UNIVERSITY OF AVEIRO
DEPARTAMENT OF ELECTRONICS, TELECOMUNICATIONS AND INFORMATICS

Computer Vision

Lecture 07

3D Vision

- Disparity map
- Dense mapping
- 3D reconstruction
- Visualization and manipulation of 3D cloud of points in pcl
- Co-registration of cloud of points using ICP in pcl

1. Disparity map

Recover the code from the exercise on image rectification (exercise 5 in last lecture - in alternative you might use the available code in reconstruct.cpp). Use the class `StereoBM` and the function that implements a block matching techniques (template matching will be explored later within this Computer Vision course) to find correspondences over two rectified stereo images. Use the parameters specified as follow since we will not enter in details of these functions.

```
// 1- Variable definition
// the preset has to do with the system configuration (basic, fisheye, etc.)
// ndisparities is the size of disparity range,
// in which the optimal disparity at each pixel is searched for.
// SADWindowSize is the size of averaging window used to match pixel blocks
// (larger values mean better robustness to noise, but yield blurry disparity maps)
int ndisparities = 16*5;
int SADWindowSize = 21;

cv::Mat imgDisparity8U;
cv::Mat imgDisparity16S;

//-- 2. Call the constructor for StereoBM
cv::Ptr<cv::StereoBM> sbm;
sbm = cv::StereoBM::create( ndisparities, SADWindowSize );

//-- 3. Calculate the disparity image
sbm->compute( remap_imgl, remap_imgr, imgDisparity16S );
```

```
//-- Check its extreme values
double minVal; double maxVal;

cv::minMaxLoc( imgDisparity16S, &minVal, &maxVal );
printf("Min disp: %f Max value: %f \n", minVal, maxVal);

//-- 4. Display it as a CV_8UC1 image
//Display disparity as a CV_8UC1 image
// the disparity will be 16-bit signed (fixed-point) or
//32-bit floating-point image of the same size as left.
imgDisparity16S.convertTo( imgDisparity8U, CV_8UC1, 255/(maxVal - minVal));
namedWindow( "disparity", cv::WINDOW_NORMAL );
cv::imshow( "disparity", imgDisparity8U );
```

2. 3D Reconstruction

Use the function `cvReprojectImageTo3D` to compute the 3D coordinates of the pixels in the disparity map. The parameters of `cvReprojectImageTo3D` are the disparity map (`disp` in previous exercise), and the matrix Q given by the function `cvStereoRectify`. Save the 3D coordinates in an xml file using the function `FileStorage`.

3. Visualization of point cloud in pcl

Modify the source code `viewcloud.cpp` to read the 3D points of the file you have saved in the previous section and visualize the results of the 3D reconstruction.

Assignment to the pointCloud (cloud in the example below) can be performed using the following code (or similar):

```
int p=0;
for (int i=0;i< (*cloud).height; i++)
for (int j=0;j< (*cloud).width; j++)
{
(*cloud).points[p].x = pos3D.at<cv::Vec3f>(i,j)[0];
(*cloud).points[p].y = pos3D.at<cv::Vec3f>(i,j)[1];
(*cloud).points[p].z = pos3D.at<cv::Vec3f>(i,j)[2];
p++;
}
```

Visualize the 3 points and add any filtering necessary to avoid visualization of not well reconstructed 3d Points.

4. PCD (point cloud data) 3D format

Modify the source code viewcloud.cpp to read and visualize the two provided kinect images `filt_office1.pcd` and `filt_office2.pcd` The Point Cloud Data file format (PCD) used is the 3D file format from PCL and can be written and read directly using the PCL functions `loadPCDfile` and `savePCDFileASCII`. As shown in the following sample code.

```
pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud1 (new pcl::PointCloud<pcl::PointXYZRGB>);
if (pcl::io::loadPCDFile<pcl::PointXYZRGB> (".//depth_images//filt_office1.pcd", *cloud) ==
-1) //* load the file
  {
    PCL_ERROR ("Couldn't read file filt_office1.pcd \n");
    return (-1);
  }
```

Note:

By default, kinect sensor returns `NaN` values that may cause problems in the processing, to remove `NaN` values and at the same time, down sample (reduce the number of points in the file) using the

VoxelGrid filter with a grid size of 0.05 in each direction. The filt_office1.pcd and filt_office2.pcd files have already been treated with this filter resulting down sampled cloud of points (original Kinect images are in files office1.pcd and office2.pcd).

5. ICP alignment

Use the pcl::IterativeClosestPoint function to align the given down sampled cloud of points. Note that the values of the termination criteria are very sensitive and should be adapted for each case. Normally an initial rough registration should also be provided to avoid bad registration However in this case, and given the proximity of the provided depth images this should not be necessary.

Run the ICP algorithm with the following parameters:

```
   pcl::PointCloud<pcl::PointXYZRGB>::Ptr aligned_cloud (new
pcl::PointCloud<pcl::PointXYZRGB>);

   pcl::IterativeClosestPoint<pcl::PointXYZRGB, pcl::PointXYZRGB> icp;
   icp.setTransformationEpsilon (1e-6);
   icp.setMaxCorrespondenceDistance (0.25);
   icp.setMaximumIterations (50);

   icp.setInputCloud (cloud2);
   icp.setInputTarget (cloud1);

   icp.align (*aligned_cloud);
```

Visualize in the same window the original and the aligned cloud of points. Modify the ICP parameters to check the quality of the registration (for example, comment the lines and use the default values and evaluate the results).

Note:

The evaluated transform can be recovered with the function

```
Ti=icp.getFinalTransformation();
```

This transformation might be used with the function pcl::transformPointCloud to align the registered cloud of points in the new coordinate reference.

6. Depth acquisition (optional / Homework)

Modify the code from pcl_io.cpp to save directly a pcd file from a Kinect sensor. Use this code to acquire some kinect depth image from close viewpoints and try to register your images adapting the previous code.

The code pcl_io.cpp uses the openni driver that should have been installed with the pcl library. However in case you have problems using this driver you might use another driver (libfreenect) and use the available code simpleFreenect to perform the Kinect acquisition.

7. Report

Continue the report started in the last lecture. It should contain an example of the images displayed in each exercise, as well as your comments about them. You may also use the second set of stereo images as example in your report (StereoL and StereoR images). You should also provide the values of the several matrices obtained along the class. Optional/homework parts should appear in your final report. Use the code pcl_io or pcl_io_freenect to acquire your own Kinect images and try to present the results for section 4 and 5 using your own acquired images.

A Kinect will be available at the entrance of the DETI during next two weeks for use by the students. Any problem, contact paulo.dias@ua.pt.