Line-Based Registration of Photogrammetric Point Clouds with 3D City Models by means of Mixed Integer Linear Programming

This is a self-archived version of a paper that appeared in the Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP) - Volume 4: VISAPP, 2018, 299–306

Steffen Goebbels[1] and Regina Pohle-Fröhlich[1]

[1]*Institute for Pattern Recognition, Faculty of Electrical Engineering and Computer Science, Niederrhein University of Applied Sciences, Reinarzstr. 49, 47805 Krefeld, Germany*
{*Steffen.Goebbels, Regina.Pohle*}*@hsnr.de*

Keywords: Point Cloud Registration, Linear Programming, Structure from Motion, Building Reconstruction, CityGML

Abstract: This paper describes a method to align photogrammetric point clouds with CityGML 3D city models. Amongst others, we use photogrammetric point clouds that are generated from videos taken from the driver's perspective of a car. Clouds are computed with the Structure-from-Motion algorithm. We detect wall planes to rotate these clouds so that walls become vertical. This allows us to find buildings' footprints by accumulating points that are orthogonally projected to the ground. Thus, the main alignment step can be performed in 2D. To this end, we match detected footprints with corresponding footprints of CityGML models in a x-y-plane based on line segments. These line segments are detected using a probabilistic Hough transform. Then we apply a Mixed Integer Linear Program to find a maximum number of matching line segment pairs. Using a Linear Program, we optimize a rigid affine transformation to align the lines of these pairs. Finally, we use height information along CityGML terrain intersection lines to estimate scaling and translation in z-direction. By combining the results, we obtain an affine mapping that aligns the point cloud with the city model. Linear Programming is not widely applied to registration problems; however the technique presented is a fast alternative to Iterative Closest Point algorithms that align photogrammetric point clouds with clouds sampled from city models.

1 INTRODUCTION

Virtual 3D city models are used for simulations (eg noise maps, lighting models, solar potential analyzes, flood maps, heat requirement mapping) as well as for planning purposes like Building Information Modeling (BIM). CityGML is the XML based description standard for city models. It offers a concept of Level of Detail (LoD), see (Gröger et al., 2012) Most models currently are given in LoD 2 with defined roof facets and walls but without detailed facade information. Window and door objects belong to LoD 3. Facade models do not only serve for visualization or planning purposes. For example, gateways and routes for rescue workers are very interesting in smart-city applications. Whereas airborne laserscanning point clouds are provided as open data in our country, they only show roofs but no facades. Facade data have to be acquired individually. An inexpensive way to obtain photogrammetric point clouds of facades from videos or overlapping photos is to use the Structure-from-Motion algorithm (SfM).

We generate dense point clouds from videos taken either through the front window of a car or by UAVs with the SfM tool Agisoft Photoscan, see Figure 1. The tool delivers a cloud of colored points, estimated camera position and camera parameters for video frames, and a textured mesh.

It is difficult to do a manual registration of photogrammetric point clouds with sufficient precision to cleanly project points or textured meshes to CityGML walls. Therefore, we apply an automatic precise registration of the point cloud with the city model. As a preliminary, the cloud has to be coarsely registered with UTM coordinates, either manually or using GPS/GLONASS information. The next section discusses some approaches to do such a precise registration. In Sections 3–5 we in detail describe our approach to match line segments by solving a combinatorial problem with fast Linear and Mixed Integer Linear Programs. Section 6 summarizes results.

2 VARIOUS REGISTRATION APPROACHES

There exists a large variety of point cloud registration algorithms, see (Maiseli et al., 2017). Most

of them can be classified as non-feature- or feature-based. However, our input consist of only one (colored) point cloud. The city model basically is a set of polygons. The easiest way to apply non-feature-based

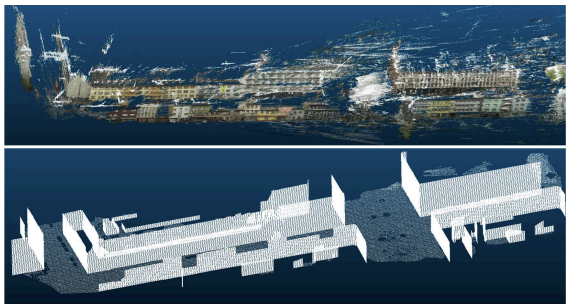

Figure 1: SfM point cloud, and cloud sampled from CityGML walls and merged with terrain points from airborne laserscanning

algorithms is to sample a point cloud from the city model. We work with 100 points/m^2. Experiments showed that increased sampling rates only do have minor influence on registration results. In contrast to simultaneous localization and mapping (SLAM) applications, the point clouds might be very different: The photogrammetric point cloud covers ground and vegetation whereas our city models do not. With regard to the ground, the problem can be healed by merging the sampled model cloud with a filtered cloud from airborne laserscanning that only consists of last pulse terrain points, see see Figure 1. Another problem is that buildings are modeled separately in CityGML. Thus, there exist walls between neighboring buildings that are not visible from the outside. Therefore, we only consider model walls for sampling if they are visible from outside the buildings. We also limit the sampled scene roughly to the area that is covered by the photogrammetric point cloud.

The standard means to align point clouds is the non-feature-based Iterative Closest Point (ICP) algorithm, see (Rusinkiewicz and Levoy, 2001).

We did several experiments with Point Cloud Library's ICP implementations (version 1.8.0) in point-to-point and in point-to-plane mode (see (Holz et al., 2015) and tutorials on `http://pointclouds.org/documentation/tutorials`), both with a maximum correspondence distance of 5m that fits with data. Whereas the base implementation of point-to-point ICP might converge very slowly or get stuck in a sub-optimum, ICP based on Levenberg-Marquard optimization and point-to-plane ICP find good solutions. In point-to-plain mode, distances are not measured between points but between points and planes defined by estimated local normal vectors (see (Chen and Medioni, 1992)). This allows for sliding along wall planes and is better suited to match segments of planes like facades. Whereas the Levenberg-Marquard based point-to-point algorithm takes 4507 seconds[1] to align the clouds shown in Figure 1, point-to-plane ICP finishes in 719 seconds. However, our proposed method terminates in less than 30 seconds (see St. Anton street scenario in Table 1).

A somewhat related approach is the Normal Distribution Transform (NDT). Its idea is to replace all cloud points within a grid cell of the target cloud by a normal distribution that describes the probability of finding a point at a certain position. Instead of matching single points to each other, the probability of points being at the right place (with regard to the target cloud) can be optimized with Newton iterations, see (Magnusson et al., 2009). This technique eliminates the time of ICP's nearest neighbor search. With a resolution parameter set to 5 and a step size parameter set to 2.5 Point Cloud Library's algorithm aligns the two previously investigated clouds in 396 seconds. For chosen settings, this indeed is faster than ICP in point-to-plane mode but still significant slower than our proposed method. In our scenario, success and running time of NDT heavily depend on choice of parameters. For example, the method converges to a wrong alignment for step size 3 but converges to the correct global optimum for step sizes 1, 2.5, and 5.

For clouds of structured environments, examples of (Ma et al., 2016) show that there might be problems with iterative non-feature-based methods. Such problems occur if clouds overlap only partially or initial coarse alignment is bad. Also, running times strongly depend on cloud size and parameters. Another disadvantage is that we need additional information like digital terrain models to enrich sampled clouds.

In our scenarios, walls are dominant structures and reference surfaces are very simple. Therefore, we concentrate on pairing specific geometric primitives as do most feature-based registration methods (cf. (Chuang and Jaw, 2015)). One could directly detect plane segments that represent walls and align them with CityGML wall polygons. However, both detected segments and CityGML polygons are only approximations of real walls. Also CityGML polygons are simplified due to their level of detail. But most CityGML models use high quality building footprints from cadastral data. Therefore, we work with walls' footprints. Based on RANSAC estimates of wall planes, we rotate the point cloud so that walls become straightened up and ground is oriented parallel to the x-y-plane. By orthogonally projecting points to the x-y-plane, walls become visible as dense

[1]Running times are measured on a single core of a 2.4GHz i5 processor (2013) with 4GB RAM.

contours. This basically simplifies the 3D alignment problem to a 2D task of aligning these contours with CityGML building footprints. The RANSAC-based rotation step is similar to the use of angle-features in the general purpose point cloud alignment algorithm of (Ma et al., 2016). They compute a rotation matrix based on an angle histogram. Rotation leads to translation of the histogram that, for example, can be detected using the Fourier transform shift property[2]. However, angle-features become disturbed if different scaling factors are used for different coordinate directions during coarse registration. This also is a problem if one tries to detect rotation between 2D footprint images based on an angle histogram of Hough lines. In our scenario, Hough space registration (cf. (Zhao, 2006)) additionally suffers from different lengths of line segments.

Established alignment procedures for 2D images mostly avoid to solve an NP-complete combinatorial optimization problem. But footprint images of point clouds and city models are quite different so that combinatorial optimization promises to deliver results that are more robust against distortions. We investigated combinatorial alignment procedures based on corners of footprints, based on line segments of footprints and based on straight lines covering line segments. For UAV point clouds covering larger urban areas, corner-based alignment works well if there are sufficient many building corners (of different height) in a scene. In our experiments, we detected building corners using Harris corner detector and aligned them with vertices of the city model using a Mixed Integer Linear Program (MIP) similar to the one in Section 4, see Figure 2. Unfortunately, the videos that we use

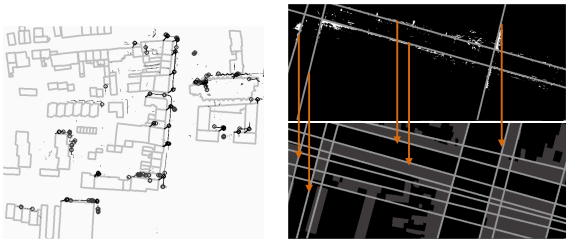

Figure 2: Left: Corner-based matching of building walls detected from a UAV point cloud (see Section 6) with CityGML model's walls. Right: Lines that are detected in the point cloud are matched with CityGML model lines.

for facade detection mostly are taken from street level. They cover more detail, and corner detection does not only find significant building corners but also corners of smaller structures. Also, in a straight street with few intersections, the number of significant corners is small. Therefore, our implementation of corner based matching does not work well with this type of point clouds. Instead, straight lines of building footprints are significant. Line-based matching approaches have been used successfully to align images with digital surface models and airborne laserscanning data, see for example (Avbelj et al., 2013; Cui et al., 2017). They have been also used to align two point clouds, see for example (Li et al., 2012). One can either match unbounded straight lines or the short bounded line segments that are part of footprints. There might be several line segments approximately lying on the same straight unbounded line. Thus joining line segments to unbounded lines reduces the number of possible matching combinations. Whereas matching of unbounded straight lines works for simple street scenarios (cf. Figure 2), it might fail for complex UAV point clouds that lead to a variety of similar lines. Therefore, we propose to match line segments (cf. Figure 4). Whereas (Avbelj et al., 2013) use a statistical, accumulator based approach to match lines, (Li et al., 2012) present an iteration scheme and (Cui et al., 2017) apply non-linear least-squares optimization, we use a simpler Mixed Integer Linear Program to find matching line segment pairs in combination with a Linear Program (LP) to compute a 2D transformation matrix. However, this requires pre- and post-processing steps that utilize the vertical nature of walls.

3 Pre-Processing

First, we both translate photogrammetric cloud and city model into a local coordinate system with origin at $-\vec{t} \in \mathbb{R}^3$ to avoid large numerical errors: The corresponding translation is given by

$$T = \begin{pmatrix} E & \vec{t} \\ (0,0,0) & 1 \end{pmatrix} \in \mathbb{R}^{4\times 4}$$

with unity matrix $E \in \mathbb{R}^{3\times 3}$. Then we rotate the photogrammetric cloud to correctly align with the z-axis and generate a 2D building footprint image as follows:

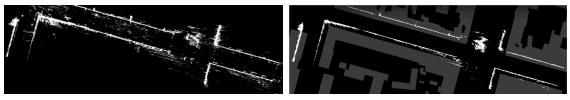

Figure 3: Density of point cloud and binary image of likely wall footprints superposed with CityGML building footprints

[2]Sarvaiya, J., Patnaik, S., and Kothari K.: Feature Based Image Registration Using Hough Transform. http://psrcentre.org/images/extraimages/48.%2050.pdf (Accessed 07 July 2017)

- To generate a preliminary binary image of likely wall footprints, a resolution of 9 pixels per square meter is sufficient for our data, see Figure 3. We compute minimum and maximum z-coordinates (height values) of all points with x- and y-coordinates within the pixel's area. If these values at least differ 3.5m in height (one building level) and if there exist at least eight points with z-coordinates pairwise belonging to disjoint intervals of width 0.5m then we classify the pixel as being part of a wall footprint. One could also generate a density image by counting the points above the pixel's area. Then thresholding could give a wall map. Unfortunately, we work with point clouds of very different local densities so that it is difficult to find or generate a suitable threshold value.
- Walls might not be exactly vertical. Before we reduce the cloud to wall and ground points, we have to rotate it with a matrix D to make walls upright. To this end, we divide the ground into $10\,\mathrm{m} \times 10\,\mathrm{m}$ sections. For each section we iteratively apply a RANSAC algorithm to the section's subset of the cloud that also corresponds roughly with previously computed pixels of wall footprints. With RANSAC we estimate nearly vertical planes for each section. We collect the normal vectors of the third of planes with largest number of inliers. Let N be the set of these normals. We estimate the common upward-direction of all walls with a RANSAC algorithm as well: We iteratively select a plane through the origin and through two non-collinear points in N. Out of all selected planes we choose one with the largest number of inliers p, $p \in N$. Then we apply a rotation D,

$$D = \begin{pmatrix} \cos(\beta) & -\sin(\beta)\sin(\alpha) & -\sin(\beta)\cos(\alpha) & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\beta) & \cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

  to the point cloud that aligns this plane's normal vector $(n.x, n.y, n.z)$, $n.z > 0$, with the z-axis. Let $h = \sin(\alpha) \cdot n.y + \cos(\alpha) \cdot n.z$. Angles are

$$\begin{aligned} \alpha &= \mathrm{sign}(n.y)\arccos(n.z/\sqrt{n.y^2 + n.z^2}), \\ \beta &= \mathrm{sign}(n.x)\arccos(h/\sqrt{n.x^2 + h^2}). \end{aligned}$$

  The approach requires the existence of walls with different orientations.
- Now we compute a sharper version of the binary wall footprint image. Since walls should exactly point upwards after applying D, we can reduce noise by filtering for even larger height differences (5m instead of 3.5m) to detect walls.

Our goal is to match the footprint image with a similar image that we obtain from the city model. To this end, we draw a single picture of filled footprints of all CityGML buildings, only considering the area of the photogrammetric point cloud. Then we detect edges with the Canny operator. These edges correspond with facades but not with walls between houses.

Both on the footprint image and on the edge picture, we apply a probabilistic Hough transform to detect line segments, see Figure 4. In the following section we use the sets P and Q that contain line segments of the footprint image and the model's edge picture, respectively.

4 Linear Programs

Linear and Mixed Integer Linear Programming have been used for registration purposes, see for example (Sakakubara et al., 2007; Wang et al., 2017). However, most alignment procedures use non-linear optimization. Linear optimization isn't even listed amongst the optimization methods for point cloud alignment in the overview article (Tam et al., 2013). But if one seeks for a linear or affine transformation and if one is allowed to measure errors in l_1-norm (sum of absolute values) instead of the widely used l_2-norm (least squares) then Linear Programming is a very powerful tool. MIP and LP also have been considered for the generation of CityGML models and 3D modeling. For example, (Boulch et al., 2014) use a MIP to reconstruct surfaces from point clouds.

The difficulty of our registration task is that we have to select from a candidate set of line pairs before we can compute a linear transformation to align line segments of P with line segments of Q. Let $P = \{(p_{1,1}, p_{1,2}), \ldots, (p_{m,1}, p_{m,2})\}$ and $Q = \{(q_{1,1}, q_{1,2}), \ldots, (q_{n,1}, q_{n,2})\}$. Each line segment is defined by its two endpoints that are given in homogeneous coordinates, for example $p_{i,k} = (p_{i,k}.x, p_{i,k}.y, 1)^\top$.

Now we have to determine a linear transform L that aligns the largest possible subset of P with a corresponding subset of Q by using translation, scaling and rotation as feasible operations.

A common method to find large corresponding sets of line features is to use RANSAC in Hough space. For example, (Colleu et al., 2008) use this approach to match video frames with city model data. In contrast to this we match bounded line segments with a MIP that automatically also computes an initial version of the transformation matrix.

First, we have to find matching candidate pairs

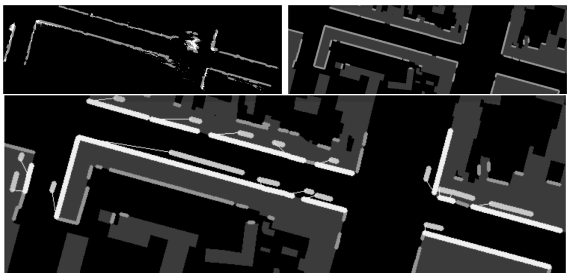

Figure 4: First two pictures: Detected line segments of point cloud and CityGML model; third picture: Matching line segments are connected with thin bright lines.

between P and Q. If a binary variable $x_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq n$, equals one, then the pair $((p_{i,1}, p_{i,2}), (q_{j,1}, q_{j,2}))$ is selected for matching. It is not selected if $x_{i,j} = 0$. We can a-priori exclude pairs by setting their binary variables fixed to zero. We do this if their lines have a distance in Hough space above a threshold value. Also, we compute a bounding box for each line segment and extend it by the expected error of coarse alignment. If bounding boxes of two line segments do not intersect then we also exclude the pair. This is the sole step in our alignment method that might cause it to only find a local but no global optimum. In contrast to ICP and other non-feature-based algorithms, what follows describes global optimization.

We have to maximize an objective function like

$$\sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j} \tag{1}$$

subject to the restriction that there is a linear mapping

$$L = \begin{pmatrix} s_1 \cos(\alpha) & -s_1 \sin(\alpha) & d_1 \\ s_2 \sin(\alpha) & s_2 \cos(\alpha) & d_2 \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{3\times 3}$$

that approximately maps line of segment $(p_{i,1}, p_{i,2})$ onto line of $(q_{j,1}, q_{j,2})$ if $x_{i,j} = 1$. Manual coarse registration should be done so that scaling s_1 in x- and scaling s_2 in y- direction is equal: $s := s_1 = s_2$. Then there is less uncertainty and more stability.

There need to exist scalars $r_{1,i,j}$ and $r_{2,i,j}$ such that for $k \in \{1,2\}$ and

$$d^+_{k,i,j} - d^-_{k,i,j} := q_{j,1} + r_{k,i,j}(q_{j,2} - q_{j,1}) - Lp_{i,k} \tag{2}$$

there holds true $0 \leq d^{\pm}_{k,i,j}.x \leq \varepsilon$, $0 \leq d^{\pm}_{k,i,j}.y \leq \varepsilon$. Thus, the coordinate-wise distance between transformed points $Lp_{i,1}$, $Lp_{i,2}$ and the straight line through $q_{j,1}$ and $q_{j,2}$ has to be bounded by a fixed threshold value ε. We work with $\varepsilon := 0.5$ for selection of candidate pairs (distances will be further minimized in a second optimization step). The source line segment has to be approximately mapped onto a straight line going through the target line segment. There is no need to actually hit the target segment. This allows for sliding like in ICP point-to-plane mode. Nevertheless, we match segments and not straight lines because we a-priori exclude pairs (i, j) of distant segments by setting $x_{i,j} = 0$.

To obtain a linear problem, we do not compute sine and cosine functions but seek for a matrix

$$L = \begin{pmatrix} l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,1} & l_{2,2} & l_{2,3} \\ 0 & 0 & 1 \end{pmatrix}$$

under restrictions

$$\left.\begin{array}{l} l_{1,1} = l_{2,2}, \\ l_{1,2} = -l_{2,1} \end{array}\right\} \text{ if } s_1 = s_2, \tag{3}$$

$$\left.\begin{array}{l} -\mu \leq l_{1,1} - l_{2,2} \leq \mu, \\ -\mu \leq l_{1,2} + l_{2,1} \leq \mu \end{array}\right\} \text{ if } s_1 \neq s_2, \tag{4}$$

$$1 - \delta \leq l_{1,1} \leq 1 + \delta, \quad -\delta \leq l_{1,2} \leq \delta, \tag{5}$$

where $\delta = 0.3$ and $\mu = 0.1$ are small threshold values. These restrictions avoid that L describes mirroring.

Instead of using simple objective function (1) we consider the lengths of point cloud line segments as weights. Let w_i be the length of the i-th point cloud line segment. We extend the objective function to $\sum_{i=1}^{m} \sum_{j=1}^{n} w_i x_{i,j}$. Maximization favors selection of long line segments.

We write the optimization problem as an Integer Linear Program with the help of a large number M. M is used to restrict the distance condition to selected pairs of line segments while keeping the problem linear. Let distances $d^+_{k,i,j}, d^-_{k,i,j} \in (\mathbb{R}^{\geq 0})^3$ and matrix coefficients $l_{r,c} \in \mathbb{R}$, $1 \leq r \leq 2$, $1 \leq c \leq 3$:

$$\text{Max.} \sum_{i=1}^{m} \sum_{j=1}^{n} w_i x_{i,j} \text{ s.t.}$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1 \text{ for } 1 \leq j \leq n, \quad \sum_{j=1}^{n} x_{i,j} \leq 1 \text{ for } 1 \leq i \leq m,$$

conditions (2) and ((3) or (4)) and (5), and

$$\max\{d^+_{k,i,j}.x, d^+_{k,i,j}.y, d^-_{k,i,j}.x, d^-_{k,i,j}.y\} + Mx_{i,j} \leq \varepsilon + M.$$

A MIP in general is NP complete. It very much depends on the number of candidate correspondences and on the size of ε how long a solver takes to find a solution. We use the GNU Linear Programming Kit library GLPK (Makhorin, 2009) to solve LPs and MIPs. In most scenarios of this paper, solutions of the MIPs are found in less than a second. Nevertheless, running times can be decreased. It turns out that the exclusion of distant line segments from the matching task allows for an LP relaxation of the MIP.

In the relaxed problem, binary variables $x_{i,j}$ are replaced by real-valued variables $x_{i,j} \in [0,1]$. It turns out that values of $x_{i,j}$ become indeed very close either to zero or to one. We then round them. The tradeoff of relaxation is that one has to deal with a few wrong matchings during the following second optimization step that improves the mapping L:

Based on an optimal solution of the MIP, we define the set R of pairs (i,j) for which $x_{i,j} = 1$ so that for all $(i,j) \in R$ distances fulfill $0 \leq d^{\pm}_{k,i,j}.x, d^{\pm}_{k,i,j}.y \leq \varepsilon$. To further minimize the errors, we apply another LP. By using the same variable names as for the MIP, this LP has to minimize

$$\sum_{(i,j)\in R} \sum_{k=1}^{2} w_i (d^{+}_{k,i,j}.x + d^{+}_{k,i,j}.y + d^{-}_{k,i,j}.x + d^{-}_{k,i,j}.y)$$

subject to (2) for $(i,j) \in R$, and conditions (3) or (4), and (5).

Weights w_i now enforce large line segments to be matched with higher precision than short ones.

Based on L, we can align the point cloud in the x-y-plane using matrix

$$A := \begin{pmatrix} l_{1,1} & l_{1,2} & 0 & l_{1,3} \\ l_{2,1} & l_{2,2} & 0 & l_{2,3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

5 Post-Processing

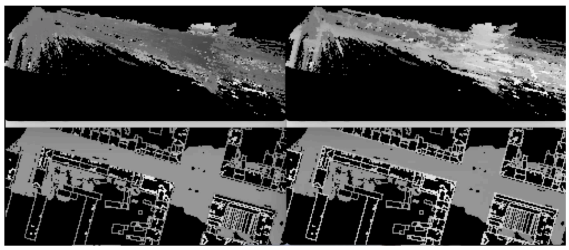

Figure 5: Height maps showing minimum and maximum values of adjusted photogrammetric point cloud and sampled model cloud

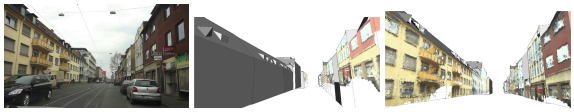

Figure 6: A video frame used for point cloud generation, projection of point cloud onto city model prior to alignment, and projection of aligned point cloud onto city model

It remains to estimate an additional z-scaling factor and a z-translation. To this end, we look at terrain intersection points of CityGML models. For each such point, we have a true z-value z_g of the ground and also determine the height z_r of the roof above this point. We also determine a corresponding lowest and highest point of the pre-processed point cloud that has been transformed with A, see Figure 5. Let z_l and z_h be the z-coordinates of these two points, respectively. Then we get a local z-scaling factor $\frac{z_r - z_g}{z_h - z_l}$. If the same scaling factor was applied to all directions during manual coarse registration, z-scaling factor should be $s = \sqrt{l_{1,1} \cdot l_{2,2} - l_{1,2} \cdot l_{2,1}}$. We allow a small deviation by considering all local scaling factors within the interval $[s-0.2, s+0.2]$. To avoid outliers, we compute the median value z_s of these feasible local scaling factors. Using z_s we determine the z-translation value z_t as the median of all local translation values $z_g - z_l \cdot z_s$. Then we finally align the cloud with the city model by multiplying its points with

$$T^{-1}ZADT, \quad Z := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & z_s & 0 \\ 0 & 0 & 0 & z_t \end{pmatrix}.$$

Figure 7: Perspective projection of video frame areas onto CityGML walls

Figure 8: A city model is drawn onto a video frame using the frame's SfM camera transformation and estimated camera projection. The upper picture is based on a coarse manual alignment between SfM point cloud and UTM coordinate system. The second picture shows optimized alignment.

After aligning the point cloud with the city model, we can match model walls with areas in video frames or orthogonally project points to walls, see Figures 6 and 7. Fortunately, the SfM tool provides camera parameters and, in a bundle output file, for each frame $k \in \{1,\ldots,n\}$ a translation vector $\vec{t}_k \in \mathbb{R}^3$ and a rotation matrix $R_k \in \mathbb{R}^{3\times 3}$ that together define a transformation C_k of the original, non-aligned photogrammetric cloud into the frame's camera coordinate system:

$$C_k := \begin{pmatrix} R_k & \vec{t}_k \\ (0,0,0) & 1 \end{pmatrix}.$$

After applying this mapping, the camera's position is the origin and the camera looks into the direction of

the negative z-axis. We multiply the photogrammetric point cloud with matrix $T^{-1}ZADT$ to align with the city model. Thus, the transformation of the city model into the camera's coordinate system is given via matrix $C_k(T^{-1}ZADT)^{-1} = C_kT^{-1}D^{-1}A^{-1}Z^{-1}T$. We can now render the model using camera parameters so that it exactly fits with the camera frame (see Figure 8 for the UAV city center scenario that is described in Section 6). Vice versa, CityGML walls can be textured either based on the point cloud (Figure 6), or based on the corresponding textured mesh or based on single video frames (Figure 7).

6 Results

We apply the algorithm to three different point clouds. So far, we have used a dense point cloud (now denoted as St. Anton street) as example. The point cloud of Knight street (see Figure 9) also is taken through the front window of a car, but it is much thinner than the one of St. Anton street. The third cloud (see Figure 10) originates from a UAV video of a city center. This scenario also allows for alignment by matching corners, see Figure 2. To visualize transformations, we worsened coarse manual registration by shifting the clouds by 4 meters in x- and y-directions, respectively. Table 1 and Figures 9 and 10 summarize results of computation.

Precision of the registration corresponds with resolution $1/3$ meter of the pictures used to detect wall lines. It can be slightly improved by increasing pictures' resolution. However, if the resolution is too high, it becomes difficult to detect walls.

To investigate the limits of our algorithm, we align sampled model point clouds with transformed versions of themselves in a local coordinate system. These small clouds are free of noise and lead to a high number of correspondences. This increases running time of our MIP optimization step whereas such clouds are simple to match for ICP in point-to-plane mode. For example, our algorithm takes 82 seconds to (approximately) undo the transformation B^{-1} (rotations by 0.01 degrees around all axes, scaling factor 0.99 and translation with $(-4,-4,4)$) applied to the model cloud of Figure 1. It computes a matrix $\tilde{B}$ that is a good approximation of ground truth matrix B. In comparison, ICP in point-to-plane mode[3] converges to a corresponding matrix $\hat{B}$ in 89 seconds. In this example, our algorithm finds 136 candidate pairs of corresponding line segments. The single MIP step runs

[3]Point Cloud Library's class IterativeClosestPointWithNormals is used with parameters TransformationEpsilon $= 10^{-8}$ and EuclidianFitnessEpsilon $= 0.1$.

Table 1: Three investigated scenarios:

	St. Anton street	Knight street	UAV city scenario
number of points	8.593.746	1.995.498	5.017.262
line segments of point cloud	114	71	68
line segments of city model	61	73	338
candidate pairs of line segments	71	95	56
pairs selected by relaxation of MIP	24	24	24
pairs selected by MIP	17	20	21
model vertices used for z-operations	32	48	33
running time pre-processing	14s	$< 4s$	$< 1s$
running time MIP and LP	$< 1s$	$< 1s$	$< 1s$
running time post-processing	15s	6s	2s

67 seconds. If we do not compute the exact solution of the MIP but use a solution of the relaxed MIP then overall running time decreases to 14 seconds and the optimization steps finish in less than a second with outcome matrix $\tilde{\tilde{B}}$. The overall running time can be further decreased by limiting the number of RANSAC plane estimates in the pre-processing step.

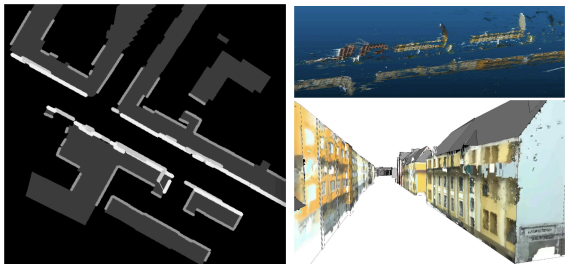

Figure 9: Knight street scenario: matching between line segments, thin point cloud, points projected to model walls

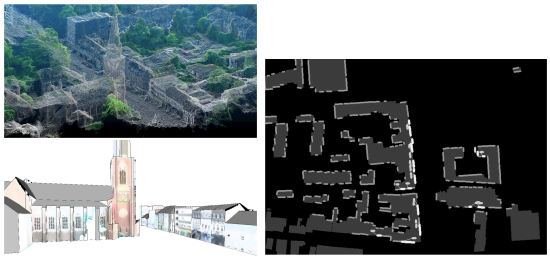

Figure 10: Left: UAV point cloud and points projected to model walls. Right: matching between line segments

The l_2-vector norm is defined via $\|(a_1,\ldots,a_n)\|_2 := \sqrt{a_1^2+\cdots+a_n^2}$, and the l_2-matrix (spectral) norm

$$\|B-\tilde{B}\|_2 := \sup\left\{\frac{\|(B-\tilde{B})\cdot\vec{a}\|_2}{\|\vec{a}\|_2} : \vec{0}\neq\vec{a}\in\mathbb{R}^4\right\}$$

is a measure for the quality of the computed alignment mapping $\tilde{B}$. Let $\vec{v} \in \mathbb{R}^3$ be a point of the cloud and $\vec{e} = (e.x, e.y, e.z) \in \mathbb{R}^3$ the difference between the correctly and approximately aligned versions of $\vec{v}$. Then

$$\begin{aligned} \|(e.x, e.y, e.z, 0)\|_2 &= \left\| (B - \tilde{B}) \cdot (v.x, v.y, v.z, 1)^\top \right\|_2 \\ &\leq \|B - \tilde{B}\|_2 \sqrt{\|\vec{v}\|_2^2 + 1}. \end{aligned}$$

Our non-relaxed algorithm aligns best with $\|B - \tilde{B}\|_2 \approx 0.1407$ in the current scenario, followed by the relaxed version ($\|B - \tilde{\tilde{B}}\|_2 \approx 0.2069$) and by ICP ($\|B - \hat{B}\|_2 \approx 0.288$).

7 Conclusions

One can utilize the vertical orientation of walls to reduce the problem of aligning photogrammetric point clouds with 3D city models to two space dimensions if at least two wall segments with linear independent directions are detected. This might be given if the scene covers an intersection of streets. Then, instead of ICP, feature-based alignment using Linear Programming is a suitable means. Useful results are obtained by matching line segments of wall footprints. While the algorithm is designed to align with city models, it can also be used to align two point clouds in which walls are dominant. Although not so fast, the point-to-plane version of ICP also aligns well with sampled CityGML models.

REFERENCES

Avbelj, J., Iwaszczuk, D., Müller, R., Reinartz, P., and Stilla, U. (2013). Line-based registration of DSM and hyperspectral images. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-1/W1:13–18.

Boulch, A., de La Gorce, M., and Marlet, R. (2014). Piecewise-planar 3D reconstruction with edge and corner regularization. *Computer Graphics Formum*, 33(5):55–64.

Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155.

Chuang, T.-Y. and Jaw, J.-J. (2015). Automated 3D feature matching. *The Photogrammetric Record*, 30(149):8–29.

Colleu, T., Sourimant, G., and Morin, L. (2008). Automatic initialization for the registration of GIS and video data. In *Proc. 2008 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 49–52, Washington, DC. IEEE.

Cui, T., Ji, S., Shan, J., Gong, J., and Liu, K. (2017). Line-based registration of panoramic images and lidar point clouds for mobile mapping. *Sensors*, 17(1).

Gröger, G., Kolbe, T. H., Nagel, C., and Häfele, K. H. (2012). *OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0*. Open Geospatial Consortium.

Holz, D., Ichim, A. E., Tombari, F., Rusu, R. B., and Behnke, S. (2015). Registration with the point cloud library: A modular framework for aligning in 3-D. *IEEE Robotics Automation Magazine*, 22(4):110–124.

Li, W., Li, X., Bian, Y., and Zhao, H. (2012). Multiple view point cloud registration based on 3D lines. In *Proc. International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*, pages 1–5.

Ma, Y., Guo, Y., Zhao, J., Lu, M., Zhang, J., and Wan, J. (2016). Fast and accurate registration of structured point clouds with small overlaps. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 643–651.

Magnusson, M., Nüchter, A., Lörken, C., Lilienthal, A. J., and Hertzberg, J. (2009). Evaluation of 3D registration reliability and speed — a comparison of ICP and NDT. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 3907–3912, Washington, DC. IEEE.

Maiseli, B., Gu, Y., and Gao, H. (2017). Recent developments and trends in point set registration methods. *Journal of Visual Communication and Image Representation*, 46:95–106.

Makhorin, A. (2009). *The GNU Linear Programming Kit (GLPK)*. Free Software Foundation, Boston, MA.

Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the ICP algorithm. In *Proc. Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152.

Sakakubara, S., Kounoike, Y., Shinano, Y., and Shimizu, I. (2007). Automatic range image registration using mixed integer linear programming. In Yagi, Y., Kang, S. B., Kweon, I. S., and Zha, H., editors, *Proc. 8th Asian Conference on Computer Vision, Tokyo 2007, Part II*, pages 424–434, Berlin. Springer.

Tam, G. K., Cheng, Z.-Q., Lai, Y.-K., Langbein, F. C., Liu, Y., Marshall, D., Martin, R. R., Sun, X.-F., and Rosin, P. L. (2013). Registration of 3D point clouds and meshes: A survey from rigid to non-rigid. *IEEE Trans. Visualization and Computer Graphics*, 19(7):1–20.

Wang, Y., Moreno-Centeno, E., and Ding, Y. (2017). Matching misaligned two-resolution metrology data. *IEEE Transactions on Automation Science and Engineering*, 14(1):222–237.

Zhao, S. (2006). Hough-domain image registration by metaheuristics. In *Proc. 9th International Conference on Control, Automation, Robotics and Vision 2006*, pages 1–5.