HONDA RESEARCH CODE SPRINT : ROAD SEGMENTATION

ALEXANDER J. B. TREVOR, FEDERICO TOMBARI, RADU B. RUSU

1. Introduction

The goal of this code sprint was to perform segmentation of stereo pairs to identify the drivable road surface. The provided data included three sequences of stereo pairs, along with corresponding disparity images generated using LIBELAS. Our aim was to add the required tools to PCL to make use of this data, and add segmentation tools that would be suitable for segmenting such scenes. As this is a PCL project, our focus is on segmentation approaches that operate on the geometry of the point cloud, as opposed to appearance-based segmentation. Note also that this is a segmentation project, and not a lane-finding project, in that we do not make assumptions about the structure of the scene, such as the presence of lane markings etc. However, this segmentation could be informative to such a system.

2. Provided Datasets

2.1. **Overview of Datasets.** The three provided sequences were collected from a wide-baseline (84cm) stereo rig on a car driving through several different locations. Example images from each dataset can be seen in Figure 1. All sequences were taken with the vehicle driving on a roadway, but with differing surroundings. The car hood is visible in the lower portion of all images.

Figure 1. Example images from the datasets used in this work. At left is the castro dataset, center is the maude dataset, and at right is the national dataset.

2.2. **Provided Disparity Images.** We initially developed a tool to convert the provided disparity images to point clouds compatible with PCL. Given a disparity image, conversion to a point cloud is straightforward. Given a stereo baseline in meters, and a principal point (c_x, c_y), and focal lengths f_x and f_y, the 3D (x, y, z) coordinate corresponding to pixel (u, v)'s, disparity is given by the well-known equations:

$$z = (baseline * f_x)/disparity$$

$$x = z * ((u - c_x)/f_x)$$

$$y = z * ((v - c_y)/f_y)$$

This resulted in point clouds useable with PCL's tools, but the resulting data had several issues that made segmentation difficult. One issue was the amount of the image with valid depth data. A brief comparison is shown in Figure 2. Much of the lower portion of the image did not include depth data, giving relatively little coverage of the ground plane. This was compared to OpenCV's Semi Global Block Matching (SGBM) approach, which provided much better coverage for the parts of the ground not occluded by the car's hood.

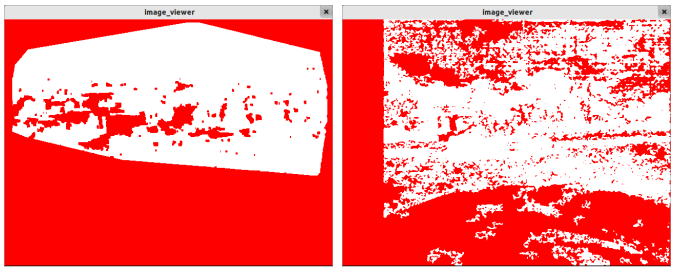

FIGURE 2. A visualization of points with valid depth data, for LIBELAS at left vs. OpenCV's SGBM at right. Points with depth data are shown in white, while points without valid depth data (depth of NaN) are shown in red.

Another difficulty was with the smoothing done by LIBELAS. In many instances, points would be smoothed and / or interpolated between surfaces, resulting in incorrect depths. This type of smoothing would be difficult for our segmentation approach to cope with, making these point clouds relatively unsuitable for our work. An example scene demonstrating this issue is shown in Figure 3.

Note that all clouds shown here were generated from the provided disparity images. It may be possible to achieve better performance from LIBELAS using a different set of parameters, but this was not investigated.

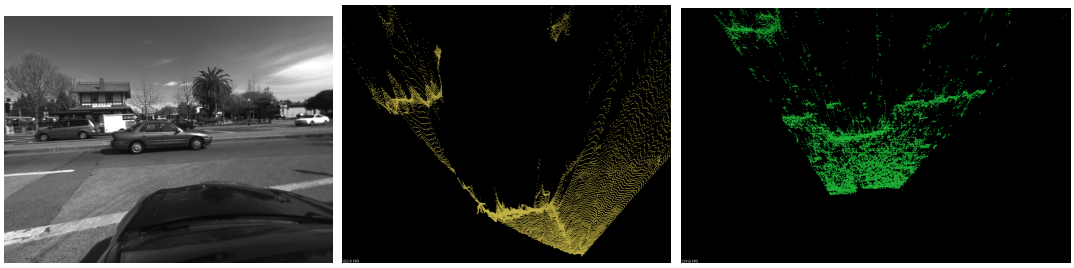

FIGURE 3. An example demonstrating the smoothing errors with the provided disparity images. At left, an image with two cars in the foreground. Center shows the resulting point cloud from the provided disparity images, while the right shows the result from SGBM. Several smoothing errors can be seen. The surface of both cars has a somewhat "lumpy" appearance in the center, but this is not apparent in the SGBM result. A raised ridge on the ground plane can be seen at the front of the nearer car, also not apparent in the SGBM result. Finally, the curb on the median is visible in the SGBM result, but has been smoothed over in the result at center.

2.3. **Decision to Evaluate Other Stereo Algorithms.** In light of the above problems, we determined that the segmentation results using this stereo approach would not be satisfactory. After discussing this with Radu, we contacted Federico Tombari, who has extensive experience with stereo processing. He agreed with our assessment that the existing stereo approach was not suitable for segmentation, and agreed to help us find alternative approaches. We then decided to split the code sprint into two parts: development of new stereo processing algorithms to be added to PCL, and the original segmentation project.

3. Evaluation of Stereo Algorithms

An informal evaluation of various stereo algorithms was performed to identify an approach that would perform better than the provided LIBELAS approach. The stereo matching framework, provided by Federico, includes several stereo matching algorithms that have been tested on the provided data in order to determine whether any of those could prove to be suitable for 3D reconstruction aimed at the successive segmentation stage. This stereo evaluation step was motivated by the fact that, given the current state of stereo matching algorithms in literature, it is often hard to *a-priori* predict which algorithm - among the huge pool of methods recently proposed in literature - could perform best on some specific dataset and under specific conditions, such as those related to our project. In particular, Stereo data provided by Honda is affected by illumination changes and photometric distortions and often includes low textured areas (e.g. roads and sky), which create challenging conditions for most state-of-the-art stereo matching algorithms.

Algorithm	Category
Block-based (BB) [1][10]	local
Shiftable Window (SW) [3]	local
Fast Aggregation (FA) [12]	local
Adaptive Weight (AW) [14]	local
Libelas (LIB) [6]	local
Semiglobal matching (SG) [7]	global
2-pass Scanline Optimization (2SO) [10]	global
Efficient Belief Propagation (EBP) [4]	global
Adaptive Cost - Scanline Optimization(ACSO)	hybrid

TABLE 1. A subset of the evaluated stereo matching algorithms.

For this reason, in addition to testing different stereo matching algorithms, also specific pre- and post-processing steps were evaluated. In particular, as for pre-processing, algorithms were tested also on images were each pixel was subtracted with the mean intensity value computed over a square window centered on that pixel. This is a typical pre-processing step - equivalent to a high-pass filter removing the AC component - useful to remove constant additive photometric distortions between the two images (e.g. such as that typically brought in by different values of the gain and offset on the two cameras composing the stereo rig). The pre-processing turned out to be useful for almost the totality of algorithms. As for post-processing, possibly wrong disparities were filtered by means of four different filters:

- Ratio Filter (RF) [1]: this filter analyses the set of values related to the matching (or cost) scores computed for a point over its disparity range. If the ratio between the global minimum and the first local minimum is higher than a pre-defined threshold, then the associated disparity is not reliable enough and it is discarded.
- Peak Filter (PF): this filter also takes into account the matching costs related to a specific cost, similarly to the RF. Differently, though, it discards a disparity if the right and left derivatives around the global minimum are too low, which is a sign that the global minimum lies on a flat peak (typical of low textured areas).
- Left-Right Consistency Check (LRC) [5]: this filter eliminates disparities that are not consistent with their homologous values on the disparity map computed by inverting the stereo pair (i.e. considering the reference image as the target one, and viceversa). It is an effective filter, at the cost of twice the disparity map computation.

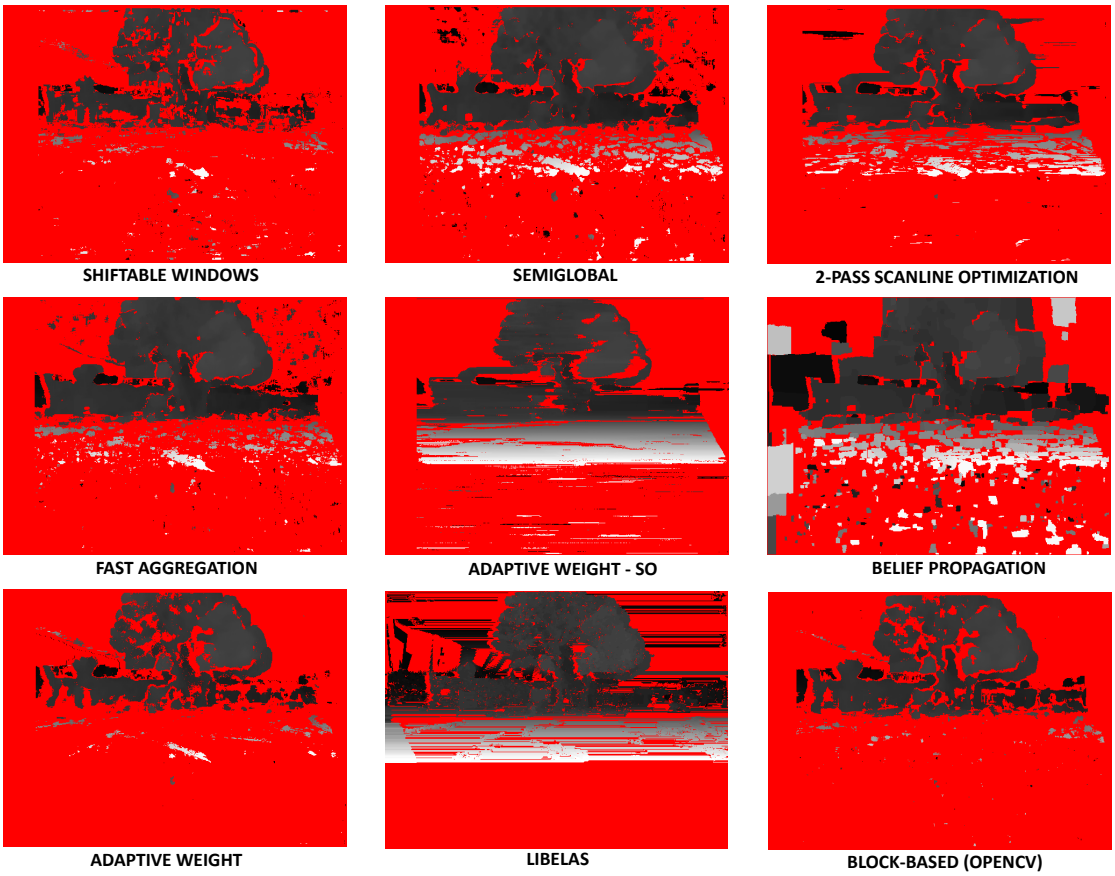

FIGURE 4. Disparity maps yielded by a subset of the evaluated stereo matching algorithms on the first frame of the *National* dataset. Adaptive Weight - SO has been implemented in PCL, and was used for the segmentation task.

- Median Filter (MF): differently from previous filters, this one does not eliminate ambiguous disparity values from the disparity maps, but substitutes outliers computed over a squared window with the median value of the disparity distribution within the window.

A first "screening" stage has outlined a subset of algorithms that have been subsequently evaluated more in details (e.g., trying different parameter combinations on different frames from the three provided datasets). Table 1 reports a summary of these proposals, while Figure 4 shows the disparity maps yielded by these algorithms on the first frame of the *National* dataset. All disparity maps shown in the Figure were obtained with the same values of the common parameters, such as disparity range and radius for the matching score computation. All matching scores are based on the Sum of Absolute Differences (SAD) measure. Also, all

algorithms undergo the same pre-processing (see before) and post-processing steps (RF, PF, LRC, MF, with their parameters adjusted so as to eliminate most wrong disparities from the map).

Quickly commenting the results, we can see that most algorithms suffer the presence of noise in the data and are not able to yield accurate 3D reconstructions. Local algorithms are those that seem to provide the worst results, especially in terms of visual sparsity of the map (but please note that sparsity is directly proportional to errors, due to the presence of the post-processing filters). EBP also has an insufficient performance, rather "blocky" and noisy. Instead, the smoothness constraint applied on scanlines, such as that provided by 2SO, SG, ACSO seems to be effective in yielding more accurate reconstructions. Overall, the method that seems yielding the best performance is ACSO, which is a method provided by Federico and inspired by [13]. The main difference between ACSO and [13] is that Dynamic Programming is substituted by a 2-pass Scanline Optimization. The algorithm proposed in [13] was also evaluated but did not perform as good as the others. As for Libelas, it can be seen that the visual output does not look too different from that provided by ACSO. Nevertheless, the presence of strong streaking effects which were hard to filter (visible in particular in the 3D point cloud) tended to distort notably the 3D output. Successive tests carried out on other frames from all 3 datasets yielded analogous results.

The ACSO is thus the algorithm selected to process all stereo data in order to compute the point clouds. Processed point clouds were also augmented with the RGB information provided by the reference image of the stereo pair, so as to help visualization and since it could be useful for successive stages.

4. Segmentation Approach

Once the stereo pair has been converted to a point cloud, we can proceed with segmentation. We have designed the segmentation approach with efficiency in mind, and it runs in near-real time. In this section, we will describe the overall segmentation approach.

4.1. **Organized Connected Component Segmentation.** We have developed a connected component based segmentation approach can operate on point cloud data, surface normals, and image data. Given a pixel $I(x, y)$, we can examine its neighboring pixels $I(x-1, y)$ and $I(x, y-1)$ in a 4-connected sense, comparing various features, such as proximity in euclidean space, similarity of surface normals, or similarity in color space. If the pixel and its neighbor match according to the comparison function, the neighbor is added to the same segment, otherwise a new segment is created. This segmentation approach can be used for a variety of tasks by using difference comparison functions, including planar segmentation, color segmentation, greedy euclidean clustering, or combinations of these (planes of a solid color, etc). Smooth surfaces such as planes or road-surfaces can be segmented

by comparing the dot product between neighboring normals, as well as euclidean distance between neighboring points. If desired, surfaces with smoothly changing surface normals that are not actually planar can be discarded if the curvature is too high.

Running this algorithm will result in a set of labeled segments that are both spatially connected in the image, and are considered part of the same segment based on the comparison function used (for example, the all pixels in the segment may be a similar color, have similar normals, are close in euclidean space, etc). For most applications, the areas of interest will correspond to segments with more than a specified number of inlying pixels, so a minimum inlier parameter is available for filtering out small segments. The application of this algorithm to the problem of road segmentation is described below.

4.2. **Road Segmentation.** Our organized connected component segmentation is a very general technique, which can be applied to road segmentation. We designed a comparison function for segmenting road surfaces. One important note here is that this does not assume that the ground is perfectly planar, as most roads are not. Roads that slope towards the edges are quite common, primarily for drainage purposes. Instead, we assume that the drivable surface has smoothly changing normals, and that the points lie within some expected range (possibly quite wide) from the "expected ground plane". This requires the "expected ground plane" as input, specified in Hessian normal form. The expected ground plane is specified with respect to the left camera pose, and is the ground plane the vehicle's tires would be on, if the ground were perfectly planar, as shown in Figure 5. Tolerances for the normal direction and range are also specified as input to the segmentation. This allows points that are too-far outside the drivable range to be excluded. Given the expected ground plane equation $G = \{a, b, c, d\}$, and thresholds $road\_angular\_thresh$ and $distance\_thresh$, to be considered part of a road segment, a point P with normal N must satisfy the following condition:

- $N \cdot \{a, b, c\} > cos(road\_angular\_threshold)$

An example demonstrating this can be seen in Figure 6.

To enforce smoothness, points must also have a similar surface normal to their neighbors. Recall that points are compared in a 4-connected sense in our algorithm. To be considered part of the same segment, a point P_1 with normal N_1 and a neighboring point P_2 with normal N_2 must satisfy the following condition:

- $N_1 \cdot N_2 > cos(angular\_threshold)$

An example demonstrating this is shown in Figure 7.

The result of using this comparator with the organized connected component algorithm is a set of zero or more regions that fit the above criteria: they are connected in the image, have surface normals within the specified range of the expected ground plane, have smoothly changing normals within the segment, and

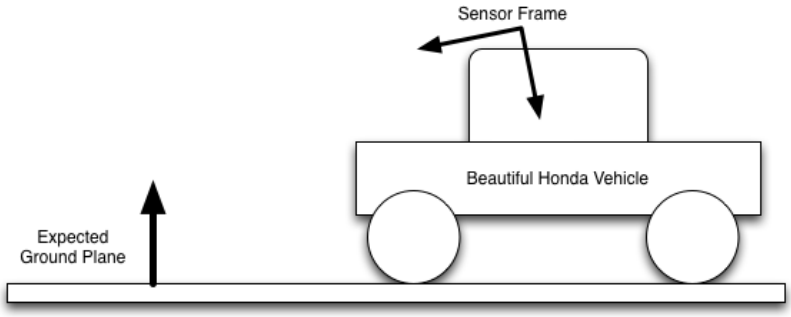

FIGURE 5. A diagram demonstrating the "expected ground plane" – the plane relative to the sensor pose that the vehicle's wheels should rest on.

FIGURE 6. Given the expected ground plane normal in the center, the normals at the sides indicate the acceptable range of normals given an angular threshold of 15 degrees.

FIGURE 7. At left, adjacent normals with acceptably smooth variation given an angular threshold of 2 degrees. At right, normals without smooth variation. Note that all normals shown are within the acceptable range from the example above.

are connected within the image. Only regions with more than $min\_inliers$ points are returned.

4.3. **Road Region Growing.** Surface normals can be quite noisy on the stereo data, which can result in several disjoint regions labeled as ground. To alleviate this problem, we perform an additional segmentation run on the point cloud to grow the set of ground regions to include points that do not match the surface normal requirements described above, but have a point-to-plane distance less than

a specified threshold, and are adjacent pixels to one of the identified road surface regions. The "planar refinement" comparator has been implemented to perform this task, and can be called by using the "refine" method of the organized connected component segmentation algorithm.

As the resulting points do not meet the more strict requirements described above, we consider these to be lower confidence ground points. These are denoted in yellow in the resulting images, rather than green as is done for road regions that do match the surface normal requirements.

4.4. **Obstacle Detection.** Limited experiments were also performed for detection of nearby obstacles. Once again, we make use of the organized connected component class, but with a different comparison function. To detect obstacles, we detect sets of pixels that have valid depths, where points are within some distance of each other in euclidean distance, and are not part of the previously labeled groundplane. Segments that have both a sufficient number of points, and have centroids that are sufficiently far away from the plane are returned. An example is shown in Figure 8.

FIGURE 8. An example of the obstacle detection.

5. Evaluation of Segmentation

We evaluate the results of the segmentation both quantitatively and qualitatively. Video results are provided for each sequence, which are described below. Additionally, images were labeled by Alex for use as ground truth, and compared to the segmentation results.

5.1. **Video Results.**

5.1.1. *Kinect Ground Segmentation.* During the development of the stereo tools, an initial verification of the ground segmentation approach was performed on data collected from a Microsoft Kinect on Alex's mobile robot. Point clouds generated from the Kinect are much denser and less noisy than point clouds generated from

stereo, but this was suitable for an initial verification of the ground segmentation approach. An example is shown in Figure 9, and the full video sequence is available on Youtube at the following URL: `http://www.youtube.com/watch?v=nHvRGaDO72c`.

FIGURE 9. Example scenes of Kinect data segmented using our approach. This was performed to validate the segmentation approach during the development of the stereo tools.

5.1.2. *Segmentation of Provided Datasets.* The provided datasets have also been segmented using this technique. Several example results are shown in Figure 10. Video results for the castro dataset are available on Youtube at the following URL: `http://www.youtube.com/watch?v=ZCuOxw3thDE`. Video results for the national dataset are available at: `http://www.youtube.com/watch?v=7uGPbg7XZOE`. Video results for the maude dataset are available at: `http://youtu.be/DU5G-dHOlog`.

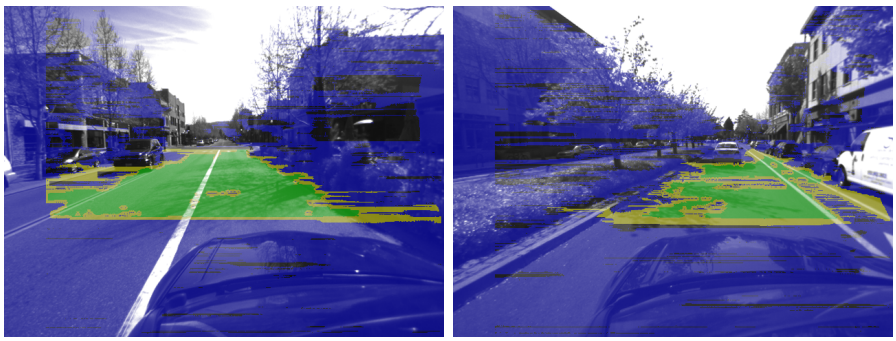

FIGURE 10. An example of our segmentation results on the castro dataset. Segments detected using the road surface comparator are shown in green, and pixels added by the road region growing are shown in yellow. Pixels without valid depth data are shown in blue.

5.2. **Quantitative Results.** In order to evaluate the segmentation approach, a small subset of the provided images from were manually labeled, and compared with the segmentation result. For these quantitative results, every 500th frame from the castro dataset was used (12 images). Roadways were labeled in green. In addition to the roadways, sidewalks, curbs, and other surfaces that the vehicle could (but probably should not) drive on were labeled in yellow, to allow us to provide more informative statistics regarding false positives. An example of such a labeled image is shown in Figure 11.

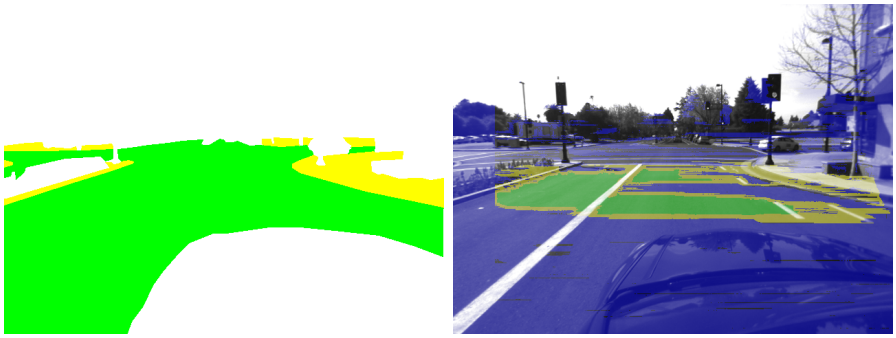

FIGURE 11. At left, an example of the ground truth segmentation for frame 2500 of the Castro dataset. Roadway is labeled in green, while sidewalks, curbs, and other low-lying areas are labeled in yellow. All other pixels are labeled with white. At right, our segmentation result for this image.

An evaluation script was written to load both the ground truth image and segmentation result for comparison. Results are given in Table 2. The values indicates the percentage of image pixels for which valid depths were computed, and the second indicates the percentage of ground truth pixels with valid depths – this is the percentage of the image that resulted in valid points. Reasons for invalid depths include areas not in view of both cameras, and low texture areas such as sky or the car hood. We should also note that points at the same or closer distance to the camera as the car hood will not have valid depths, as the disparity range used for these experiments was limited to disparities smaller than this. This design decision was made because most of these points are not visible anyway (due to the car hood in one or both images), and the complexity of most matching algorithms scales with the size of the disparity range. Since the points are the input to the segmentation system, all subsequent statistics are reported with respect to input points (valid depths), rather than all pixels. We report separate values for both the regions segmented using the road surface comparator (shown in green), and regions extended with the plane refinement comparator (shown in yellow).

TABLE 2. Segmentation Results

% pixels with valid depths (points)	0.297539
% ground pixels with valid depths (points)	0.336766
% points correctly classified (High Confidence / Green)	0.324700
% points correctly classified (High and Low Confidence / Green +Yellow)	0.775866
% non-ground points classified as ground (High Confidence / Green)	0.007390
% non-ground points classified as ground (High and Low Confidence)	0.052089
% non-ground+sidewalk points classified as ground (High Confidence / Green)	0.000844
% non-ground+sidewalk points classified as ground (High and Low Confidence)	0.015273

5.3. **Discussion of Results.** The results shown demonstrate that the drivable ground surface can be detected in many circumstances using our approach, but there are also a number of shortcomings. The results on the Kinect dataset indicate that the base algorithm works well on dense and clean data, but current stereo techniques do not yet produce point clouds of comparable density and quality. The most apparent difference is the relatively large number of pixels without valid depths, giving us many fewer pixels to work with. Another challenge is that it is often difficult to distinguish roadways from sidewalks when using purely geometric cues – this seems to be the most common failure mode in the results, as can be seen in the above quantitative results.

6. Relevant Tools in PCL

PCL now includes several tools relevant to this project. These tools fall in two main categories: stereo processing tools, and segmentation tools. These will be described below.

6.1. **Stereo processing tools.** A stereo library has been introduced in PCL. This required creating a new library module, which was called *pcl stereo*. This module includes a basic abstract class for stereo matching and a set of subclasses, each representing a different stereo matching algorithm. Among additional functionalities that were not developed within this code sprint since not necessary to the goal of the project, but that could be usefully implemented to improve the module in the next future, we report

- implementation of a *stereo grabber* module extending the PCL grabber that is interfaced with the stereo matching library
- porting of algorithms for hole-filling, to reduce the sparsity of the filtered stereo maps
- extension of the stereo class architecture to color stereo pairs, providing support for those algorithms that inherently perform stereo matching based on color information

- calibration and rectification utilities, allowing users to perform stereo reconstruction on data that is not pre-rectified

Obviously, one main direction towards improving the stereo library is porting and implementation of additional stereo matching algorithms, since currently only two solutions are available.

6.1.1. *Stereo Matching class.* The stereo matching class (*pcl::StereoMatching*) provides basic methods and interface for stereo matching algorithms. The input data is a point cloud of type *pcl::RGB*, i.e. including an array of color triplets. The class includes the following functionalities:

- stereo matching: given a rectified stereo pair in the form of two point clouds, it computes a disparity map based on the specific algorithm invoked
- pre-processing: the input pair can be pre-processed (see above) before being inputted to the stereo matcher
- post-processing: the computed disparity map can be post-processed by means of any of the 4 previously introduced filters: RF, PF, LRC, MF.
- visual map: a visual map with the disparity values rescaled on the [0 255] range can be computed to be visualized (e.g. by means of *pcl::ImageViewer*).
- point cloud computation: a *pcl::PointCloud* of 3D points can be computed out of the disparity map provided the stereo calibration parameters. The point cloud can be augmented with color information taken from the reference image if needed.

The stereo matching class includes also a subclass, called *pcl::GrayStereoMatching*, that specializes the functionalities of its abstract class to grayscale processing. In case color stereo pairs are inputted and an algorithm extending this subclass is called, then the input pair is automatically converted to grayscale. It is worth noting that this hierarchy is suitable in case, in the future, an extension of the library module to stereo processing of color images is planned (see above). Please refer to the provided documentation on PCL for more details on these classes.

Parallel work [1] has provided a useful tool to convert a *.png* image to a *.pcd* file of type *pcl::RGB*, so that the Honda dataset could be batch converted to pcd files. This is comfortable but memory inefficient, since pcd files are not compressed and the operation results in a notable surplus in terms of memory occupancy of the pcd files with respect to their png counterparts. This aspect could be better dealt with with future work focused on PCL I/O types and definitions.

6.1.2. *Block-based stereo matching.* This subclass implements the baseline Block-Based (BB) stereo matching algorithm. For what concerns the stereo matching part, it is algorithmically equivalent to its OpenCV counterpart [1], while in terms of implementation, the algorithm present in OpenCV is optimized with SIMD instructions - thus that is expected to be more efficient.

[1] carried out by Gioia Ballin within her Google Summer of Code code sprint

The algorithm includes a running box filter so that the computational complexity is independent of the size of the window (O(1) with respect to the size of window). It is based on the SAD matching function and extends the *pcl::GrayStereoMatching*, thus it only works with grayscale (single channel) rectified images.

6.1.3. *Adaptive Cost 2-SO stereo matching.* This subclass implements the ACSO method as explained above. Cost aggregation is performed using adaptive weigths computed on a single column as proposed in [13], but instead of using Dynamic Programming as in [13], the optimization is performed via a 2-pass Scanline Optimization approach. Also this algorithm is based on the SAD matching function and extends the *pcl::GrayStereoMatching*, thus it only works with grayscale (single channel) rectified images.

6.1.4. *Stereo Matching demo.* This demo loads a stereo image pair and computes the disparity map and related organized point cloud using the ACSO stereo matching algorithm implemented in the *pcl stereo* module, using as default parameters those values that were found being among the best in terms of 3D reconstruction on the Honda datasets. After stereo matching, a rescaled version of the disparity map is displayed, as well as the computed RGB point cloud.

6.2. **Organized Segmentation.** The segmentation algorithm described in Section 4 can be found in *trunk/segmentation/* as *organized_connected_component.h*, *organized_connected_component.hpp*, and *organized_connected_component.cpp*. Several comparators are available for different segmentation problems.

6.3. **Road Surface Comparator.** The road surface comparator is for use with the organized connected component class. It implements a comparison function as described in Section 4.2. It can be found in:

pcl/trunk/segmentation/include/pcl/segmentation/road_surface_comparator.h

6.4. **Planar Refinement Comparator.** As described in Section 4.2, road surface regions that include surface normal information are extended in a second pass to include adjacent pixels that have valid depth data, and have a point-to-plane distance under a specified threshold. This step is performed with the planar refinement comparator, which can be found in:

pcl/trunk/segmentation/include/pcl/segmentation/planar_refinement_comparator.h

6.5. **Stereo Road Segmentation Demo.** A demonstration program was developed to test and make use of the above components to perform the actual segmentation task. Segmentation results are displayed in two visualization windows: an image viewer, and a 3D point cloud viewer. This includes code examples demonstrating the usage of all components described above. The input to the program is two directories of images that have been converted to PCD format using the *pcl_png2pcd* tool. Image pairs are loaded, and processed using the ACSO stereo matcher, resulting in point clouds. Surface normals are estimated using PCL's integral image normal estimation class. The point cloud and its surface normals are the input to our organized connected component algorithm, which makes use of the road surface comparator to segment drivable regions, which are annotated in green in the application's visualizers. These are then extended using the planar refinement comparator. Finally, obstacles can be optionally detected. This example can be found in:

pcl/trunk/apps/src/stereo_ground_segmentation.cpp

7. Possible Future Work

7.1. **Appearance-based Segmentation.** The segmentation approach thus far has focused on using only geometric features. However, appearance-based features could be used in conjunction with the geometric features to improve segmentation results. For example, if color images were available, color information could also be included in the segmentation, requiring regions to be "road colored". This cue would be particularly helpful for roads with lane markings, allowing our segments to end at the edges of lane markings. Most road segmentation and lane-finding techniques in the literature make use of color information.

7.2. **Segmentation from Multiple Views.** Currently, the segmentation operates on single frames. However, the accuracy of the segmentation could be improved by making use of multiple frames. Areas with invalid depths are particularly noisy, so using multiple frames to perform segmentation would help alleviate this. Because the vehicle moves, this would require some way of computing the relative poses between frames so that points can be compared in a consistent coordinate frame. A visual odometry system such as FOVIS may be appropriate for this [2] [8].

7.3. **Lane Finding & Utilization of Semantic Knowledge.** If the vehicle will always be operating on well-marked roadways, the segmentation approach could make use of additional knowledge of the environment. Humans make use of a variety of visual cues for navigation on roads, including lane markings, road signs, curbs, and traffic lights. The existing segmentation approach could be extended to take advantage of these cues as well. There have been several efforts on this topic, including [9]. Image-based edge detection tools have recently been added to

PCL, which may be helpful for such a task. If lane boundaries are available, these could be combined with the existing segmentation approach to truncate segments to within the lane.

7.4. **Evaluation of stereo techniques for typical AVN data.** In terms of stereo, apart from the aforementioned improvements to the PCL stereo module, which are more related to the development of the stereo library within PCL, interesting future directions specifically aimed at the goals of the code sprint concern a further in-deep analysis of state-of-the-art stereo algorithms based on the results of the present activity. Our analysis has in fact highlighted how certain categories of the vast stereo matching literature - such as, e.g., those enforcing the smoothness constraint along scanlines - seem to be better suited to yield accurate reconstruction on the data of interest. Nevertheless, as just mentioned, the stereo literature is vast, and includes a high number of techniques even just published - consider for example the Middlebury Stereo Evaluation website, the reference benchmark for stereo matching development, where 61 out of the 163 currently ranked techniques refer to a conference proceeding or journal published just in the past three year (i.e. since 2010). Hence, future directions would include an activity of literature scouting aimed at theoretically determining the best stereo matching algorithm(s) published in the past few years based on the outcome of the present analysis. This analysis should not only favor accuracy and robustness to noise, but also efficiency, given the strict real-time constraints imposed by the application of interest. The analysis could be carried out by trying to highlight the best accuracy/efficiency trade-off, e.g. similarly to what was done in [11]. PCL implementation/porting and optimization (for further efficiency) of the selected algorithm(s) would then represent the successive step along this direction.

Moreover, analyzing different, more advanced pre- and post-processing techniques for filtering and denoising the input data and the disparity map could prove to be useful to improve the results, given the specific distortions present in the data of interest.

References

[1] `http://opencv.willowgarage.com`.
[2] `http://code.google.com/p/fovis/`.
[3] A.F. Bobick and S.S. Intille. Large occlusion stereo. *Int. Journal Computer Vision*, 33(3):181–200, 1999.
[4] P.F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. In *Proc. CVPR 2004*, 2004.
[5] P. Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6(1):35–49, 1993.
[6] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Proc. Asian Conference on Computer Vision*, November 2010.

[7] H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Proc. Conf. on Computer Vision and Pattern recognition (CVPR 2005)*, volume 2, pages 807–814, 2005.

[8] A.S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *International Symposium on Robotics Research (ISRR)*, 2011.

[9] A.S. Huang, D. Moore, M. Antone, E. Olson, and S. Teller. Multi-sensor lane finding in urban road networks. *Proceedings of Robotics: Science and Systems, Zurich, Switzerland*, 2008.

[10] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. Jour. Computer Vision*, 47(1/2/3):7–42, 2002.

[11] F. Tombari, S. Mattoccia, and L. Di Stefano. Stereo for robots: quantitative evaluation of efficient and low-memory dense stereo algorithms. In *Proc. Int. Conf. on Control Automation Robotics and Vision (ICARCV)*, 2010.

[12] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. Near real-time stereo based on effective cost aggregation. In *Proc. Int. Conf. on Pattern Recognition (ICPR 08)*, 2008.

[13] Liang Wang, Miao Liao, Minglun Gong, Ruigang Yang, and David Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *Proc. 3rd Int. Symposium 3D Data Processing, Visualization and Transmission (3DPVT'06)*, pages 798–805, 2006.

[14] K.J. Yoon and I.S. Kweon. Adaptive support-weight approach for correspondence search. *IEEE Trans. PAMI*, 28(4):650–656, 2006.