最后修改于 2年 前

Stereovision reconstruction system

This project focuses on a method of 3D reconstruction of the scene by processing the high resolution video data generated by a stereo camera. Currently it is based on methods described in:

- **Learning OpenCV: Computer Vision with the OpenCV Library by Gary Bradski and Adrian Kaehler, Published by O'Reilly Media, October 3, 2008.**
- **Andreas Klaus, Mario Sormann and Konrad Karner "Segment-Based Stereo Matching Using Belief Propagation and a Self-Adapting Dissimilarity Measure"** [Link]

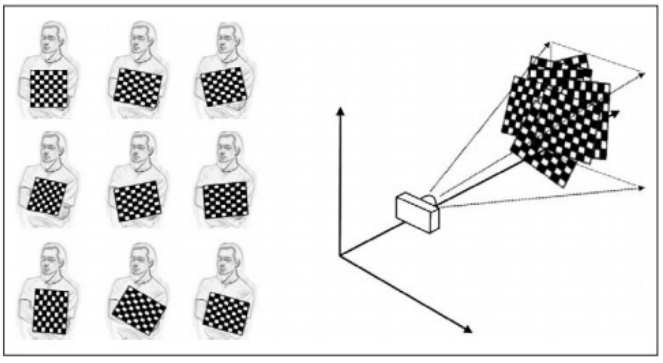
Approach

Implemented solution base on segment-based stereo matching idea. When there is no camera calibration parameters extracted, the first step is calibration process. It base on proper camera calibration data. The result of this process is two files with the camera parameters. When we already have the data to camera calibration, the rest of the process takes place in the data stream from the stereo camera. First, frame image from left data stream is segmented by color using pyramid segmentation algorithm. Next, using both left and right data stream we obtain depth map using chosen stereo matching algorithm. The result of this algorithm contains a lot of noisy data, so we use the previously obtained segmented image to determine which parts of the scene are fragments of depth map data. Assuming that the points of the segment correspond to a similar depth, we replace the values of this points with mean value of the segment points depth thus getting rid of outliers. If there is more than one pair of left and right image, whole process of creating depth map is repeated. The final depth map is obtained by averaging all of the created depth maps. The last two steps are used to create 3D model of the scene. The final depth map is reprojected to point of 3D cloud and finally the model is builded using triangularization.

Solution steps

0. Stereo calibration and rectification

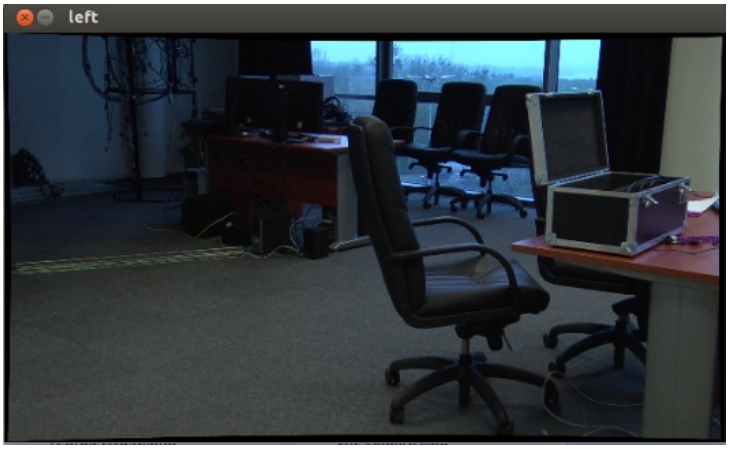
Stereo calibration is the process of computing the geometrical relationship between the two cameras in space. The process of camera calibration gives a model of the camera's geometry and a distortion model of the lens. These two informational models deifne the *intrinsic parameters* of the camera. The method of calibration is to target the camera on a known structure that has many individual and identifiable points. By viewing this structure from a variety of angles, it is possible to compute the relative location and orientation of the camera at the time of each image as well as the intrinsic parameters of the camera. In order to provide multiple views, we rotate and translate the *calibration object* which is used in a flat grid of alternating black and white squares that is usually called a "chessboard" (see Pic 1).



Pic 1. Scheme presenting calibration process using calibration object named chessboard.

There are two things that we can do with a calibrated camera:

- correct for distortion effects; OpenCV provides us with a ready-to-use undistortion algorithm that takes a raw image and the distortion coefficients from and produces a corrected image. The basic method is to compute a distortion map, which is then used to correct the image.
- construct three-dimensional representations of the images camera receives,which is described in section 4: Reprojection to 3D.

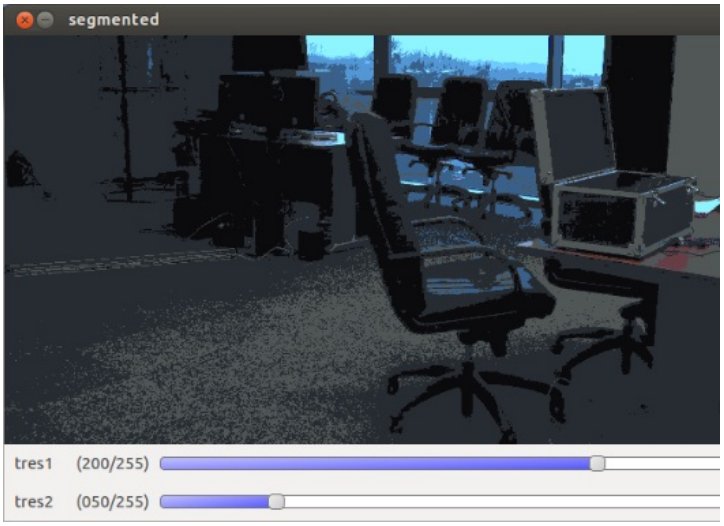


Pic 2. An example of video frame after a distortion reduction process.

The three-dimentional reconstruction is possible to accomplish with a professional three-dimensional camera that allows to change the focal length and to set it to *infinity*. If we cannot influence this property, the process of stereo rectification must be performed. It is the process of "correcting" the individual images so that they appear as if they had been taken by two cameras with row-aligned image planes. With such a rectification, the optical axes (or principal rays) of the two cameras are parallel and so we say that they intersect at infinity. When the image data is undistorted and rectified, it can be used to create depth map.

[See also]

1. Image Segmentation

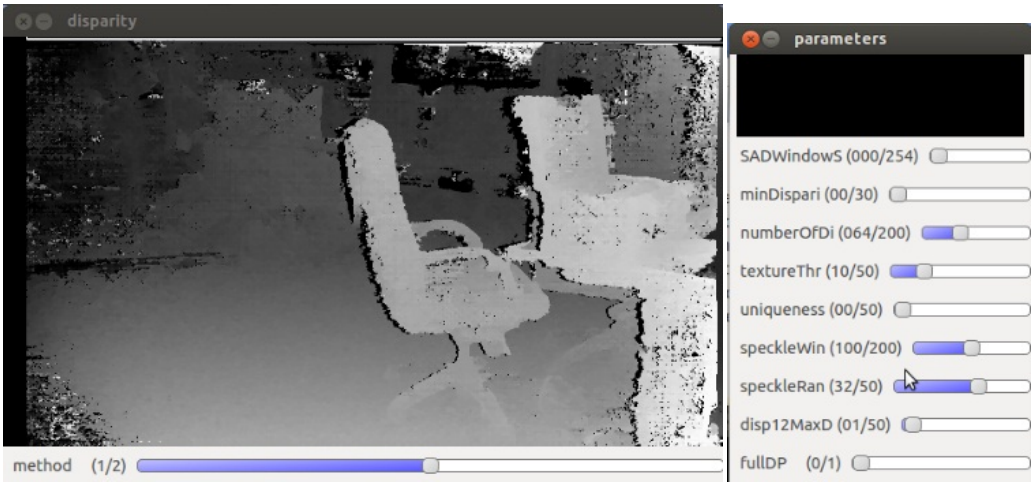


Pic 3. The window showing image after segmentation. There are two parameters to control piramid segmentation quality.

Segment-Based Stereo Matching requires color segmented image of scene which will be used to distinguish the different elements of the scene. For this purpose an *image segmentation by pyramids algorithm* has been used. The result of the algorithm is an image with a limited number of colors. Next, each of the different colors from this image are separated by using a histogram and thresholding process so that for each color we determine its region. Finally, we find each solid segments of the same color by extracting blobs using blob library.

[See also]

3. Segment-Based Stereo Matching

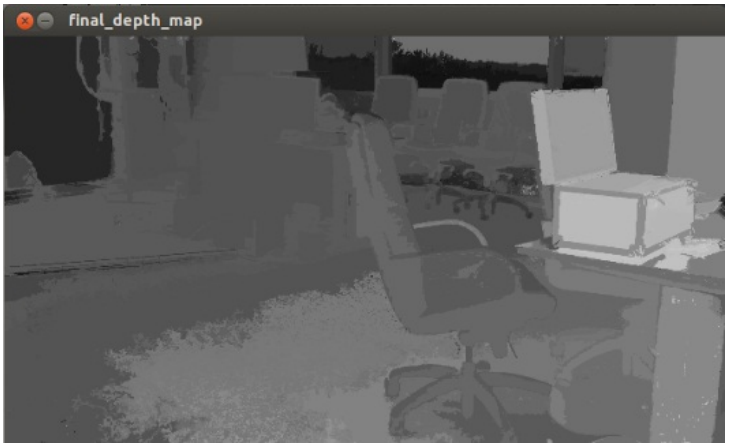


Pic 4 and 5. Window with disparity map obtained from left and right video frame using stereo matching algorithm.

By finding correspondences between points of the left and right images from stereo camera, it is possible to compute the three-dimensional location of the points. This process is named stereo matching. Application allows to use and tests one of the six stereo matching algorithms, three of them uses GPU architecture for computation:

- The Block Matching Stereo Correspondence Algorithm - CPU and GPU versions
- The Belief Propagation Algorithm - GPU version only
- Constant Space Belief Propagation Algorithm - GPU version only
- Semi-Global Block Matching Stereo Correspondence Algorithm - CPU version only
- Variational Matching Algorithm - CPU version only

Algorithm returns a disparity map, on which color corresponds to the apparent pixel difference between a pair of stereo image, so that - the lighter the point, the further it is located. The output map is almost always very noisy, so there is a need to filter out outliers - incorrectly specified points. Assuming that the points of the same color segment correspond to a similar depth, we use the previously obtained segmented image to determine which parts of the scene are fragments of depth map data. Then, we replace the values of this points with *mean value* of the segment points depth thus getting rid of outliers. The steps of color segmentation and segment-based stereo matching (steps 2 and 3) are repeated for every pair of image frames from stereo camera. The final depth map is mean of all generated depth maps. When the data stream from the stereo camera runs out, the process of reprojection is performed.



Pic 6. Final disparity map obtained with separation of the image segments.

[See also]

4. Reprojection to three-dimention

The purpose of this step is to transform the depth map on the cloud of three-dimentional points. This process requires *Q matrix - the perspective transformation matrix*, obtained from the camera calibration. The algorithm of reprojection transforms 1-channel disparity map to 3-channel image representing a 3D surface. That is, for each pixel (x,y) and the corresponding disparity d=disparity(x,y) it computes:

$$[X\ Y\ Z\ W]^T = \mathtt{Q} * [x\ y\ \mathtt{disparity}(x,y)\ 1]^T$$
$$\_\ \mathtt{3dImage}(x,y) = (X/W,\ Y/W,\ Z/W)$$

[See also]

5. Generating three-dimensional model

The last step concerns the generation of 3D models from point clouds. To accomplish this, the process of *triangulation* is performed using algorithms from PCL - the open source Point Cloud Library. There are several parameters of this process that can be defined:

- maximum Nearest Neighbors setMu control the size of the neighborhood. The former defines how many neighbors are searched for, while the latter specifies the maximum acceptable distance for a point to be considered, relative to the distance of the nearest point (in order to adjust to changing densities). Typical values are 50-100 and 2.5-3 (or 1.5 for grids).
- search radius is practically the maximum edge length for every triangle. This has to be set by the user such that to allow for the biggest triangles that should be possible.
- the minimum and maximum angles in each triangle. While the first is not guaranteed, the second is. Typical values are 10 and 120 degrees (in radians);
- maximum surface angle and normal consistency are meant to deal with the cases where there are sharp edges or corners and where two sides of a surface run very close to each other. [See also]

[Example model]

Modules

- **Camera Calibrator** - implements method for stereo camera calibration
- **Depth Substraction** - implements usage of stereo matching methods in order to obtain depth map
- **Stereo Reader** - implements handling of stereo camera stream
- **Stereo Vision** - main class of application, which monitor the process
- **Visualizer** - class implementing methods to reproject depth map to point cloud and saving to file using Point Cloud Library
- **Segmentation** - implements image segmentation method
- **Matching Method** and inheriting methods classes - implement stereo matching methods originally founded in OpenCV library

Requirements

- Boost C++ libraries ≥ 1.46
- Eigen libraries ≥ 3.0
- FLANN libraries ≥ 1.7.1
- VTK libraries ≥ 5.6
- The Point Cloud Library (PCL)
- OpenCV libraries ≥ 2.4.2
- CUDA ≥ 4.0

Installation

```
svn checkout https://apps.man.poznan.pl/svn/stereovision/ stereovision
cd stereovision
./build.sh
```

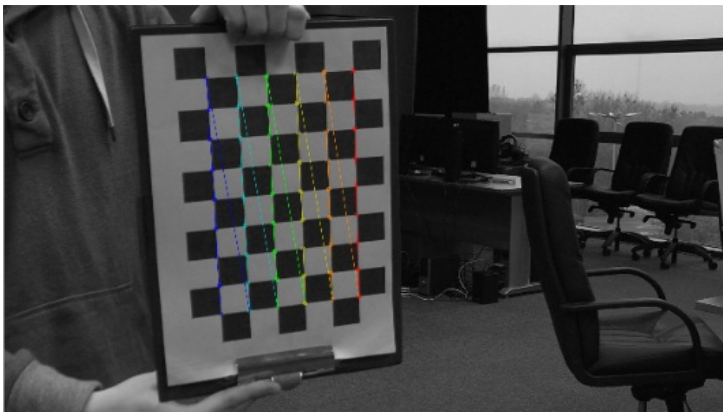
User manual

Camera calibration

To run the application properly, first the calibration proces must be performed. We need a pairs of images from both lenses of the "calibration chessboard" with various views and put

them in a directory: "calibration/fullHD" or we can define different place by changing *calibImagesPath* constant in *main.hpp* file. Then we need to run application in *Calibration Mode*:

```
cd build
./sv 1
```



Pic 7. Sample calibration image.

The application retrieves the images in pairs according to the alphabetical order. Because of that the images must be named in the way that the pairs of the images from the same view, but different lenses, would be next to each other in alphabetical order. The sample calibration images are in "calibration/fullHD" directory. After the calibration process the application will generate files with *intrinsics* and *extrinsics parameters* in "calibration" directory. If these files are already in this place, there is no need to restart the application in *Calibration mode*, if the recording was made with the same three-dimensional camera.

Testing stereo matching

The application can use one of implemented stereo matching algorithms taking the various parameters. We can test each of them to find the the most effective or most efficient solution by running application in *Stereo matching testing mode*:

```
cd build
./sv 2
```

In this mode application performance is limited to perform stereo matching process only and allows user to check algorithms by changing parameters. To test GPU algorithms, we need to change a value of a directive **USE_GPU** from 0 to 1 in *depthsubstraction.cpp* file.

Running parameters

```
cd build
./sv [agrs]
/**
 * arg1 - application mode:  0 - normal, 1-calibration 2-Stereo matching testing
 * arg2 - the number of frames taken into account in building model
 * arg3 - path to the left stereo camera stream
 * arg4 - path to the right stereo camera stream
 * arg5 - path to the intrinsics camera calibration parameters //not required in Calibration Mode
 * arg6 - path to the extrinsics camera calibration parameters //not required in Calibration Mode
 **/
```

Future work

Things that can be improved:

- The image segmentation algorithm is the most time-comsuming element of the application during the disparity map creation process. The good point would be to replace it with more efficient solution i.e. by proper color histogram analysis or GPU algorithm.
- Currently for each video segment the average of the depth is calculated using the noisy depth map. In the original segment-based algorithm the fitting surface is determined. It can be done by using the least squares algorithm, but it is very time-comsuming for HD frames size. The solution might be to get rid of some outliers points at first and then performing the fitting surface algorithm.
- During the triangulation process the model consisting of every points from point cloud is created. The final model should rather consist of planes, so that some plane interpolation process should be performed on point cloud.