JACOBS
UNIVERSITY

School of Engineering and Science

Bachelor's Thesis

An extension of Felzenszwalb-Huttenlocher segmentation to 3D point clouds

Mihai-Cotizo Sima

May 2012

First supervisor: Prof. Dr. Andreas Nüchter

Abstract

This paper investigates the segmentation algorithm proposed by Felzenszwalb and Huttenlocher and its compatibility with 3D point clouds. In order to be able to use the algorithm, we adapt the range and intensity data to the smoothed graph structure used by the algorithm. We investigate the influence of the algorithm's parameters to its behavior and which representation methods that are meaningful to both the machines and the humans.

Contents

Chapter 1

Introduction

1.1 Overview

The three core components of human perception are grouping, constancy, and contrast effects [10]. Image segmentation approaches this natural way of observing the world by splitting images in components with constant attributes, and grouping them together to enhance their contrast.

In the digital world, the interface between robots and the environment is comprised of various sensors – out of which laser scans offer the most reliable replacement for vision. Laser range scans are a facile and effective way of producing point clouds, on top of which semantic maps can be created. Once the surroundings are mapped, several techniques can be used to enhance the robot's knowledge base and thus enable it to perform the assigned tasks. One important such technique is object recognition.

Using object recognition, the robot gains information about the type and position of the objects near it, and sequentially solve different operations [6]. (i.e. inspections and checks, movement coordination and navigation, or spatial positioning) Image segmentation is an optional intermediate step that optimizes the yield of object recognition and one of the algorithms that implements it and will be of interest in this work.

1.2 Scientific Contribution

The problem of segmentation is one of the main research areas of computer vision; many existent algorithms produce satisfying results based on different local characteristics of the images. Felzenszwalb and Huttenlocher propose a graph-based algorithm which, despite its greedy-approach, satisfies global properties by identifying not only constant intensity regions, but also regions of high variations or gradients [4].

We push forward this implementation, by switching from the 2D case to 3D point clouds, and investigating the behavior of the algorithm in this situation. We expect that, since this segmentation approach is based on graphs (which are inherently dimensionless), no impact on the performance will be observed. However, it is important to investigate the influence of the algorithm parameters, and whether there are other hidden constraints still to be added.

The results of this thesis will provide an efficient algorithm for 3D point cloud segmentation,

which can be used during object recognition, (i.e. faces, irises, fingerprints, medical affections) and machine vision in general.

Chapter 2

State of the Art

Image segmentation is the process through which a set of contours or segments are extracted from the features of an image. An important requirement is that the pixels in the interior of a region are similar with respect to a characteristic or property, but adjacent regions are kept significantly different.

There are several segmentation methods existent at the moment. One of the most basic ones involve thresholding a gray scale image, transforming it to a binary image based on the "clipping level". One efficient method of thresholding was designed by Otsu [9]. His algorithm splits the image into two classes (considered "foreground" and "background") based on a minimum intra-class variance:

$$\sigma_i^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t) \tag{2.1}$$

In (2.1), σ_i is called "internal variance", and represents a weighted sum (w_1 and w_2 of the areas of the two classes corresponding to the chosen thresholding level). This action is equivalent to maximizing the inter-class variance:

$$\sigma_e^2(t) = \sigma^2 - \sigma_i^2(t) = w_1(t)w_2(t)\left[\mu_1(t) - \mu_2(t)\right]^2 \tag{2.2}$$

In both (2.1) and (2.2), we used the class probability (sum of the probabilities of each intensity level) and the class mean defined as follows:

$$w_1(t) = \sum_{i=0}^{t} p(i) \tag{2.3}$$

$$\mu_1(t) = \sum_{i=0}^{t} p(i)x(i) \tag{2.4}$$

The algorithm follows linearly through the thresholding levels and updates the variance and the mean of each class at each step. Finally, it returns the t value for which the inter-class variance is maximized.

Otsu's method is regarded as a histogram-based method. Generally, this technique requires computing the histogram out of all the pixels in the image and then using its shape (local

maxima and minima) to produce clusters (depicted in Figure 2.1). Moreover, in order to refine clusters, one will recursively apply the method to each cluster.

(a) Initial image (b) Result of Otsu thresholding

Figure 2.1: Otsu's method: original image and result [8]

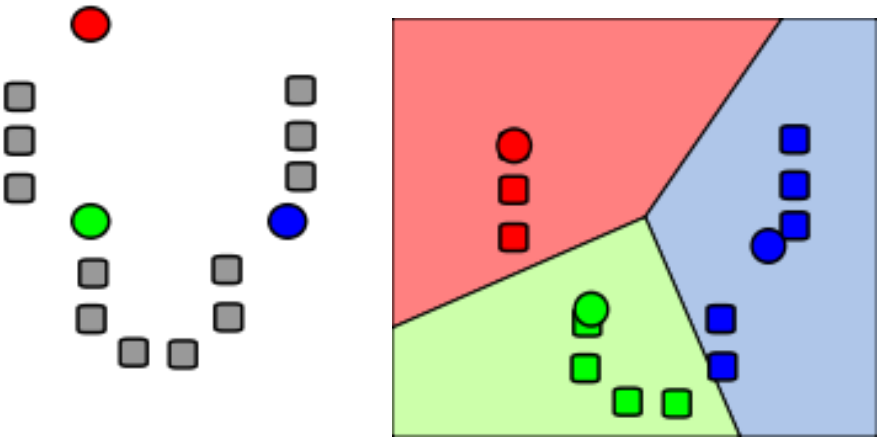
An important clustering algorithm is the k-means clustering. This approach is not limited only to 2D or 3D imagery, but rather to general feature-spaces. It divides n entities (observations) into k clusters such that each cell is a cluster around the nearest mean, as it is seen in Figure 2.2. Ultimately, this approach will converge to the Voronoi Diagram satisfying the properties of interest. The generic algorithm is:

Algorithm 1: k-means clustering

Input: n observation points, the number k of clusters
Output: A segmentation of the observation points into k clusters
1 Select k random seeds for the means;
2 **repeat**
3 Compute the Voronoi Diagram of the current means;
4 Assign each of the n observations to a cell of the Voronoi Diagram;
5 Update each mean with the centroid of the points within each cell;
6 **until** *convergence of the means*;

The standard k-means algorithm has no guarantee that it will converge. The choice of the initial seeds severely influences the termination of the algorithm and the quality of the output. A better choice of the seeds was proposed by David Arthur and Sergei Vassilvitskii [3]. Their variant of the algorithm, called "k-means ++", solves problems such as the one depicted below in Figure 2.3.

The observation points form a rectangle here, with the width larger than the height, and the initial seeds are at the midpoints of the width sides. The clustering splits the points by a horizontal line through the center of the rectangle, but this solution is sub-optimal. Their algorithm only changes the choice of the seeds (line 1 in Algorithm 1), leaving the rest of the classical k-means algorithm intact, as it is seen in Algorithm 2.



(a) Initial points (b) Result of k-means algorithm

Figure 2.2: k-means algorithm: original image and result [7]

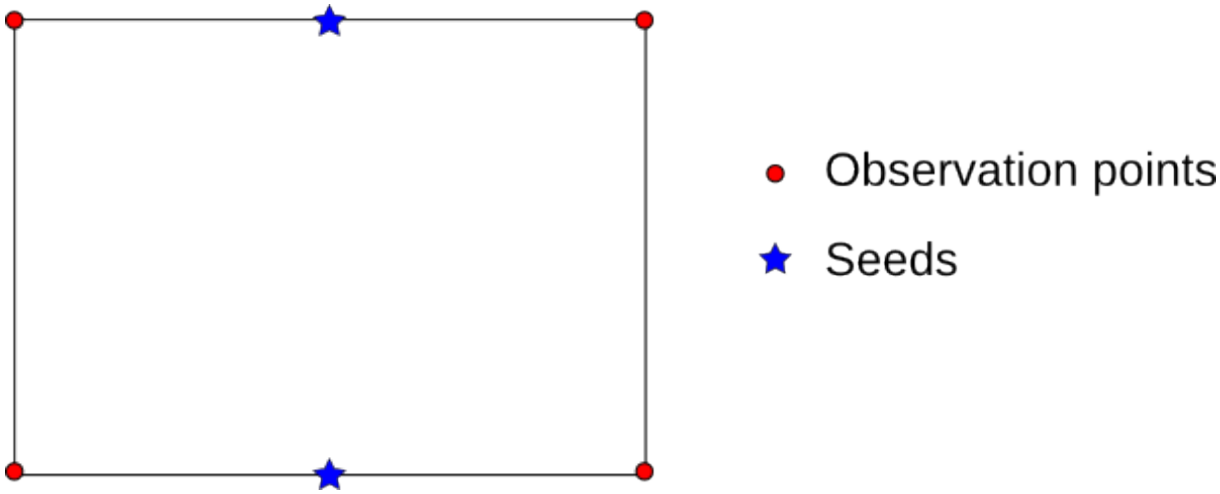


Figure 2.3: k-means seeding problem solved by k-means++

Algorithm 2: k-means ++

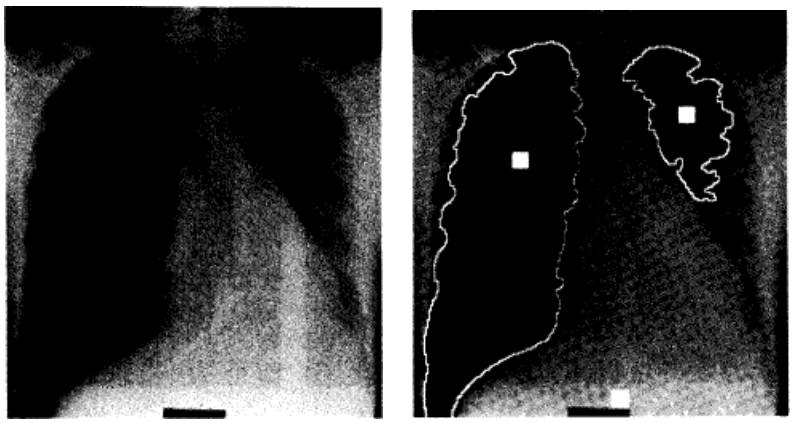
Input: n observation points, the number k of clusters
Output: k seed points for the initial means

1 Pick a starting seed at random from the n points;
2 **forall the** *seeds* s_i **do**
3 Compute $D(p_x)$, the minimum distance from point p_x to any already chosen seed;
4 Pick s_i randomly from a distribution where every point p_i has a probability proportional to $D(p_i)^2$;

There exist a set of compression-based methods based on the fact that segmentations take advantage of the connections between features. Therefore, they identify "patterns" that help compression, such that a good segmentation produces short compressions. For boundaries we observe that natural regions have smooth edges, and therefore smoother edges (which correspond to shorter encodings) are preferred. For textures, a lossy compression based on the entropies of multivariate distributions is used, in order to minimize the variance inside a region.

Segmentations are also achieved by following intensity transitions that are not smooth, as they should be in natural regions. These transitions are detected as edges; however, an important problem is that edges usually do not form closed shapes, and therefore additional algorithms connecting the edges are needed to produce segmentations.

A classic approach is growing regions from a set of seed points, by adding neighboring pixels to regions for which they are closest to the mean (Figure 2.4). Initially, the seed points were input by the user, to indicate the objects of interest in the image. An alternative method starts from a single region and, while picking pixels neighboring the frontier, decide if a new region is to be created based on whether their deviation is larger than a threshold value or not.



(a) Initial image (b) Result of applying region growing

Figure 2.4: Region growing algorithm applied to an X-ray image of human torso, with the initial seeds shown as white squares [1].

Graph-based segmentation methods generate a graph based on the pixels of the image and their properties, with the weights of edges connecting neighboring pixels proportional to the level of similarity of the two pixels. These nodes are then clustered based on a similarity criterion.

One method was discussed by Shi and Malik in [11]. Their segmentation method searches for minimal cut by splitting the graph in k sub-graphs such that the maximum of the cuts is minimum. This approach has the deficit of favoring small isolated sets (the more edges are cut, the higher the cost of the cut). To fix this, the authors propose a normalized cut with the following definition:

$$NCut(A,B) = \frac{Cut(A,B)}{Assoc(A,V)} + \frac{Cut(A,B)}{Assoc(B,V)} \tag{2.5}$$

In equation (2.5), the measures Cut and $Assoc$ are defined as follows:

$$Cut(A,B) = \sum_{u \in A, v \in B} w(u,v) \tag{2.6}$$

$$Assoc(A,V) = \sum_{u \in A, t \in V} w(u,t) \tag{2.7}$$

This cut-based approach is effective in preserving the non-local properties of the image; however, it focuses only on the characteristics of a cut rather than of the whole segmentation [4].

Grady uses a random walk algorithm in [5], for producing segmentations based on k user-defined seeds. The central point of his approach is the labeling of each non-seed pixel with respect to the question "Given a random walker, starting at this pixel of interest, what is the probability that it reaches each seed point?", as it is seen in Figure 2.5. By assigning each point the label of the seed which has the largest probability, a pure-noise or constant-level image will produce approximately the Voronoi Diagram of the seeds. The weights on the edges of the graph help to bias the walker not to cross sharp intensity gradients or any other features that should be preserved. One example of such weight function is the Gaussian function of the intensity or color values of the points (g_i and g_j in equation (2.8)).

$$w_{i,j} = \exp\left(-\beta \|g_i - g_j\|^2\right) \tag{2.8}$$

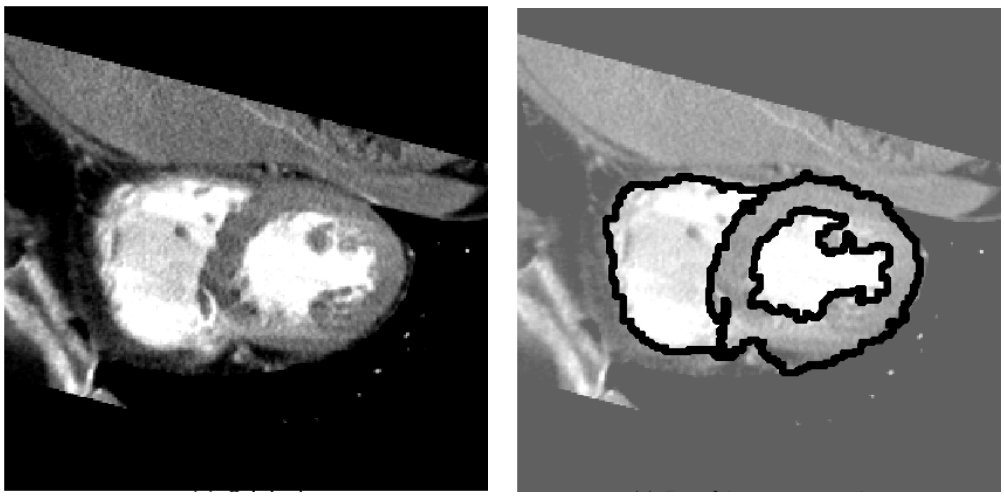
A classical approach to treat random walks is to assign a physical model – which places an electric charge that generates a potential of 1 sequentially in each seed, and an electric "sink" (potential 0) in all the other seeds. Then, the value of the electric potential at each other non-seed point is determined by solving a system of linear sparse equations, also using the property that the sum of probabilities at each point will always be 1.

Zahn considers a more general case in [12], such that only distances between points in an n-dimensional feature space are used produce segmentations of members with different properties (such as disease symptoms). His approach is based on the Gestalt hypothesis that humans group image components such that the psychological encoding is lowered for less neurological stress. His method pursues groups that are homogeneous – with a small standard deviation σ of the lowest distances between points of a same group – and well defined – with a minimum distance between the elements of two distinct groups much larger than the mean distance μ inside a group.

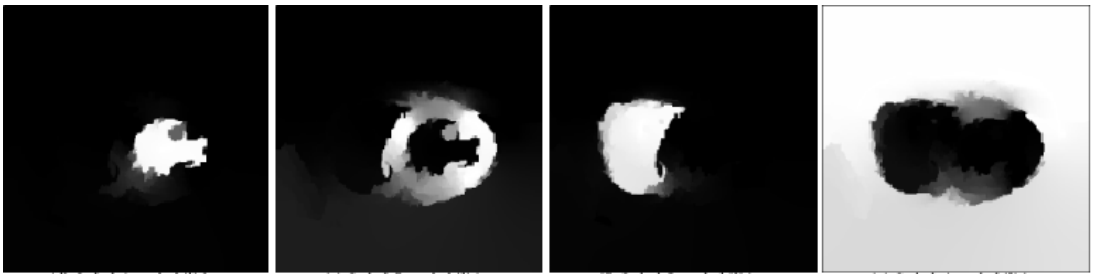
For implementing reliable segmentations, he presents solutions for clusters with constant densities (connect points at a distance lower than $\mu + 3\sigma$ and compute the connected components), clusters with smoothly varying densities (connect the 3-nearest neighbors) or clusters connected by topological "bottlenecks" (connect k-nearest neighbors and detect bridges in the graph, in order to cut the corresponding links).

He also introduces a solution using minimum-spanning trees – which segments a graph in the observable clusters given an appropriate criterion for cutting edges (it separates clusters without breaking their structure). One such criterion is that the weight of the edge to be cut is much larger than the standard deviation of the edges connected to the nodes neighboring the edge of interest.

Felzenszwalb and Huttenlocher proposed in [4] an approach that can be extended to general feature spaces and not only treats the local properties, but also the global characteristics of an



(a) Initial image (b) Resulting segmentation



(c) Assigned probabilities for each region

Figure 2.5: Initial and resulting image for Grady's random walk algorithm proposed in [5]

image. They use the weight of an edge $w(v_i, v_j)$ between two vertices as the level of dissimilarity of two points. (difference in intensity, range, color, location etc) A segmentation splits nodes such that edges inside a component have relatively low weights and edge between components have relatively high weights.

For every component, the internal difference is defined as the maximum edge weight in the minimum-spanning tree (MST) of the component:

$$Int(C) = \max_{e \in MST(C)} w(e) \tag{2.9}$$

The external difference is, analogously, the minimum weight of an edge connecting a vertex from one component and a vertex from another component:

$$Dif(C_1, C_2) = \min_{v_1 \in C_1, v_2 \in C_2, (v_1, v_2) \in E} w((v_1, v_2)) \tag{2.10}$$

We can replace the condition $(v_1, v_2) \in E$ in equation (2.10) with a constraint on the weights of inexistent edges, such that $w((v_1, v_2)) = \infty$ if $(v_1, v_2) \notin E$. Also, an alternative definition for

equation (2.10) uses the median weight and not the smallest one, in order to make it robust to outliers.

A thresholding function is used to make two components distinct based on their sizes: more exactly, for small components, a stronger evidence for a boundary is needed for the components to be distinct. The function is defined as:

$$\tau(C) = \frac{k}{|C|} \tag{2.11}$$

The parameter k sets preference for large or small components, depending on whether its value is large or small. However, it does not represent a minimum component size.

Based on these quantiles, the minimum internal difference of two components is defined as:

$$MInt(C_1, C_2) = \min\{Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)\} \tag{2.12}$$

Furthermore, the central predicate, which indicates whether there is evidence of a boundary between two components, is defined as:

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases} \tag{2.13}$$

In order to decide upon the effectiveness of the segmentation algorithm, we must know whether it is too coarse or too fine:

1. a segmentation is too fine if there exists a pair of regions for which there is no evidence of a boundary between them

2. a segmentation is too coarse if there exists a refinement of the segmentation which is not too fine

It can be proven that, for any graph, a segmentation exists such that it is neither too fine, nor too coarse. The proposed algorithm that constructs the segmentation is presented in Algorithm 3.

Algorithm 3: FH Segmentation

```
Sort all the edges according to their weight;
Assign each vertex to its own component for the initial segmentation S^0;
forall the edges e_i = (v_1, v_2) ∈ E do
    Consider v_1 ∈ C_1 and v_2 ∈ C_2 (the two distinct components containing the vertices of
    the edge, as they exist in the previous step S^{i-1});
    if w(e_i) ≤ MInt(C_1, C_2) then
        Merge C_1 and C_2 in S^i;
    else
        S^i = S^{i-1};
```

The algorithm can be extended to nearest neighbor graphs by mapping each pixel in the image to a point in the feature space (e.g. (x, y, r, g, b) or (x, y, z, i)) and by using the Euclidean distances L_2 as edge weights. Each point should be connected to a limited number of neighbors (e.g. via nearest k neighbors or the neighbors in a radius of r) for keeping the run-time low. Also, to further refine the quality of the segmentation, one takes into consideration both the reflectance (intensity) of the point, and the distance from the origin (range). Following the method proposed in [4], the segmentation algorithm is run twice (once for each image), and then the two resulting images are combined into one third segmentation, where two points are in the same region only if they are in the same regions for both the original segmentations. Another approach creates and uses a (possibly weighted) metric involving both the reflectance and the range - for example a weighted Euclidean distance.

Chapter 3

Proposed approach

The Laser scanner records the coordinates (x, y, z) and, optionally, the intensity (reflectance) of every point in the cloud. The points are converted to a feature space that uses spherical coordinates, producing intensity images (φ, θ, i) or range images (φ, θ, r).

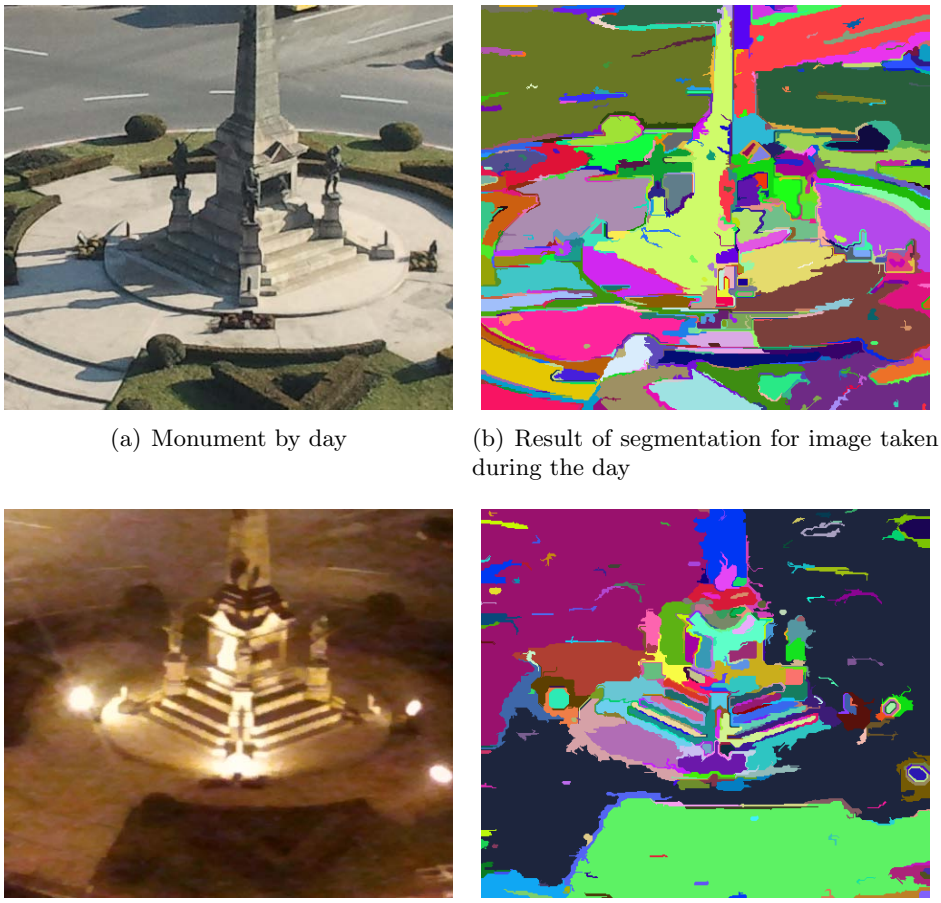
From this starting point, we write a program that connects the RXP scan reader to the implementation distributed by the authors of the paper. The segmentation of the range images will be converted back to point clouds (x, y, z) and visualized with the "show" utility of the 3DTK.

We effectuate a preliminary investigation on the general behavior of the parameters on the result of the algorithm. The third parameter used in the original implementation (minimum accepted size of a segment) represents a post-processing way to eliminate "noisy" small regions (any region of area smaller than this parameter is merged with the surroundings).

We start the approach with two pictures of a monument in Ploiesti, taken during the day and the night (Figures 3.1(a) and 3.1(c)). This is meant to test the behavior of the implementation on noisy images (the night image has a high ISO value and hence more noise). The segmentations were produced using a Gaussian smoothing factor $\sigma = 1$, the constant $k = 150$ and a minimum resulting component size of $min = 20$. (Figures 3.1(b) and 3.1(d)) We observe that these parameters induce two segmentations where the general shape of the monument and of the soldiers are consistent.

Another set of images tested were produced by extracting range images from a medium-sized 3D scan. The code to produce the image is embedded in the RXP file scanner such that it outputs a PPM file (Figures 3.2(a) and 3.2(c)). Then, the segmentations are produced using $\sigma = 2$, $k = 300$ and $min = 0$. The results are observed in Figures 3.2(b) and 3.2(d). We observe the bad influence that the parameter $min = 0$ has – many noisy small regions appear in the segmentations. Also, depending on the application of the segmentation, the lack of differentiation between the objects on the ground and the floor (see Figures 3.2(a) and 3.2(b)) or between the floor, the walls and the ceiling (Figures 3.2(c) and 3.2(d)) might be undesirable. However, these problems can addressed by using the combination of range and intensity image segmentations, as discussed in the last chapter.

The next step in the implementation of the algorithm is to load the point cloud in memory and then convert it to a range image. This image is passed to the implementation provided by



(a) Monument by day

(b) Result of segmentation for image taken during the day

(c) Monument by night

(d) Result of segmentation for image taken during the night

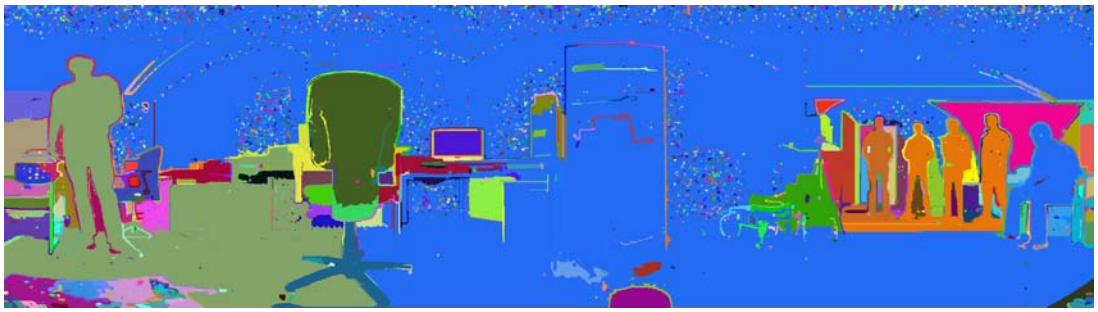
Figure 3.1: Images of a monument in Ploiesti taken during day and night, and the associated segmentations

Felzenszwalb and Huttenlocher. After the segmented image is obtained, the point clouds are reconstructed by retrieving the original 3D point corresponding to each pixel of a segment and outputting it to the appropriate file associated to the segment (Algorithm 4). A first observation to this algorithm are the unnecessary steps of converting the point cloud to a range image and then reconstructing the point clouds from the resulting segmented image (Table 3.1).

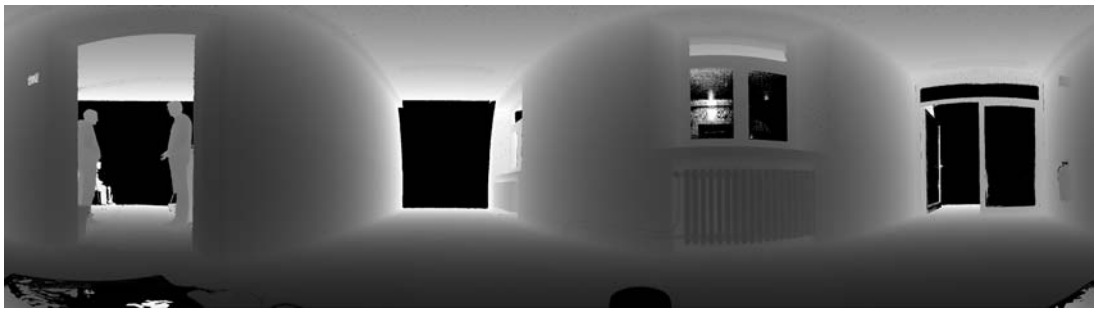
The results of this implementation are comparable to the ones obtained when using the segmentation program directly on PPM images (see Figure 3.3). Also, the segmented clouds can be visualized using the "show" application (Figure 3.4).

In order to improve the quality of the segmented images, we proceed to use the reflectance information in the blue channel of the images that were segmented using the FH algorithm, while keeping the range information in the red and green channels. This solution produces a better separation of the objects (see Figure 3.5).

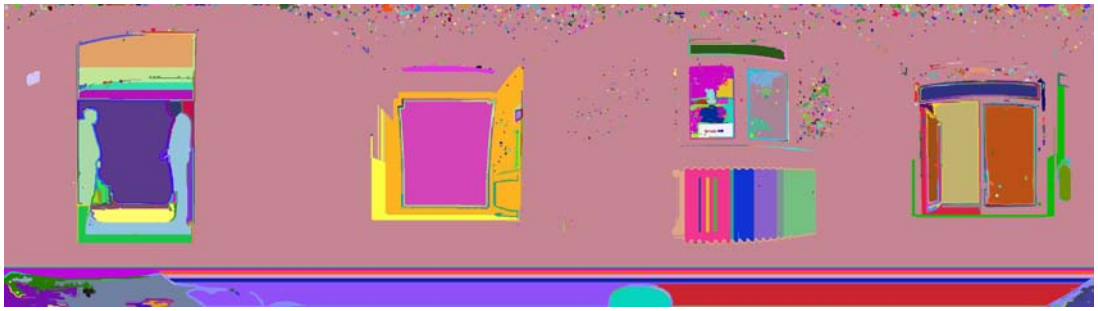
(a) First range image



(b) Segmentation of the first image



(c) Second range image



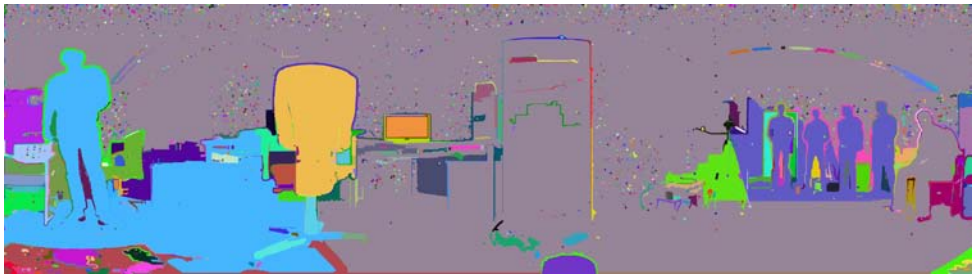
(d) Segmentation of the second image

Figure 3.2: Range images and their segmentations

Algorithm 4: Initial implementation

Input: the point cloud from scanner
Output: segmented point cloud
1 Read the initial points;
2 Convert the 3D points to a range image;
3 Adapt image to FH format;
4 Apply FH segmentation algorithm;
5 Prepare a vector with the original 3D points for each resulting segment;
6 Save point clouds individually to files;

(a) Range image testing first implementation



(b) Result of applying the first implementation to the range image

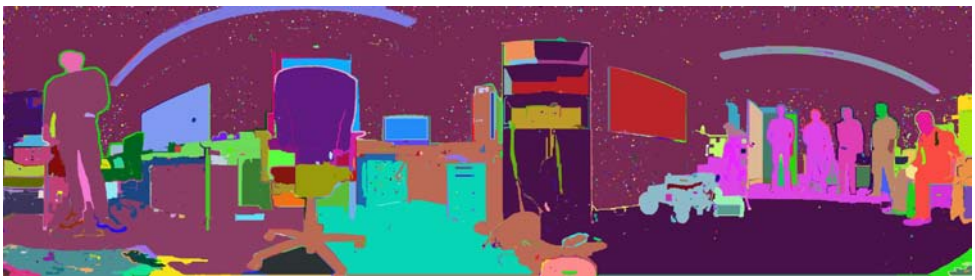
Figure 3.3: The range image and its segmentation using the first implementation

Figure 3.4: Screenshots of the show program displaying the segmented point clouds

	Measurement 1	Measurement 2	Measurement 3	Average
Read	4506ms	4725ms	5118ms	4783ms
Convert	2944ms	3086ms	2857ms	2962ms
Adapt	210ms	223ms	208ms	213ms
Segment	7569ms	7587ms	7538ms	7564ms
Prepare	1378ms	1322ms	1334ms	1344ms
Save	40290ms	40061ms	39704ms	40018ms

Table 3.1: Measurements of run time for each step described in Algorithm 4

(a) The combination of reflectance and range data



(b) Result of applying the first implementation to the image

Figure 3.5: Results of combining range data (in red and green channels) with reflectance data (in blue channel)

In order to fully use the 3D information stored in point clouds, we handle the creation of the graphs processed by the segmentation algorithm: each point represents a graph vertex, and for the edges we connect each pixel to its k nearest neighbors (the other option is to connect a pixel to its neighbors in spherical coordinates, but this approach was already evaluated at the previous step). Each edge is weighted by the distance between points and by difference in reflectance (equation (3.1)).

$$w((a,b)) = \|\vec{a} - \vec{b}\| + |i(a) - i(b)| \tag{3.1}$$

The 3DTK library, which is used to load the points from the data files, also provides the approximate k nearest neighbors (aKNN) library [2]. library provides algorithms for computing

the k nearest neighbors of a vertex, searching all the points in a given radius from a vertex, and also approximate variants of these algorithms. Both the exact and the approximate searches are appropriate for this case, in which each point must be connected to some other points with which it shares characteristics or not. (edges in the former category reinforce segment cohesion, while edges in the latter category augment the inter-component distinction) The user can provide the number of neighbors k and, optionally, the radius r as parameters: in the former case, the k nearest neighbors, and in the latter, at most k random neighbors which are at a distance smaller than r are used. We must emphasize that the former method is efficient for dense images, while the latter is appropriate for sparser images. (i.e. Velodyne scans)

The original implementation of the Felzenszwalb-Huttenlocher uses Gaussian smoothing of the input image in order to also smooth the weights of the graph edges. In our case, the most suitable approach is to directly apply Gaussian smoothing to the edges resulting after the aKNN searches. Each edge is smoothed using the Algorithm 5. The following notations are used: $G(x; \mu, \sigma)$ is the Gaussian function of mean μ and standard deviation σ applied to value x, η is the normalization factor and w'_e is the resulting weight. Furthermore, the algorithm is graphically explained in Figure 3.6.

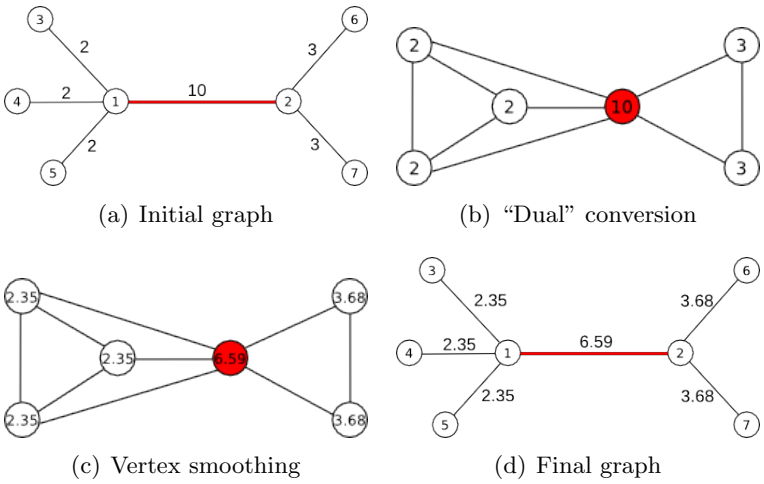
Algorithm 5: Gaussian smoothing of an edge

Input: edge $e = (a, b)$ of weight w_e, σ for Gaussian smoothing
Output: smoothed weight w'_e

$\mathcal{L} = \{\}$;
forall the *edges* $e' = (a', b')$ *with weight* $w_{e'}$ **do**
  if $a' \in \{a, b\}$ *or* $b' \in \{a, b\}$ **then**
    $\mathcal{L} = \mathcal{L} \cup \{(w_{e'}, G(w_{e'}; \mu = w_e, \sigma))\}$;
$\eta = \sum_{i \in \mathcal{L}, i=(w,\alpha)} \alpha$;
$w'_e = \frac{1}{\eta} \sum_{i \in \mathcal{L}, i=(w,\alpha)} w \cdot \alpha$;

After the Gaussian smoothing of the graph weights, the segmentation algorithm can proceed without any modification. However, the necessary final step is to iterate through all the resulting segments and merge those which are smaller than the minimum component size. After this post-processing step, the point clouds can be grouped according to the result and then further used, for example for object identification. Our application will store all the points (including their reflectance values) to a directory, to be visualized using the 3DTK show application. A final investigation of the run times per processing stage can be found in table 3.2, and a list with all the parameters used in the algorithm, together with a comment on their influence can be found in 3.3. Also, an overview of the final algorithm can be found in Algorithm 6.

Running the program revealed that using the 3DTK components (for reading the points and creating the aKNN structures) consume around 3GB of RAM for around 12 million points. However, some outdoors scenes involve a lot more points (18 million or even more, depending on the duration of acquisition of the scan). However, an effective trade-off between segmentation quality and memory usage is to split the scans in several groups, depending on the distance from



(a) Initial graph (b) "Dual" conversion

(c) Vertex smoothing (d) Final graph

Figure 3.6: The four steps of applying Gaussian smoothing. (a) a possible initial graph; the edge of interest is in red. (b) the "dual" graph, obtained by converting each edge to a vertex, and by connecting each edge to all the other edges with which it shares a vertex; here, the vertices are assigned the weights of the original edges, with the vertex corresponding to the edge of interest in red. (c) the result of applying Gaussian smoothing with $\sigma = 4$ to the values of the vertices; only the direct neighbors of each vertex have an influence. (d) the reconstructed graph; in red, the original edge of interest has a "smoothed" value.

	Measurement 1	Measurement 2	Measurement 3	Average
Read	6673ms	6801ms	7124ms	6866ms
Compute neighbors	42442ms	41846ms	43887ms	42725ms
Gaussian smoothing	21107ms	28044ms	21867ms	23672ms
Segment	11414ms	11552ms	11200ms	11389ms
Post-processing	2971ms	2847ms	2597ms	2805ms
Prepare	805ms	1221ms	1107ms	1044ms
Save	78260ms	81871ms	74240ms	78123ms

Table 3.2: Measurements of run time for each step described in Algorithm 6

Algorithm 6: The final segmentation algorithm

Input: a point cloud
Output: the segmented cloud

1 Construct the aKNN tree (initialize structure);
2 For each point, compute the neighbors using aKNN search or radius search;
3 Create a graph based on the computed neighbors, with the edge weights produced by the distance function;
4 Smooth the edges using Gaussian smoothing (Algorithm 5);
5 Apply the Felzenszwalb-Huttenlocher segmentation algorithm to the graph;
6 Merge small segments until the minimum segment size is satisfied;

σ	the smoothing factor used in the Gaussian function
K	the size factor, which will increase the segmentation algorithm's preference towards bigger components
I	the minimum segment size
N	the (maximum) number of neighbors to be returned by the aKNN search
R	optional, the radius used in searching for neighbors
ϵ	the error accepted in the aKNN methods

Table 3.3: The parameters that influence Algorithm 6

the scanner, and then to segment each group individually. An effective strategy is to split at 15 meters: this results in a balance of around 11-12 million points in the closer zone and 6-8 million points in the farther zone; moreover, objects of interest are usually closer than 15 meters, while background objects (i.e. trees) are found farther.

The 3DTK point cloud loading library already supports parameters for limits of the distance between the scanner and the loaded points, which can be used to segment the "closer" zone and the "farther" zone separately. The results are output in different directories, or can be passed directly to different systems such as the object recognition algorithms. However, for visualizing the segmentations, we use a different program that produces range images based on the segments from several directories. The approach is to color the pixels in "farther" zone first, and then the pixels in the "closer" zone, in an algorithm similar to the Painter's algorithm; this will ensure that close objects are not overwritten by the background. Some additional features of this second program include pixel dilation (useful for sparse scans), and overlaying the intensity information (by modulating the V-channel when using the HSV color model at output).

3.1 Results

The first analyzed data set is the medium-density scan taken in the lab. (Figure 3.7) An improvement in the level of detail of the segmentation can be observed with respect to the previous versions (Figure 3.2(b) and Figure 3.5(b)): the walls are clearly distinct from the floor, the squares on the checkerboard are observable, and the human silhouettes are nearly always separated from the floor. Figure also displays the effect of the reconstruction program. (i.e. overlaying the intensity data by modulating color intensity)

The small lab data set (horizontally sparse) requires the usage of radius search, instead of the k-nearest neighbors search. Despite the fact that we randomly pick 4 neighbors within the radius, this type of search provides good chances of connecting points which are horizontally farther. The result of this strategy is a homogeneous segmentation, with correctly segmented walls and silhouettes. (Figure 3.8(a)) The sparsity of the data set and the effect of dilated points can be observed respectively in Figure 3.8(b) and Figure 3.8(c).

Several outdoor scans were made on the campus of Jacobs University. These data sets contain various challenges, such as occlusions, (i.e. trees) or the high number of points. (which required the usage of "close" and "far" zones technique) Figure 3.9 contains a scan made in

Figure 3.7: Spherical reflectance image and its segmentation, using $\sigma = 0.2$, $K = 50$, $I = 30$, and 8 nearest neighbors for every point.

Figure 3.8: Segmented spherical image of the small data set, with a sparse and diluted detail. Parameters used: $\sigma = 1$, $K = 10$, $I = 40$.

front of Research 1 building: it can be observed that the cars are segmented, in a case even the number plate is in its own segment. Also, the background trees are segmented in their components, while the foreground trees (i.e. the closest tree) has each branch segmented in a different component; this behavior can be assigned to the lower variance (in both distance and intensity) of the points corresponding to background trees.

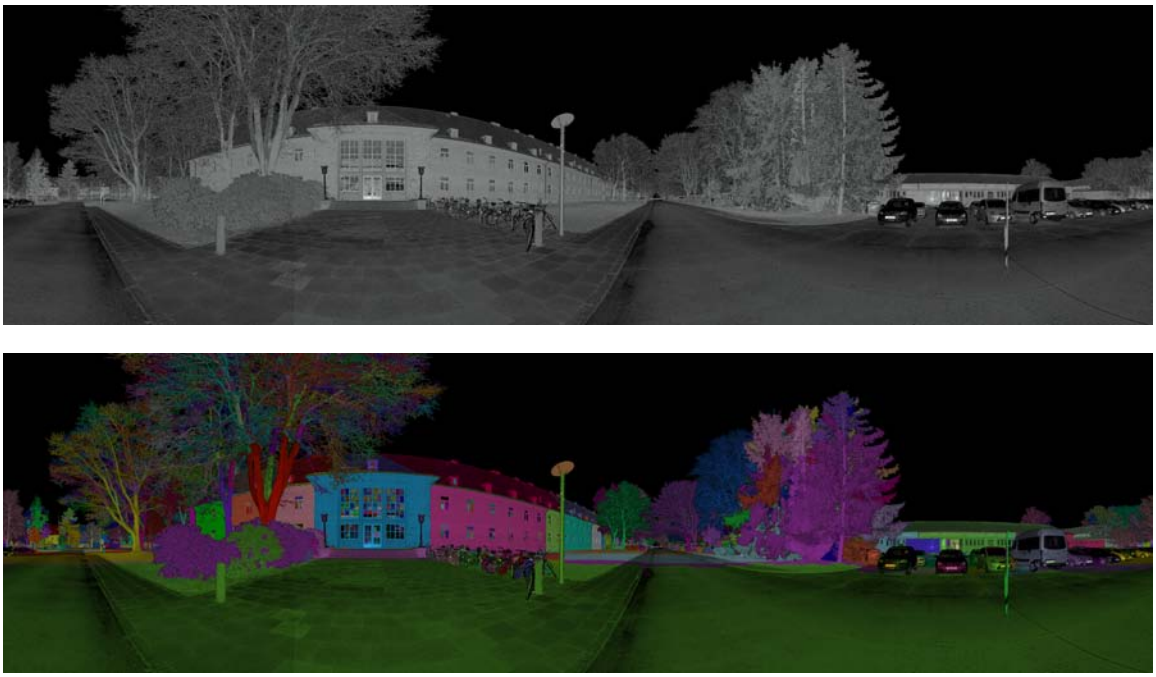


Figure 3.9: Spherical intensity image of the Research 1 building and its segmentation. Parameters used: $\sigma = 2$, $K = 300$, $I = 100$.

The plate of the blue car was also segmented away in Figure 3.10; this behavior suggests a possible successful application of the algorithm for the Police or other traffic supervision institutions. The car also has the the interior and a head light segmented, so another application can be in car model recognition: the interior of the car can be removed such that only the chassis is searched in a database.

The background trees are also segmented in Figure 3.11, while the foreground tree (right) has bigger branches in different components. However, the oversegmentation of the foreground tree is not as pronounced as in 3.9 because the tree is far from the scanner.

A tree is partly occluding a building in Figure 3.12, in a scan taken from a point placed to the lower-left relatively to the viewpoint. However, it can be observed that some occlusions are not big enough to separate segments. (for example the dark green segment in the upper-center part) A radius search was used in order to obtain such an image, which suggests that a higher radius could make the entire roof of the building belong to be not so oversegmented. The image was obtained using the "show" application of the 3DTK, with dilated points; the application has a feature which hides a percentage of points, until the rendering framerate is high enough, which is why some of the points on the ground are missing.

Figure 3.10: The campus of Jacobs University. In the foreground, a car is correctly segmented. Parameters used: $\sigma = 2$, $K = 300$, $I = 100$.

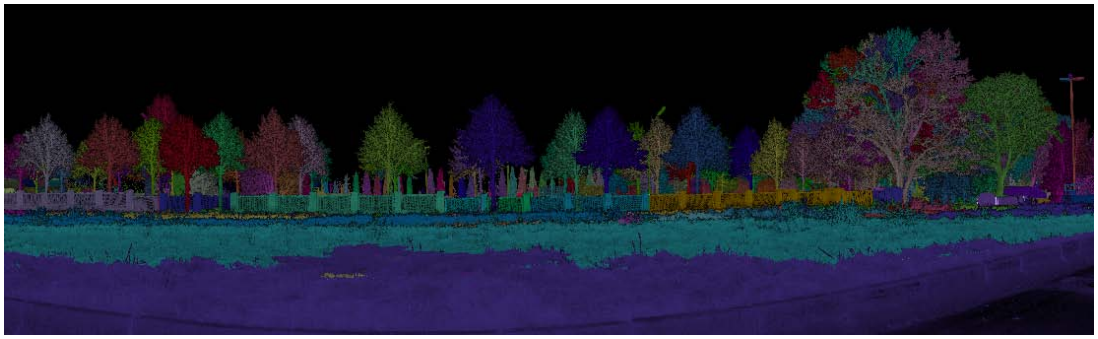


Figure 3.11: The campus of Jacobs University. Distant trees are segmented correctly. Parameters used: $\sigma = 2$, $K = 300$, $I = 100$.

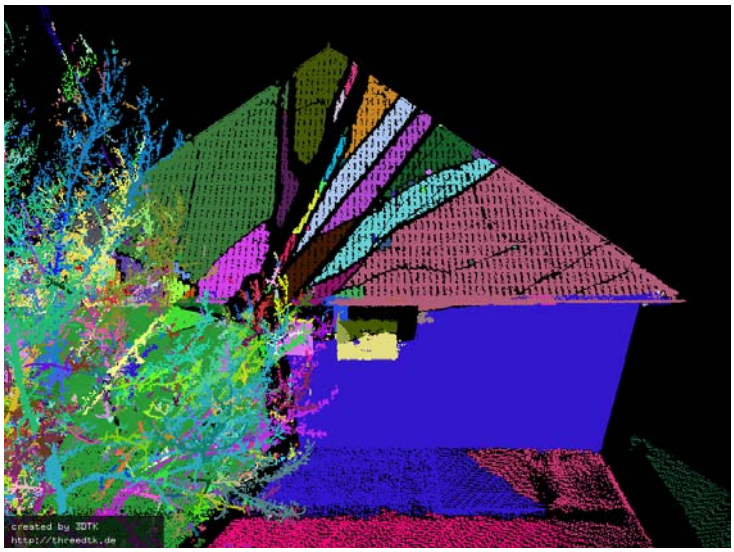


Figure 3.12: A building on campus, occluded by a tree. Despite the occlusions, the rooftop is not oversegmented. Parameters used: $\sigma = 2$, $K = 300$, $I = 100$.

Lastly, a different format (Velodyne) was used to test the algorithm. Velodyne produces a vertically-sparse image, and the radius search does not overcome this problem entirely. However, some cars are segmented nearly entirely in Figure 3.13. The poor results are partly due to the lack of reflectance data, (limitation of the scan) or the low radius used in the search.

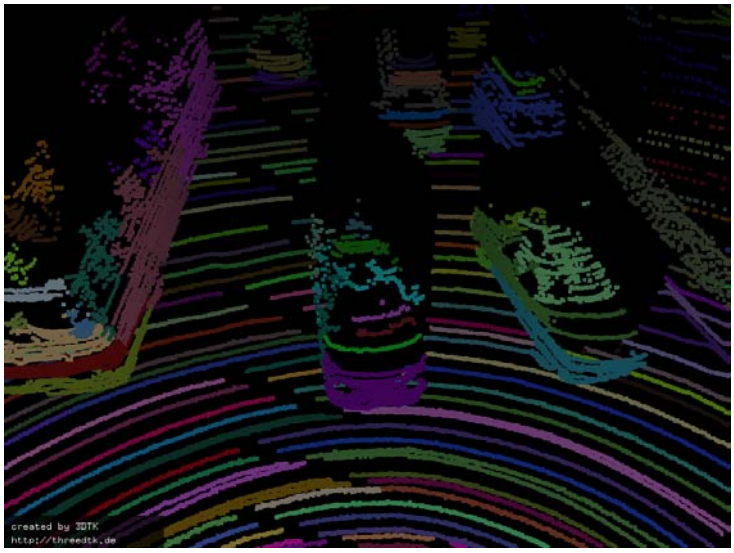


Figure 3.13: Segmented Velodyne data set, visualized using the "show" application. Parameters used: $\sigma = 1$, $K = 100$, $I = 30$.

Chapter 4

Conclusions

This work proved that the Felzenszwalb-Huttenlocher segmentation algorithm can be safely applied to 3D point clouds. The range information (position of the points) produced good segmentations, but also including reflectance information to the data set increased the fidelity of the segmentation. In order to refine the result, an extension of the Gaussian smoothing to graph edges was designed. Finally, the influence of the parameters to the behavior of the algorithm were outlined. Possible applications of this work include object recognition and image registration and car plate recognition for traffic supervision.

4.1 Future work

An improved graph generation algorithm should be designed. Currently, the radius search picks k random nodes from all the points returned; however, a better approach is to use $\alpha \cdot k$ points determined via k-nearest neighbors search, and the rest of $(1 - \alpha) \cdot k$ points determined via radius search. (α becomes a new parameter for the algorithm)

Another research focus point is the edge weighting function. Currently, the function is defined as in equation (3.1). An improved version of this function uses normalized distance and intensity. (i.e. the distance is divided by the average distance, and the intensity interval is converted to $[0, 1]$)

Finally, segmenting several zones separately (such as "close" and "far" zones) enhances memory consumption and can be parallelized. However, stitching together the resulting segments does not produce accurate results, and a better zone treatment approach such as overlapping segmentation zones is to be investigated.

Bibliography

[1] R. Adams and L. Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, June 1994.

[2] Approximate k-nearest neighbors. `http://www.cs.umd.edu/~mount/ANN/`, May 2012.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. *ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[4] P. Felzenszwalb and D. Huttenlocher. Efficient graph–based image segmentation. *International Journal of Computer Vision*, 59(2), September 2004.

[5] Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006.

[6] William E.L. Grimson. *Object Recognition by Computer*. MIT Press, 1991.

[7] *k*-means clustering. Images retrieved from `http://en.wikipedia.org/wiki/Kmeans`, January 2012.

[8] Otsu's method. Images retrieved from `http://en.wikipedia.org/wiki/Otsu's_method`, January 2012.

[9] Nobuyuki Otsu. A threshold selection method for gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 1979.

[10] Mary A. Peterson and Gillian Rhodes. Perception of faces, objects, and scenes : analytic and holistic processes, 2003.

[11] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 2000.

[12] C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, 1971.

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Bremen, May 2012