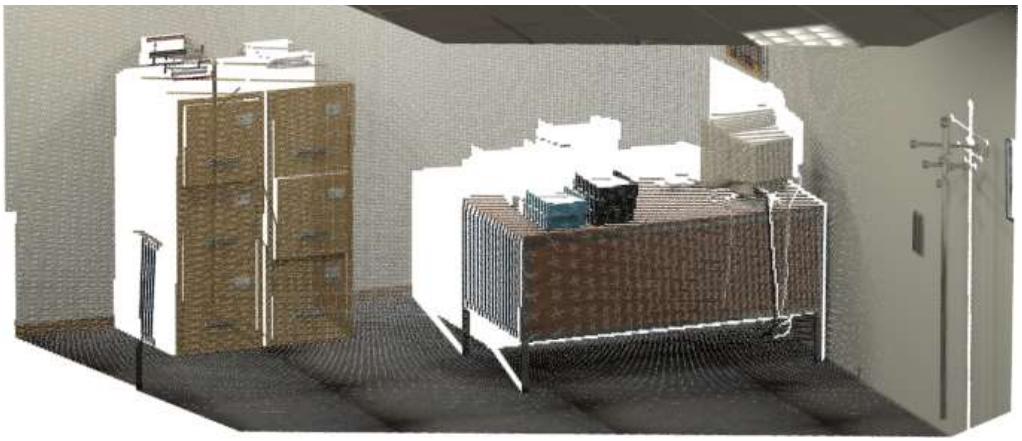
```
% Read the two images.
I1 = rgb2gray(imread('rgb/0.png'));
I2 = rgb2gray(imread('rgb/10.png'));

% Find the SURF features. MetricThreshold controls the number of detected
% points. To get more points make the threshold lower.
points1 = detectSURFFeatures(I1, 'MetricThreshold', 200);
points2 = detectSURFFeatures(I2, 'MetricThreshold', 200);

% Extract the features.
[f1,vpts1] = extractFeatures(I1, points1);
[f2,vpts2] = extractFeatures(I2, points2);

% Match points and retrieve the locations of matched points.
indexPairs = matchFeatures(f1,f2) ;
matchedPoints1 = vpts1(indexPairs(:,1));
matchedPoints2 = vpts2(indexPairs(:,2));

% Display the matching points. The data still includes several outliers,
% but you can see the effects of rotation and scaling on the display of
% matched features.
figure; showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
legend('matched points 1','matched points 2');
```

3D coordinates calculation

From the feature position in the image (px, py), having the depth value for a given pixel (d) you should calculate real-world coordinates for the interesting feature points. To do this you need camera parameters:

$$fx = 481.2,\ fy = 480.0,\ cx = 319.5,\ cy = 239.5$$

As the data is perfect (generated from the 3D model), it has no distortion. Also, the depth data is dense, so for every point in the color image, there is a depth value available. Sample full 3D view of the selected frame (100.png) looks like this:

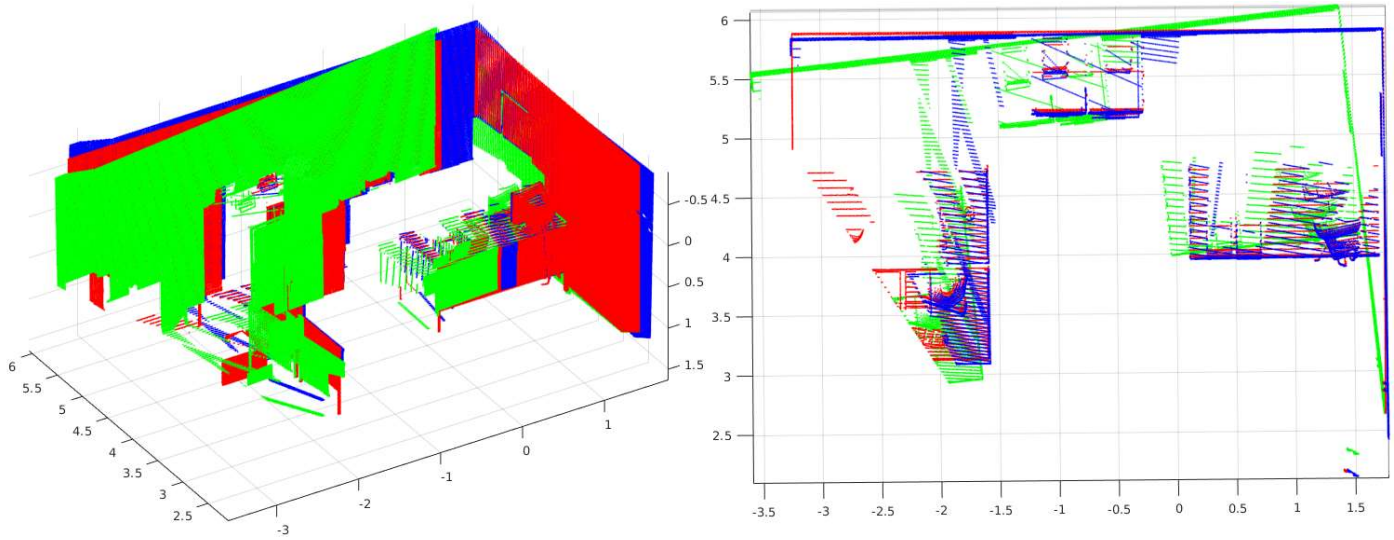


For this job implement the function with the given prototype:

```
function XYZ = reproject(depth, scale, fx, fy, cx, cy)
% calculate XYZ matrix based on a given depth matrix
% XYZ should have the same width and height as depth with three planes (x, y and z)
% scale: number of units in depth corresponding to one meter
% fx, fy: focal length
% cx, cy: principal point
end
```

Transformation estimation

Based on two sets of matching 3D point sets you have to calculate the initial transformation. Implement the RanSaC algorithm to find the 3D transformation between the points. Please prepare it in a way such that it is possible to change the RanSaC parameters: number of iterations, inliers ratio, and reprojection threshold.



On the image above, the red cloud is frame 0 (reference) and the green is frame 50. After calculating the best initial transformation the green cloud is transformed into the blue, which is more or less overlapping with the red one. For visualization purposes floor and ceiling were removed.

As a reference for estimating the transformation between a subset of points please look at the SVD approach (by Olga Sorkine-Hornung and Michael Rabinovich from the Department of Computer Science, ETH Zurich).

For this job implement the functions with the prototypes given below:

```
function [R, T] = estimateTransform(p1, p2)
% estimate rotation matrix R and translation vector T such that
% p1 * R + T = p2 (if possible)
end
```

```
function [R, T] = ransacTransform(pts1, pts2, iter, ratio, thr)
% estimate the best transform R, T between the two given point sets, such that
% pts1 * R + T are as close to pts2 as possible
```


Project progression

1. Data loading and depth to pointcloud conversion: 1 pt
2. 3-point transformation estimation: 3 pts
3. RanSaC implementation: 3 pts
4. Transformation estimation between the pairs of images: 1 pts

code.zip	10 stycznia 2022, 21:00
ecovi_t5.zip	15 grudnia 2020, 08:27