High-Performance and Tunable Stereo Reconstruction

Johann Diep jdiep@ethz.ch
Milan Schilling milansc@ethz.ch
Ian Stähli staehlii@ethz.ch

Abstract

Most mobile robots require fast computation of their immediate surroundings in order to perform fast and maneuverable tasks. Conventional stereo algorithms are not adapted for this kind of applications since they focus on reconstruction quality rather than run-time performance. Therefore, the high-performance and tunable stereo reconstruction method [5] was implemented in this project. The algorithm provides the option to adjust the desired reconstruction quality, at the cost of a slower run-time performance. The result is a disparity image for each left and right frame pair.

1. Introduction

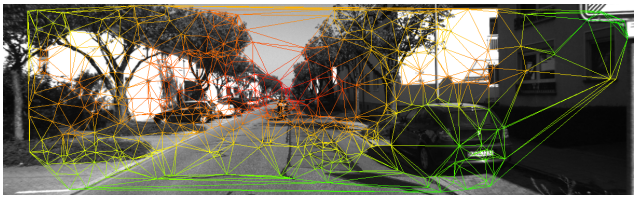

Figure 1. Delaunay-Triangulation are used for the interpolation of disparities between support points.

Many current robotic applications, especially autonomous mobile robots, require real-time performance in order to act quickly and move around its immediate environment in a fast and efficient manner. A 3D reconstruction of the surrounding is needed in order to plan a collision-free path and to achieve local obstacle avoidance. The reconstruction step is done, for example, by processing recordings of a calibrated stereo camera and pose informations of an IMU to a global point cloud. Unfortunately, conventional stereo algorithms focus on reconstruction quality instead of run-time performance and therefore have difficulty to keep up with the high-speed run-time requirement of certain applications. In contrast, the high-performance and tunable stereo reconstruction method proposed in [5] provides the option to adjust the accuracy-versus-speed ratio and is therefore an elegant solution to the described problem. The goal of this project is the implementation of the proposed algorithm.

2. Algorithm Concept

The described algorithm [5] is composed of the following steps:

1. For the iterative part of the algorithm only a small portion of pixels are used (see figure 4). These pixels correspond to the high-gradient regions. These regions are computed by looking at the four direct neighbors of each pixel as seen in figure 2. When $(p_E - p_W)^2 + (p_N - p_S)^2 > t_{grad}$ (see table 2) the pixel is considered to have high intensity gradient.

	p_N	
p_W	p	p_E
	p_S	

Figure 2. Pixel neighbors are considered.

2. Sparse Stereo Matching: In a first step, a sparse set of keypoints is initialized by detecting their positions and corresponding disparities by using the exFAST method described in [4] (see figure 5).

3. Delaunay Triangulation: Using the calculated support points from the initialization step, triangular planes are constructed via Delaunay-Triangulation. These triangles are situated in a 3D space consisting of the pixel coordinates and the disparity, therefore [u,v,d]. The cross product of two edges of a triangle allows to compute the normal vector of the plane $\vec{n} = [n_1, n_2, n_3]$ (see figure 3). The final plane parameter is computed using any vertex of the triangle plane as $n_4 = n_1 \cdot u + n_2 \cdot v + n_3 \cdot d$.

4. Disparity Interpolation: The resulting mesh from the Delaunay triangles is then used to linearly interpolate disparities of pixels within those triangles (see figure 1 and 6). Using the plane parameters calculated before

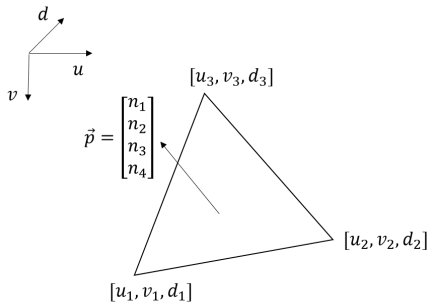

Figure 3. Triangular plane in 3D space.

as $\vec{p} = [n_1, n_2, n_3, n_4]$ the disparity of any pixel [u,v] inside of that triangular plane is computed as $d_{it} = (n_4 - n_1 \cdot u - n_2 \cdot v)/n_3$.

5. Cost Evaluation: Each pixel on the left image is compared with the pixel on the right image using the interpolated disparity. In order to validate these interpolations, the cost is computed using a 5x5 census descriptor.

6. Disparity Refinement: An occupancy grid with a specific edge size is placed and the best and worst matches of each quadrant are stored (see figure 7).

7. Support Resampling: The bad candidate pixels are resampled using epipolar search and then added to the support points (see figure 8). The good candidate pixels are stored directly as new support points.

Subsequent Iteration: At this point, a denser Delaunay mesh can be constructed using the new support points and a further iteration can be done. For any subsequent iteration, the occupancy grid size is reduced by a factor of 2.

Figure 4. High-gradient regions are used for the iterative part of the algorithm.

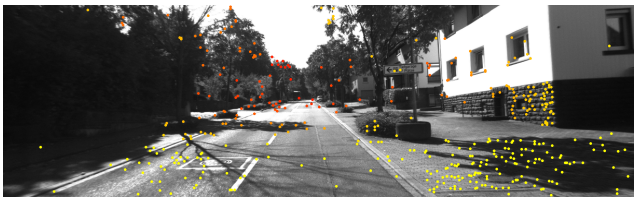

Figure 5. Initialization with a sparse set of keypoints by using the exFAST method.

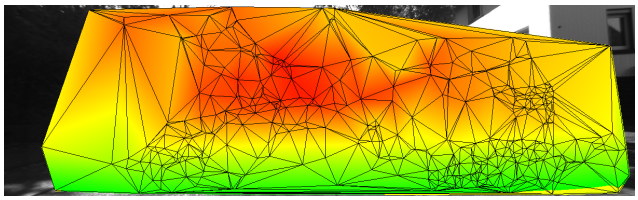

Figure 6. Interpolating disparities via mesh created by the Delaunay-Triangulation.

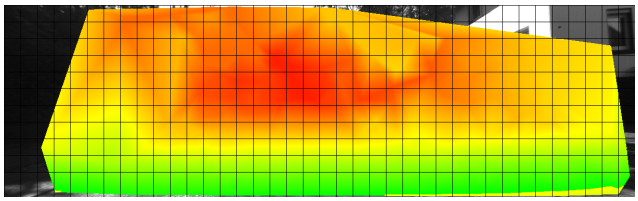

Figure 7. Disparity refinement by using an occupancy grid to store the best and worst matches of each quadrant.

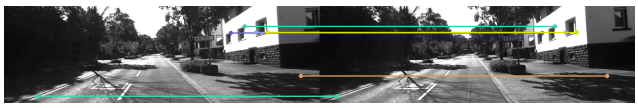

Figure 8. Epipolar search resampling of the bad candidate pixels determined in the previous step.

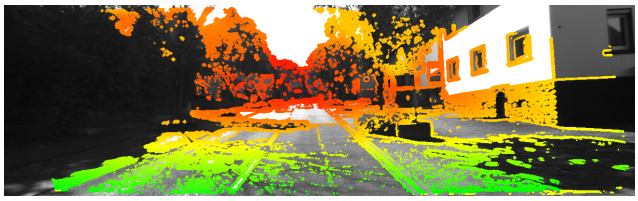

Figure 9. The disparities of pixels with a cost less than a certain threshold are stored into the final disparity image.

3. C++ Implementation

3.1. Variables

Variable	Description
H	Height of the input image
W	Width of the input image
S	Set of support points, [Nx3]
G	Pixel to triangle associations, [HxW]
T	Plane parameters for all triangles, [4xN_T]
D_{it}	Interpolated disparities, [HxW]
D_f	Final disparity image, [HxW]
C_{it}	Cost associated to D_{it}, [HxW]
C_f	Cost associated to D_f, [HxW]
C_g	Pixels with high confidence matching, [H'xW']
C_b	Pixels with high matching cost, [H'xW']
O	Pixels of high-gradient regions, [N_Hx2]

Table 1. Description of variables used in the implementation.

For multidimensional variables the OpenCV container Mat was used. The relevant matrix variables and its dimensions can be found in table 1.

The variable S stores N points and their respective pixel coordinates as well as their disparities as [u, v, d]. This matrix is initialized with the points from the sparse stereo function, which uses the exFAST implementation.

The lookup-table that matches the pixels to its corresponding triangular plane consists of two variables G and T. The matrix G stores an integer value for each pixel to the corresponding triangle. It has value -1 for pixels with no correspondences. The matrix T stores the four plane parameters for each triangle.

To make use of the high-gradient regions we constructed a matrix O of dimension N_Hx2. It stores the pixel coordinates [u,v] for the number of high-gradient pixels N_H. This is used in order to avoid looping over the whole image and rather perform computations on only the relevant pixels.

3.2. Functions

The 7 steps explained in section 2 were implemented as individual functions. The inputs and outputs of these functions, as well as important design choices will be discussed in detail. The functions do not return any values, but rather take the inputs as well as outputs as arguments that are passed by reference.

1. `image_gradient.cpp` This function takes as input the Gaussian blur filtered gray-scale image of the left camera. It provides as output an image of the same size as the input, containing the intensity gradient for each pixel. With the help of this function it is possible to extract the high gradient region of the image. The pixels with a gradient higher than a certain threshold can be stored on the matrix O.

2. `sparse_stereo.cpp` This function takes the gray-scale stereo images as an input and identifies a sparse set of support keypoints which are stored in the matrix S. The keypoints are detected and matched along their epipolar lines via the exFAST method described in [4]. In comparison with the regular FAST algorithm, exFAST consists of better matching accuracy resulting in a less clustered keypoints distribution. Since the library is optimized for performance and do not consider bounds checking, arbitrary image sizes result in a segmentation fault during compiling. Therefore, the input stereo images (1241 x 376 pixels) from the KITTI dataset need to be down-scaled to the standard image size (640 x 480 pixels) for the exFAST to work. After the calculation is done, the sampled keypoints and the corresponding disparities are up-scaled to the original size again.

3. `delaunay_triangulation.cpp` This function takes as input the matrix of support points S and computes a mesh of triangles from these points. The output are the two matrices G and T that constitute a lookup-table to assign each pixel to its corresponding plane parameters.

   The Triangle library [3] provides a function that builds a Delaunay Mesh, given several 2D points. The function returns a triangle list that contains the vertices of each individual triangle. A triangle rasterization is done using the information of the vertices to associate every pixel to its corresponding plane.

4. `disparity_interpolation.cpp` The inputs of this function are the outputs of the previous step, namely the matrices G and T. It also takes the matrix of high-gradient regions O. The output is a disparity image of interpolated disparities D_{it}.

   For each pixel in the high-gradient regions the intersection between a line and a plane is computed in order to find the disparity.

5. `cost_evaluation.cpp` This routine takes as input the left and the right image, the container of the high gradient pixels O, the matrix G from the function 'delaunay_triangulation' as well as the interpolated disparities D_{it} from the previous step. It returns the matrix C_{it} which contains the normalized cost for each pixel in relation to D_{it}. Other than that, this routine returns the census transformed left and right image, which will be re-used in step 7.

   The input images get census-transformed using the SIMD optimized census transform included in the exFAST algorithm. Since the left and the right image can be transformed independently, the computations can be distributed on two threads.

6. `disparity_refinement.cpp` This function takes the interpolated disparity D_{it} as well as their associated cost C_{it} and the matrices G and O as an input. It returns the matrices C_b and C_g which contain for each quadrant the pixel with the worst census match or the best census match, respectively, as shown in figure 7. Further, it returns the updated final disparities D_f and final cost C_f.

7. `support_resampling.cpp` This function takes as input the best and worst costs C_g and C_b from the last step as well as the two census transformed images from step 5. It returns the updated container of the support points S.

   This function adds the pixels with a good matching cost C_g directly to the matrix S, while the pixels

with a bad matching cost C_b are sent to the epipolar stereo matcher. This matching routine tries to find a match along the epipolar line for each previously bad matched pixel. To achieve this, we extract a pattern of five census values (the census at (u, v) and those at $(u \pm 7, v \pm 7)$) from the pixel of interest in the left image. Then we compare this pattern to the associated pattern for each pixel in the right image along the epipolar line within a certain region. If the best match of those comparisons is below a certain threshold, the match is considered as valid and the correspondent disparity is also added to the matrix S.

`pipeline.cpp` The pipeline is the function that runs the whole algorithm, taking a left and a right image as input. It follows the same structure as the pseudocode explained in [5].

3.3. Parameters

For the exFAST implementation of sparse stereo, there are two relevant parameters that can be adjusted, the uniqueness and a minThreshold. These parameters allow to modify the number of initial points that are used as support points. The t_{grad} variable is the intensity threshold that is used to define the range of the high-gradient regions. After the cost evaluation is done the two thresholds t_{lo} and t_{hi} are used to validate the disparities. The number of iterations n_{iters} can be increased and this will provide a higher resolution of the triangle mesh. The value for all these parameters can be found in table 2.

Parameter	Value
uniqueness	0.4
minThreshold	10
t_{grad}	20
t_{lo}	21/24
t_{hi}	2/24
n_{iters} (tunable)	[1, 4]

Table 2. Used parameters and their corresponding values.

3.4. Pointcloud

In a further step, a pointcloud was constructed. In order to determine the 3D positions of a features given the corresponding disparity map, the following mapping functions were considered:

$$Z = f * b/d \tag{1}$$

$$X = u * Z/f \tag{2}$$

$$Y = v * Z/f \tag{3}$$

With the disparity d in pixel length, the baseline b and the focal length f of the KITTI dataset ($b = 0.54\ m$ and $f = 7.18856 * 10^2\ px$ as defined in [2]), the feature point (with coordinates u and v) is projected to the 3D space according to the equations 1, 2 and 3. In order to bring the 3D projection of each frame into one common inertial system, the 3D coordinates are multiplied with the corresponding ground-truth transformation matrix.

4. Results

	Accuracy (%)			
Method	$<$ 2px	$<$ 3px	$<$ 4px	$<$ 5px
SGBM [1]	92.3	95.1	96.3	96.9
HPTSR-1	70.3	77.9	82.7	85.9
HPTSR-2	69.8	77.6	82.5	85.8
HPTSR-4	69.5	77.4	82.3	85.6

Table 3. Accuracy comparison between SGBM [1] and the implemented stereo algorithm at different iterations (indicated by the number next to the method).

	Run-Time	
Method	Hz	ms
SGBM [1]	1.52	659
HPTSR-1	22.5	46.7
HPTSR-2	16.1	64.3
HPTSR-4	11.5	89

Table 4. Run-time comparison between SGBM [1] and the implemented stereo algorithm at different iterations (indicated by the number next to the method).

The algorithm was compared to the Semi-Global Block-Matching (SGBM) algorithm [1]. As shown in the table 3, the presented algorithm provides a less accurate result while performing on a speed that is more than 14 times faster than the SGBM algorithm. The algorithm runs at a top speed of 22.5 Hz when performing a single iteration (see table 4). As the number of iterations is increased, the speed decreases to 11.5 Hz for 4 iterations.

The pointcloud with a single iteration of 100 sequences stacked together can be seen in figure 10. As expected, the pointcloud provide an overview of rough structure as well as the trajectory of the stereo camera. Due to the loss in accuracy of the disparity estimation algorithm, fine details are not distinguishable.

4.1. Limitations

The maximal performance in the paper of 100Hz could not be reached. The accuracy does not get better with more iterations, in fact it gets worse. The cause of this problem might be the resampling using epipolar search. The matches between the left and right image are not always right, and this causes false disparities in the support points.

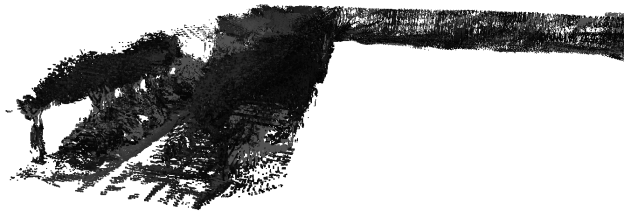

Figure 10. Pointcloud computed by combining HPTSR disparity estimations with ground-truth orientation datas.

5. Work Distribution

For the initial work distribution we assigned the development of functions to each team member. Functions that were closely related to each other were done by the same person, as seen in table 5.

Function	Responsibility
sparse_stereo	Johann
delaunay_triangulation	Ian
disparity_interpolation	Ian
cost_evaluation	Milan
disparity_refinement	Milan
support_resampling	Milan

Table 5. Initial work distribution within the group.

Johann developed the accuracy calculation using ground truth disparities and also the pointcloud. Ian developed the SGBM setup to compare our algorithm to an existing method. Milan developed the implementation of high-gradient regions. Ian managed the work distribution and together with Milan debugged the pipeline.

6. Conclusion

Most conventional stereo matching algorithms focus on quality over quantity and therefore sacrificing runtime performance for high accuracy disparity calculation. However, most mobile robotic applications set the calculation runtime as a high priority in order move around the environment in a fast and efficient manner. In that sense, the HPTSR algorithm implemented in this project is proved to be highly suitable for these kinds of problems. While running at 22.5 Hz for one iteration, it is 14 times faster than the SGBM algorithm while being 22% less accurate. However, the 100 Hz runtime performance described in the paper could not be reached. Besides, the tunable feature of the algorithm could not be implemented. Instead of improvement with more iterations, the accuracy falls down.

References

[1] Semi-Global Block Matching Stereo Correspondence Algorithm in OpenCV. http://docs.opencv.org/.

[2] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset, 2013.

[3] Jonathan Richard Shewchuk. Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator. https://www.cs.cmu.edu/ quake/triangle.html.

[4] Konstantin Schauwecker, Reinhard Klette, and Andreas Zell. A New Feature Detector and Stereo Matching Method for Accurate High-Performance Sparse Stereo Matching, 2012.

[5] Sudeep Pillai, Srikumar Ramalingam, and John J. Leonard. High-Performance and Tunable Stereo Reconstruction, 2015.