ITERATIVE CLOSEST POINT ALGORITHM FOR ACCURATE REGISTRATION OF COARSELY REGISTERED POINT CLOUDS WITH CITYGML MODELS

S. Goebbels[1,*], R. Pohle-Fröhlich[1], P. Pricken[1]

[1] iPattern Institute, Niederrhein University of Applied Sciences, Reinarzstr. 49, 47805 Krefeld, Germany - (Steffen.Goebbels, Regina.Pohle)@hsnr.de, Philipp.Pricken@stud.hn.de

KEY WORDS: CityGML, Registration of Point Clouds, Iterative Closest Point algorithm, Data Fusion

ABSTRACT:

The Iterative Closest Point algorithm (ICP) is a standard tool for registration of a source to a target point cloud. In this paper, ICP in point-to-plane mode is adopted to city models that are defined in CityGML. With this new point-to-model version of the algorithm, a coarsely registered photogrammetric point cloud can be matched with buildings' polygons to provide, e.g., a basis for automated 3D facade modeling. In each iteration step, source points are projected to these polygons to find correspondences. Then an optimization problem is solved to find an affine transformation that maps source points to their correspondences as close as possible. Whereas standard ICP variants do not perform scaling, our algorithm is capable of isotropic scaling. This is necessary because photogrammetric point clouds obtained by the structure from motion algorithm typically are scaled randomly. Two test scenarios indicate that the presented algorithm is faster than ICP in point-to-plane mode on sampled city models.

1. INTRODUCTION

3D city models are used for several simulation and planning purposes. The XML based description language CityGML, see (Gröger et al., 2012), has become standard for exchanging semantic city models. In the past, most contributions focused on finding correct roof topologies since most models were generated from airborne laser scanning data. For example, such LIDAR point clouds are freely available from the state cadastral office of our country (open data initiative). Resulting 3D models are given in a level of detail (LoD) 2, i.e., they consist of roof and wall polygons but do not show further details like windows or doors. To obtain facade information, other data sources than airborne laser scanning point clouds have to be used. Whereas access to oblique areal photos or mobile mapping data might be expensive, video sequences of buildings are available at nearly no cost. Using the "structure from motion" algorithm on video frames, one obtains colored 3D point clouds. However, often positioning data is not available or not sufficiently precise. To do a precise registration of only coarsely positioned photogrammetric point clouds, one can match them, for example, with available airborne laser scanning point clouds or with already available 3D models. We discuss a new algorithm to directly fit point clouds with CityGML models (cf. Figure 1).

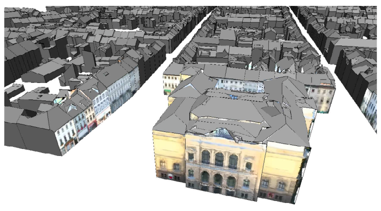


Figure 1. Photogrammetric point cloud aligned with a CityGML model. Facade textures consist of orthogonally projected points.

There are several feature- and non-feature based approaches to register a source point cloud $P \subset \mathbb{R}^3$ with target cloud $Q \subset \mathbb{R}^3$. For example, the algorithm in (Makadia et al., 2006) determines rotation angles by using Fourier methods on surface orientation histograms. One can also determine curvature values, edges, corner points, planes etc. as features for matching (cf. (Cheng et al., 2018) and literature cited there).

The Iterative Closest Point algorithm (ICP) (Chen , Medioni, 1992, Besl , McKay, 1992) is the standard non-feature-based algorithm to iteratively register point cloud P with Q. Basically, it is a greedy algorithm. In each iteration, it determines the nearest neighbor in Q of each source point in P with regard to its Euclidian distance (ICP in point-to-point mode). Then it solves a least-squares optimization problem to obtain a transform that maps the points to its nearest neighbors in a best possible sense. This transform is applied to all source points. Then the next iteration starts. Over the years, many variants of ICP evolved, see (Rusinkiewicz , Levoy, 2001) and the literature cited there. In (Goebbels , Pohle-Fröhlich, 2018), a feature based aligning procedure is compared with several ICP variants that are implemented in Point Cloud Library (version 1.8.0). To this end, target point clouds were generated by sampling CityGML data. Whereas ICP in classical point-to-point mode did not perform well, ICP in point-to-plane mode (see (Holz et al., 2015) and tutorials[1]) converged much faster. In point-to-plane mode, distances are not measured between source and target points but between source points and planes defined by estimated local normal vectors of target points (see (Chen , Medioni, 1992)). Looking at intermediate steps, the source point cloud appears to slide along planes until it matches with the target point cloud. A similar approach is the Normal Distribution Transform (Magnusson et al., 2009). It optimizes the probability of a point to be in the right place by matching it with superimposed density functions of the normal distribution instead of planes. Such matching of points with surfaces motivates us to modify ICP in point-to-plane mode to directly consider polygonal plane segments of target CityGML models. By using information about planes, there is no need to estimate normal vectors based on

[*] Corresponding author

[1] http://pointclouds.org/ documentation/tutorials

nearest neighbor points. Also, we consider an isotropic scaling factor. This should result both in faster convergence and higher precision. Classical ICP variants are not capable of scaling, but there are extensions for isotropic (Zinßer et al., 2005) and even for anisotropic scaling, see (Du et al., 2007). Scaling might allow a point cloud to contract to a single point. This cannot happen if one uses ICP to match two point clouds and if one also uses an injective correspondence between a (large) subset of the source and the target point cloud. But it definitely can happen if multiple correspondences to a target point are allowed or if one projects points to planes. Thus, we have to avoid this phenomenon. Whereas the algorithm in (Zinßer et al., 2005) allows arbitrary scaling factors, small factors are excluded in (Du et al., 2007) by an additional restriction. We also limit a scaling factor σ to be in a small interval around 1 by implementing separate alternating optimization procedures for rotation and translation on the one side and for scaling on the other side.

The paper is organized as follows. The next section deals with preparing data. Then the algorithm is described in the main section. After that, a section deals with experimental results. The two latter sections contains our main contributions: a new variant of ICP based on orthogonal projections to the city model and performance optimization based on the optimization method, replacement of CityGML polygons by bounding rectangles and reduction of the number of multiplications.

2. PRE-PROCESSING

We expect point clouds to be coarsely registered with the UTM coordinate system that is also used within CityGML models. A significant fraction of points should be not farther away from corresponding positions of the city model than the used threshold distance (5 m in our tests).

Photogrammetric point clouds of city scenes consist of building points but also of background points mostly representing ground and vegetation. We only consider points on the perimeter of buildings. The set of these points changes from one to the next iteration. However, this mostly excludes ground. Buildings and facades might be occluded by vegetation. Therefore, we initially remove all points with a dominant green color. Typically, computation of corresponding points is the most time consuming step of ICP, cf. (Rusinkiewicz , Levoy, 2001). Several strategies are available to further reduce the number of points for speedup. When matching with CityGML models, i.e., with planar polygonal areas, only such points are relevant that are approximately placed on a plane which also matches a larger number of other points. Thus, one can use RANSAC to determine the largest planes. If a point is not close to one of these planes, it can be removed. However, such a reduction step is time consuming, too (cf. (Goebbels , Pohle-Fröhlich, 2018)). Compared with the number of points (millions), we project onto only a few thousand polygons. Thus, a reduction strategy is not required.

3. THE ALGORITHM

In each outer iteration step of the algorithm, the task is to find and apply a linear transformation matrix $T = T_{\vec{a}} \in \mathbb{R}^{4\times 4}$ that maps source points $\vec{s}$, given in (normalized) homogenous coordinates, to their corresponding points $\vec{d}$, also noted in homogenous coordinates, best possible or at least better than without the transform. To find such a transform, an optimization problem has to be solved. This is done using inner iterations.

3.1 Corresponding Points

In each outer iteration step, we have to identify corresponding points first. In a preprocessing step, we identify bounding rectangles for each wall and each roof polygon that have to be planar in CityGML. Each rectangle lies on the same plane as its polygon and encloses it. Typically, walls have left and right boundaries that are orthogonal to the ground. Thus, small enclosing bounding rectangles can be determined easily. When dealing with roof polygons, we compute the principal component direction of their projection to the x-y-plane. We use this direction to orientate the rectangle. To this end, we equidistantly sample 2D points of the polygonal curve's projection. Ten sample values per meter appear to be consistent with precision of city models. Then we compute the center of gravity of 2D points and an eigenvector belonging to the largest eigenvalue of these point's covariance matrix. The eigenvector points towards the principle direction. In the x-y-plane, we define a 2D bounding box around the center of gravity by considering the largest distances of points to the lines through the center of gravity defined by the eigenvector and by a vector orthogonal to the eigenvector. To obtain the 3D bounding rectangle, we add z-coordinates according to the plane's equation.

For each bounding rectangle, we determine all source points $\vec{s}$ within a threshold distance τ ($\tau = 5\,\mathrm{m}$ for results of Section 4). This can be done efficiently by organizing the source point cloud in a quadtree or octree. Whereas x- and y coordinates typically cover a wide range of values, z-coordinates (height values) only lie within a small interval. Thus, organizing only x- and y-values is sufficient for dealing with most city models. We project each point $\vec{s}$ orthogonally to the plane defined by the bounding rectangle, getting a candidate nearest point $\vec{c}$. If the polygon's bounding rectangle has vertices $\vec{p}_1$, $\vec{p}_2$, $\vec{p}_3$, and $\vec{p}_4 = \vec{p}_1 + (\vec{p}_2 - \vec{p}_1) + (\vec{p}_3 - \vec{p}_1)$ with linear independent edge vectors $(\vec{p}_2 - \vec{p}_1)$ and $(\vec{p}_3 - \vec{p}_1)$ then by setting

$$\mu_1 := \frac{< \vec{p}_2 - \vec{p}_1, \vec{s} - \vec{p}_1 >}{\|\vec{p}_2 - \vec{p}_1\|_2}, \quad \mu_2 := \frac{< \vec{p}_3 - \vec{p}_1, \vec{s} - \vec{p}_1 >}{\|\vec{p}_3 - \vec{p}_1\|_2}$$

the candidate point is

$$\vec{c} = \vec{p}_1 + \frac{\mu_1}{\|\vec{p}_2 - \vec{p}_1\|_2}(\vec{p}_2 - \vec{p}_1) + \frac{\mu_2}{\|\vec{p}_3 - \vec{p}_1\|_2}(\vec{p}_3 - \vec{p}_1). \quad (1)$$

Here $< \cdot, \cdot >$ denotes the standard inner product in $\mathbb{R}^4$, and $\|\cdot\|_2$ is the Euclidian length of a vector.

If $\mu_1 > \|\vec{p}_2 - \vec{p}_1\|_2$, $\mu_1 < 0$, $\mu_2 > \|\vec{p}_3 - \vec{p}_1\|_2$, or $\mu_2 < 0$ then $\vec{c}$ lies outside the bounding rectangle. In that case we do not consider $\vec{s}$ if $\mu_1 > \|\vec{p}_2 - \vec{p}_1\|_2 + \tau$, $\mu_1 < -\tau$, $\mu_2 > \|\vec{p}_3 - \vec{p}_1\|_2 + \tau$, or $\mu_2 < -\tau$. Otherwise, if $\vec{c}$ lies outside the bounding rectangle but is closer to its boundary, we choose point $\vec{c}$ on the boundary of the rectangle by replacing μ_1 or μ_2 in (1) by $\mu_1 = \|\vec{p}_2 - \vec{p}_1\|_2$, $\mu_1 = 0$, $\mu_2 = \|\vec{p}_3 - \vec{p}_1\|_2$, or $\mu_2 = 0$.

If $\vec{c}$ is the first point associated with $\vec{s}$ or if $\vec{c}$ is nearer to $\vec{s}$ in Euclidian distance than a previously associated point $\vec{d}$ then we update the candidate target point $\vec{d}$ for $\vec{s}$ with $\vec{c}$. If after iterating through all bounding rectangles no feasible candidate $\vec{d}$ exists for a point $\vec{s}$ then we exclude $\vec{s}$ from further computations during current iteration.

Most wall and many roof polygons are rectangles that exactly fit with their bounding rectangle. If roof polygons have a different shape then bounding boxes are oriented according to a

Figure 2. Distance transform: Grey values in the right picture represent the distance to the polygon shown in the left image. Within the polygon, the distance is zero (black).

principal component direction of their vertices. Thus, the rectangle's area not covered by the polygon is small. However, one can further improve the computation of corresponding points by taking the real shapes of polygons into account. As an option, our algorithm is able to consider these shapes instead of their bounding rectangles. This leads to some additional computational costs. One has to determine if a point is inside or outside a polygon, for example by using a scan line algorithm. For an outside point, one also has to find a nearest point on the polygon's edges. But, for example, the nearest point on a polygon's border might not be positioned on the edge between the two nearest vertices, see Figure 3. Thus, one has to con-

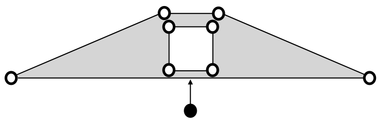


Figure 3. The black point has to be projected to the edge with vertices that are farthest away.

sider distances to all vertices and to all orthogonal projections to edge lines if they lie on the edges. We use an alternative raster image approach that would also work for arbitrary planar shapes: For each bounding rectangle the algorithm generates an image showing its polygonal area. CityGML polygons do have one outer boundary but are also allowed to have several inner boundaries that define holes. Then for each pixel outside the polygon, a shortest distance to the polygon is computed (distance transform) and stored in a matrix, see Figure 2. Such a distance map of a bounding rectangle is generated when a point is projected to the rectangle for the first time. With this lazy instantiation, we only compute distance maps that are really needed. If $\vec{c}$ corresponds with an inner pixel of the polygon, $\vec{c}$ can be used without modification. But if $\vec{c}$ lies on a pixel outside the polygon, then one can follow the negative gradient of the distance map, i.e., the direction of steepest descent, until the polygon is reached. Then $\vec{c}$ is replaced with this new position.

3.2 Rotation and Translation Parameters, Inner Iterations

Typically, coordinates of city models are given in very large UTM coordinates. Because of precision, it does not make sense to rotate or scale with respect to the origin in UTM coordinates. Thus, we compute a point $(\overline{x}, \overline{y}, \overline{z})$ near the center of the scene. Then we subtract this vector from all points.

After computation of the so translated matching pairs $(\vec{s}_k, \vec{d}_k)$, $1 \leq k \leq n$, we have to find a best possible transformation matrix T with $T\vec{s}_k \approx \vec{d}_k$, $1 \leq k \leq n$. This part of the algorithm is basically the same as for matching two given point clouds. However, we add an extra optimization step to allow limited scaling in the next subsection, see (5).

Matrix $T = T_{\vec{a},\sigma}$ depends on a scaling factor σ and parameters $\vec{a} := (\alpha, \beta, \gamma, \Delta x, \Delta y, \Delta z)$. Euler angles α, β, and γ describe rotation around x-, y-, and z-axis (in this order). Then translation by a vector $(\Delta x,\Delta y,\Delta z,0)^\top$ is performed. The outcome is scaled by $\sigma > 0$. Typically, scaling of photogrammetric point clouds (for example generated with structure from motion algorithm) is isotropic.

Then $T_{\vec{a},\sigma} :=$

$$\sigma \cdot \begin{pmatrix} \cos(\gamma)\cos(\beta) & t_{1,2} & t_{1,3} & \Delta x \\ \sin(\gamma)\cos(\beta) & t_{2,2} & t_{2,3} & \Delta y \\ -\sin(\beta) & \cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) & \Delta z \\ 0 & 0 & 0 & \frac{1}{\sigma} \end{pmatrix},$$

$$\begin{aligned} t_{1,2} &= -\sin(\gamma)\cos(\alpha) + \cos(\gamma)\sin(\beta)\sin(\alpha) \\ t_{1,3} &= \sin(\gamma)\sin(\alpha) + \cos(\gamma)\sin(\beta)\cos(\alpha) \\ t_{2,2} &= \cos(\gamma)\cos(\alpha) + \sin(\gamma)\sin(\beta)\sin(\alpha) \\ t_{2,3} &= -\cos(\gamma)\sin(\alpha) + \sin(\gamma)\sin(\beta)\cos(\alpha). \end{aligned}$$

For each pair $(\vec{s}, \vec{d})$ we discuss local residuum

$$\vec{r}(\vec{a}, \sigma) = (r_1, r_2, r_3, 0)^\top := T_{\vec{a},\sigma} \cdot \vec{s} - \vec{d} \in \mathbb{R}^4.$$

With respect to the variables $\vec{a}$ and a fixed scaling factor $\sigma = 1$, residuum $\vec{r}(\vec{a}, 1)$ has a Jacobian

$$J(\vec{a}) := \begin{pmatrix} \frac{\partial r_1}{\partial \alpha} & \frac{\partial r_1}{\partial \beta} & \frac{\partial r_1}{\partial \gamma} & \frac{\partial r_1}{\partial \Delta x} & \frac{\partial r_1}{\partial \Delta y} & \frac{\partial r_1}{\partial \Delta z} \\ \frac{\partial r_2}{\partial \alpha} & \frac{\partial r_2}{\partial \beta} & \frac{\partial r_2}{\partial \gamma} & \frac{\partial r_2}{\partial \Delta x} & \frac{\partial r_2}{\partial \Delta y} & \frac{\partial r_2}{\partial \Delta z} \\ \frac{\partial r_3}{\partial \alpha} & \frac{\partial r_3}{\partial \beta} & \frac{\partial r_3}{\partial \gamma} & \frac{\partial r_3}{\partial \Delta x} & \frac{\partial r_3}{\partial \Delta y} & \frac{\partial r_3}{\partial \Delta z} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Let $(T_{\vec{a}})_\alpha$ be the matrix that is derived from $T_{\vec{a},\sigma}$ for $\sigma = 1$ by partially differentiating all components with regard to α. Then the first column of $J(\vec{a})$ can be written as $(T_{\vec{a}})_\alpha \cdot \vec{s}$. In the same manner, all other columns can be expressed using component-wise derivatives of $T_{\vec{a}}$.

We combine all local residua to one vector in $\mathbb{R}^{3n}$:

$$\vec{R}(\vec{a}, \sigma) := (\vec{r}_{1,1}, \vec{r}_{1,2}, \vec{r}_{1,3}, \vec{r}_{2,1}, \ldots, \vec{r}_{n,3})^\top$$

where $\vec{r}_{k,j}$ is the j-th component, $j \in \{1, 2, 3\}$, of the local residuum of pair $(\vec{s}_k, \vec{d}_k)$.

Subject to a scaling factor $\sigma = 1$, we have to minimize an objective function (mean of the squared residua)

$$e(\vec{a}, 1) := \left\| \frac{1}{\sqrt{n}} \vec{R}(\vec{a}, 1) \right\|_2^2 = \frac{1}{n} \sum_{k=1}^{3n} (\vec{R}_k(\vec{a}, 1))^2 \qquad (2)$$

to find optimal rotation and translation parameters.

Amongst other approaches like singular value decomposition, a minimum of (2) can be computed with the Gauss Newton or Levenberg-Marquardt least squares optimization methods. The Levenberg-Marquardt algorithm introduces a damping factor that combines the Gauss Newton method with steepest gradient descent. Gauss-Newton and Levenberg-Marquardt optimization lead to inner iterations (3) and (4) that have to be performed for each outer iteration step of ICP. The inner iterations start with $\vec{a}_0 := (0, 0, 0, 0, 0, 0)$. Then one iteratively computes vectors $\vec{a}_l$ that hopefully converge to the position of a (local) minimum. In case of Gauss Newton optimization

$$\vec{a}_{l+1} := \vec{a}_l - (D(\vec{a}_l)^\top \cdot D(\vec{a}_l))^{-1} \cdot D(\vec{a}_l)^\top \frac{1}{\sqrt{n}} \vec{R}(\vec{a}_l, 1). \quad (3)$$

The Jacobian $D(\vec{a})$ of $\frac{1}{\sqrt{n}}\vec{R}(\vec{a},1)$ is composed from Jacobians of local residua: The $3k+j$-th row of $D(\vec{a})$ is given as the j-th row of $\frac{1}{\sqrt{n}}J(\vec{a})$ where the Jacobian $J(\vec{a})$ belongs to $\vec{r}_k$, the local residuum of pair $(\vec{s}_k, \vec{d}_k)$ for $\vec{a}$.

Using the Gauss Newton method, we cannot control the size of parameter changes. This is the advantage of Levenberg-Marquardt iterations. Let I be the identity matrix and $\lambda_l \geq 0$ parameters controlling the search radius r (step size) for local minima,

$$\vec{a}_{l+1} := \vec{a}_l - (D(\vec{a}_l)^\top \cdot D(\vec{a}_l) + \lambda_l I)^{-1} \cdot D(\vec{a}_l)^\top \frac{1}{\sqrt{n}} \vec{R}(\vec{a}_l, 1). \quad (4)$$

The factor λ_l is motivated by a Lagrange multiplier that results from a condition $\|\vec{a}_{l+1} - \vec{a}_l\|_2^2 = r^2$. The step size is updated from step to step starting with an initial value λ_0, see (Marquardt, 1963). A Gauss-Newton step is given if $\lambda_l = 0$. For large λ_l, a small step of gradient descent is performed. In case of Levenberg-Marquardt iterations, only downhill steps have to be considered: If the objective function does not decrease, parameter λ_l has to be increased (doubled in our implementation). This corresponds with a smaller maximum step size, and the step has to be performed again. If the objective function directly decreases, λ_{l+1} can be chosen smaller than λ_l (here: $\lambda_{l+1} := \lambda_l/2$). This increases the step size. For strategies to choose parameters, see (Marquardt, 1963), cf. (Hanke-Bourgeois, 2002, p. 190). The vector of residua is weighted with factor $\frac{1}{\sqrt{n}}$ to allow an initial choice of λ_0 that is independent from the varying number of correspondences. Therefore, we can apply a variant of the Levenberg-Marquardt method that is defined via the identity matrix I instead of a diagonal matrix consisting of diagonal elements of $D(\vec{a}_l)^\top \cdot D(\vec{a}_l)$.

The inner Gauss Newton or Levenberg-Marquardt iterations terminate if a convergence criterion is fulfilled after l steps: The current arithmetic mean of squared distances between pairs of corresponding points is below a threshold value (we use 10^{-4}), or the difference between this mean and the previously computed mean is below a threshold value (we use 10^{-6}), or a maximum number of iterations is exceeded. Then we have found rotation and translation parameters $\vec{a}_l$.

(Fitzgibbon, 2003) proposes an ICP algorithm that only uses one single Levenberg-Marquardt step, i.e., the maximum number of inner iterations is one. With respect to Section 4 we recommend one Gauss-Newton iteration step instead of a single Levenberg-Marquardt step.

3.3 Scaling

Now we compute an appropriate scaling factor σ_0 for current outer iteration step by minimizing $e(\sigma) := \sum_{k=1}^{3n}(\vec{R}_k(\vec{a}_l,\sigma))^2$. The necessary condition $\frac{d}{d\sigma}e(\sigma) = 0$ results in

$$\sigma_0 := \frac{\sum_{k=1}^{n}\sum_{j=1}^{3}(T_{\vec{a}_l,1}\vec{s}_k)_j \cdot (\vec{d}_k)_j}{\sum_{k=1}^{n}\sum_{j=1}^{3}(T_{\vec{a}_l,1}\vec{s}_k)_j^2}. \quad (5)$$

Let σ be the product of all scaling factors obtained by previous outer iteration steps. If $\sigma \cdot \sigma_0 \notin [1-\varepsilon, 1+\varepsilon]$ for a small threshold parameter $\varepsilon > 0$ (in our tests is $\varepsilon = 0.03$) then we replace σ_0 by $\sigma_0 := \frac{1-\varepsilon}{\sigma}$ if $e\left(\frac{1-\varepsilon}{\sigma}\right) < e\left(\frac{1+\varepsilon}{\sigma}\right)$, or $\sigma_0 := \frac{1+\varepsilon}{\sigma}$ otherwise.

3.4 Completing an Outer Iteration Step

With rotation, translation and scaling parameters found, we apply $T_{\vec{a}_l,\sigma_0}$ to all source points. This completes one outer iteration of ICP. The outer iterations have to be continued until similar convergence criteria as for the inner iterations are fulfilled: The current error $e(\sigma_0)/n$ is below a threshold value (we use 10^{-4}), or the absolute error difference between $e(\sigma_0)/n$ and the corresponding error of the previous outer iteration step is below a threshold value (10^{-6} in tests), or a maximum number of iterations is exceeded. This last criterion was not used in our tests.

After finishing with n outer iteration steps, let $T_1, \ldots, T_n$ be transformation matrices computed in steps 1 to n, and let M be the translation matrix that adds $(\overline{x}, \overline{y}, \overline{z})$ to a point in homogenous coordinates. Then $M \cdot T_n \cdots T_1 \cdot M^{-1}$ is the final matrix.

4. EXPERIMENTAL RESULTS

To measure errors, we determine the arithmetic mean of squared Euclidian distances between corresponding points, i.e., a mean squared residuum. We do not consider points that do not have a correspondence in the city model. Also, the number of corresponding points might change between ICP iterations.

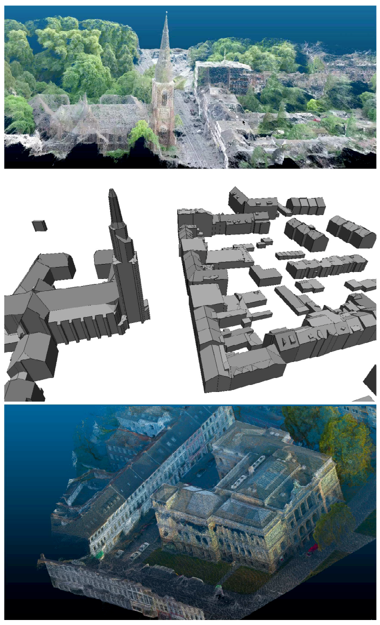
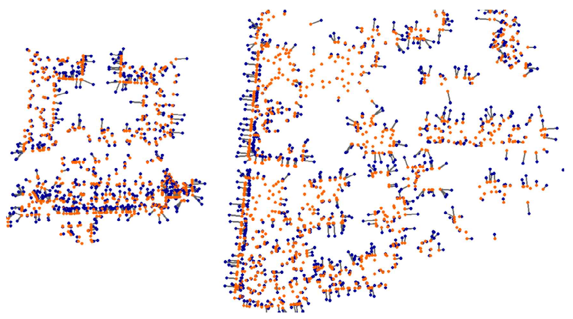
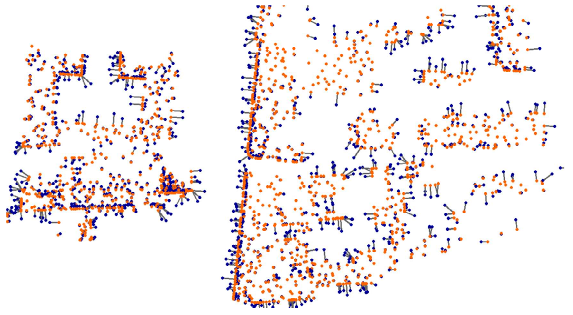


Figure 4. The upper image shows a photogrammetric point cloud of Bockum, computed from a UAV video with more than five million points, the second image shows the CityGML model that we use for registration. The point cloud of Kaiser Wilhelm museum with more than three million points is visualized in the third image, for a corresponding CityGML model see Figure 1.

We tested with two photogrammetric, manually coarsely registered point clouds that were generated from UAV videos. One point cloud covers the center of Bockum, a suburb of the German town Krefeld, see Figure 4. The other cloud represents the Kaiser Wilhelm museum of art in Krefeld, see Figure 1. We match them with models of UTM square kilometers [334.000, 335.000] × [5.691.000, 5.692.000] and [329.000, 330.000] × [5.689.000, 5.690.000], respectively. Influence of Gauss-Newton versus Levenberg-Marquardt method, maximum number of optimization iterations, and projection to bounding rectangles versus projection to real polygonal shapes is analyzed. One



Initial situation



Situation after 15 iterations using the Gauss-Newton algorithm in each iteration: One can clearly see improvement at the street front (vertical line in the middle of the images).

Figure 5. Two images showing corresponding pairs of points for the Bockum point cloud. Exact polygonal contours are used to find corresponding points.

could also compare results with ICP algorithms working on target point clouds. But to this end, one has to sample a city model to generate such a target point cloud. The outcomes are heavily dependent on the size of sampled clouds. Thus, such a comparison is difficult. However, we could compare our running times with ICP in point-to-plane mode as described in (Goebbels , Pohle-Fröhlich, 2018). In these tests, the point-to-model ICP was roughly three times faster.

Figures 5 and 6 illustrate how source point clouds align to corresponding points of the city model. To generate the figures, points were projected to the ground plane. Only 0.1% out of more than two million correspondences are shown. Blue points belong to the cloud, red points are their counter parts lying on walls and roofs of the city model. Shown correspondences are limited to a distance of 5 m. If there is a visible distance between points of a pair, a line is drawn to connect them.

ICP requires coarsely registered point clouds. Thus, computed rotation angles φ will be close to zero. For such angles, $\varphi \approx \sin(\varphi)$ and $1 \approx \cos(\varphi)$ are very good Taylor expansions. In general, several iterations of Gauss-Newton or Levenberg-Marquardt methods might be required because the objective function is replaced by a Taylor expansion that does not include higher partial derivatives of order greater one. But if such a Taylor expansion fits exactly with the objective function, then only one Gauss-Newton step is necessary to exactly find parameters for which the gradient is the zero vector and a least squares minimum is obtained. Our experiments show that this appears to happen approximately in our scenarios. With the exception of some initial iterations, almost all Gauss-Newton optimizations of test cases only consist of two inner iteration steps. The inner iterations then stop because the second step can't reduce the error by more than 10^{-6}. In this case, the Levenberg-Marquardt

Bockum point cloud, see Figures 4, 5: All runs reduce an initial error of 3.90487 to the same final error of 2.351.			
method	number of outer iterations	number of inner iterations	running time
GN	53	limited to 1	258 s
LM $\lambda_0 = 1$	95	limited to 1	440 s
LM $\lambda_0 = 1$	54	average: 4	670 s
LM $\lambda_0 = 1/16$	53	average: 2.5	452 s
LM $\lambda_0 = 1/64$	53	average: 2	412 s

Museum point cloud, see Figures 4, 6: All runs reduce an error of 6.10032 to the same final error 1.976.			
method	number of outer iterations	number of inner iterations	running time
GN	61	limited to 1	326 s
LM $\lambda_0 = 1$	91	limited to 1	488 s
LM $\lambda_0 = 1$	61	average: 4.5	816 s
LM $\lambda_0 = 1/16$	61	average: 2	483 s
LM $\lambda_0 = 1/64$	61	average: 2	484 s

Table 1. Levenberg Marquardt optimization (LM) compared with one Gauss-Newton (GN) step (more than one step does not bring improvement). Projection to bounding rectangles is used, polygonal shapes are not considered.

algorithm does not bring any improvement, see Table 1[2]. To the contrary, the parameter λ tends towards zero for which the algorithm becomes the Gauss-Newton method. We therefore recommend using only one Gauss-Newton inner iteration step. By directly limiting the number of iterations to one, we could reduce computing time by 40%. Since we start inner iterations by setting all angle parameters to zero, all partial derivatives of matrix $T_{\vec{a}}$ are either 0, 1 or -1 in this first inner iteration step. Thus, Jacobian $D(\vec{a})$ can be computed without any multiplication other than with the weighting factor $\frac{1}{\sqrt{n}}$ (that can be also omitted in case of Gauss-Newton optimization). This allows an additional speedup.

Lazy computation of distance maps appears to be inexpensive. Iterations based on real polygonal shapes are slightly slower than those using projections to bounding rectangles. But, to a certain degree, the longer running times per iteration are compensated by fewer iterations that are needed to fulfill convergence criteria.

When considering real polygonal shapes, in general the error is larger than for projections to bounding rectangles, see Figures 7 and 8. Since we only consider matching pairs in error measurement, a larger error with real polygonal shapes is natural. The larger error does not imply a worse result. For the two test point clouds, we do not observe significant differences in the outcomes of both projection approaches: If we take the final point clouds computed with projections to polygons (cf. Figures 7, 8), the mean squared distances to bounding rectangles is 2.33 for the city center and 1.98 for the museum point cloud. If we optimize using projections to bounding rectangles and also measure distances between the resulting clouds and rectangles, the errors are 2.35 and 1.98, respectively. Vice versa, if we measure distances between the aligned clouds and polygons, we obtain errors 2.55 and 2.31 for optimizing with projections to polygons, and errors 2.59 and 2.31 for optimizing with projections to bounding rectangles.

[2]Running times were measured on one 2.3 GHz i5 core of a MacBook Pro with 16 GB RAM.

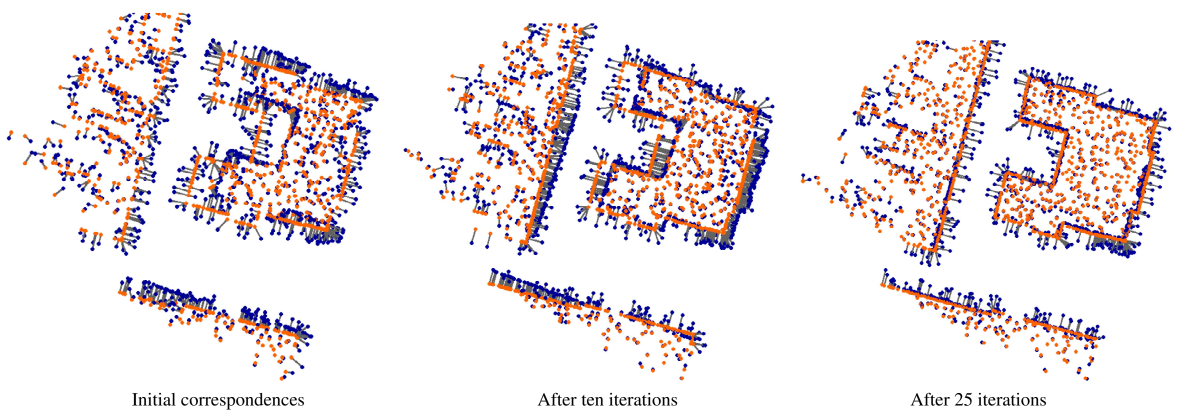


Figure 6. Using the same visualization as in Figure 5, the images show correspondences of the museum point cloud. Gauss-Newton algorithm is used in connection with projections on bounding rectangles. One can clearly see that after ten iterations many correspondences were found to correct a rotation.

5. CONCLUSION

The presented point-to-model version of ICP shows a similar but even faster behavior than the point-to-plane version requiring two point clouds. It is capable of aligning photogrammetric point clouds with CityGML models. Thus, there is no need to generate a 3D point cloud from the model for performing the registration process. It appears to be sufficient to project points to bounding rectangles instead of exact polygons of walls and roof facets. Within each (outer) iteration, only one Gauss-Newton step should be performed. In our examples, the Levenberg-Marquardt approach does not improve convergence.

We used CityGML models without terrain information. Most walls of buildings reach down to a ground plane. The z-coordinate of the ground plane corresponds with the lowest point of the building. Thus walls might intersect with terrain and have a significant part below ground. That is a potential source of error for our ICP algorithm. By additionally considering digital terrain models, results of the algorithm can be improved further.

ACKNOWLEDGEMENTS

We thank Martin Lehmann (topview4u.de), who kindly provided us with a UAV video of Krefeld's Kaiser Wilhelm museum.

REFERENCES

Besl, Paul J., McKay, Neil D., 1992. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14, 239–256.

Chen, Yang, Medioni, Gérard, 1992. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10, 145–155.

Cheng, Liang, Chen, Song, Liu, Xiaoqiang, Xu, Hao, Wu, Yang, Li, Manchun, Chen, Yanming, 2018. Registration of laser scanning point clouds: A Review. *Sensors 2018*, 18, 1–25.

Du, Shaoyi, Zheng, Nanning, Ying, Shihui, You, Qubo, Wu, Yang, 2007. An extension of the ICP algorithm considering scale factor. *Proc. 5th IEEE International Conference on Image Processing 2007*, V – 193–V – 196.

Fitzgibbon, Andrew W, 2003. Robust registration of 2D and 3D point sets. *Image and Vision Computing*, 21, 1145–1153.

Goebbels, St.J., Pohle-Fröhlich, R., 2018. Line-based registration of photogrammetric point clouds with 3D city models by means of mixed integer linear programming. *Proc. 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP) 2018*, Funchal, 299–306.

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

Hanke-Bourgeois, M., 2002. *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Teubner, Stuttgart.

Holz, D., Ichim, A. E., Tombari, F., Rusu, R. B., Behnke, S., 2015. Registration with the Point Cloud Library: A modular framework for aligning in 3-D. *IEEE Robotics Automation Magazine*, 22, 110-124.

Magnusson, Martin, Nüchter, Andreas, Lörken, Christopher, Lilienthal, Achim J., Hertzberg, Joachim, 2009. Evaluation of 3D registration reliability and speed — a comparison of ICP and NDT. *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Washington, DC, 3907–3912.

Makadia, A., Patterson, A., Daniilidis, K., 2006. Fully automatic registration of 3D point clouds. *Proc. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 1, 1297–1304.

Marquardt, D. W., 1963. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11, 431–441.

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

Zinßer, Timo, Schmidt, Jochen, Niemann, Heinrich, 2005. Point set registration with integrated scale estimation. *Proc. International Conference on Pattern Recognition and Image Processing (PRIP) 2005*, 116–119.

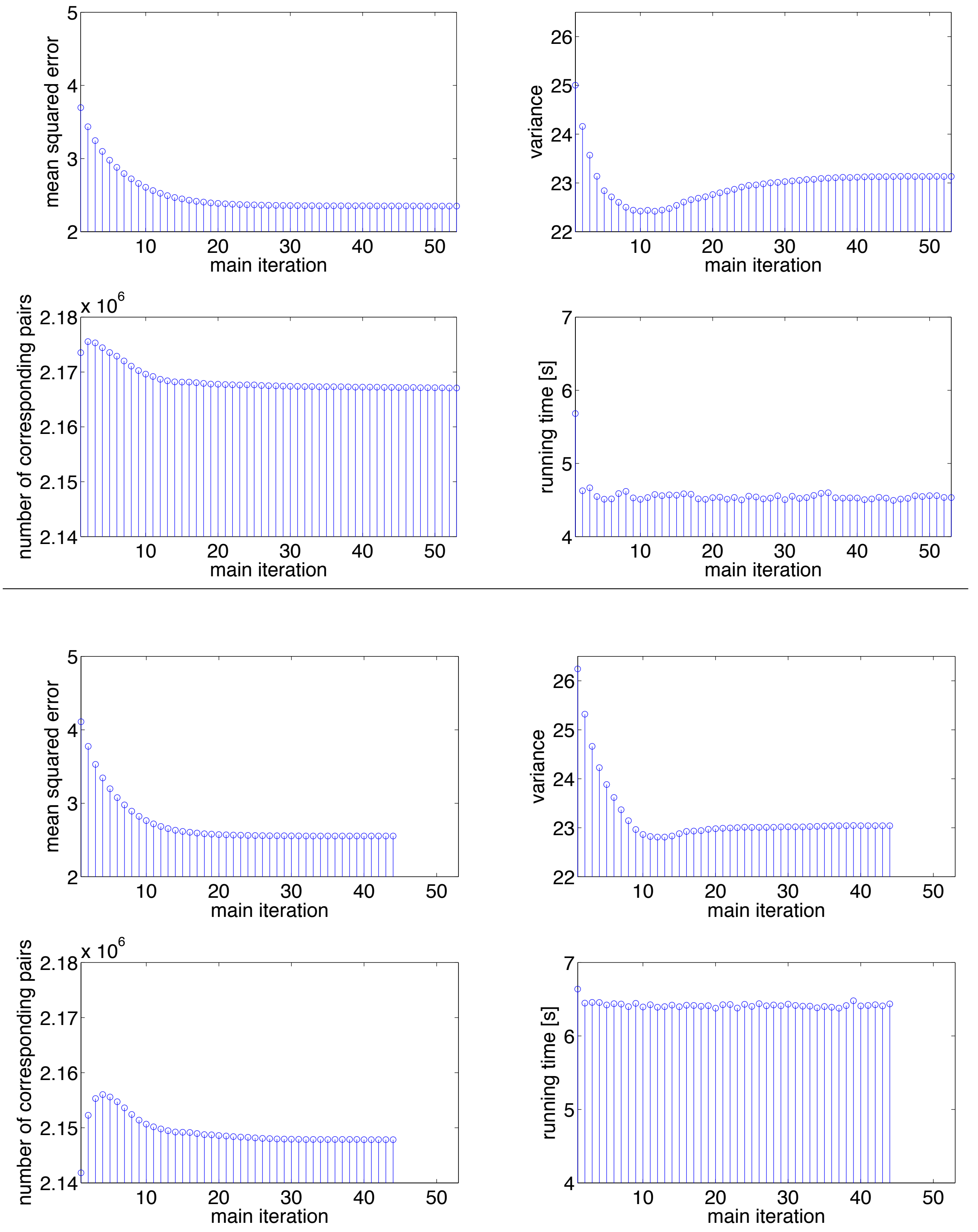


Figure 7. Computational results for the Bockum point cloud; top: projection to bounding rectangles (using one Gauss-Newton step, 53 (outer) iterations, running time: 243 s); bottom: projection to wall and roof polygons (using one Gauss-Newton step, 44 iterations, running time: 283 s)

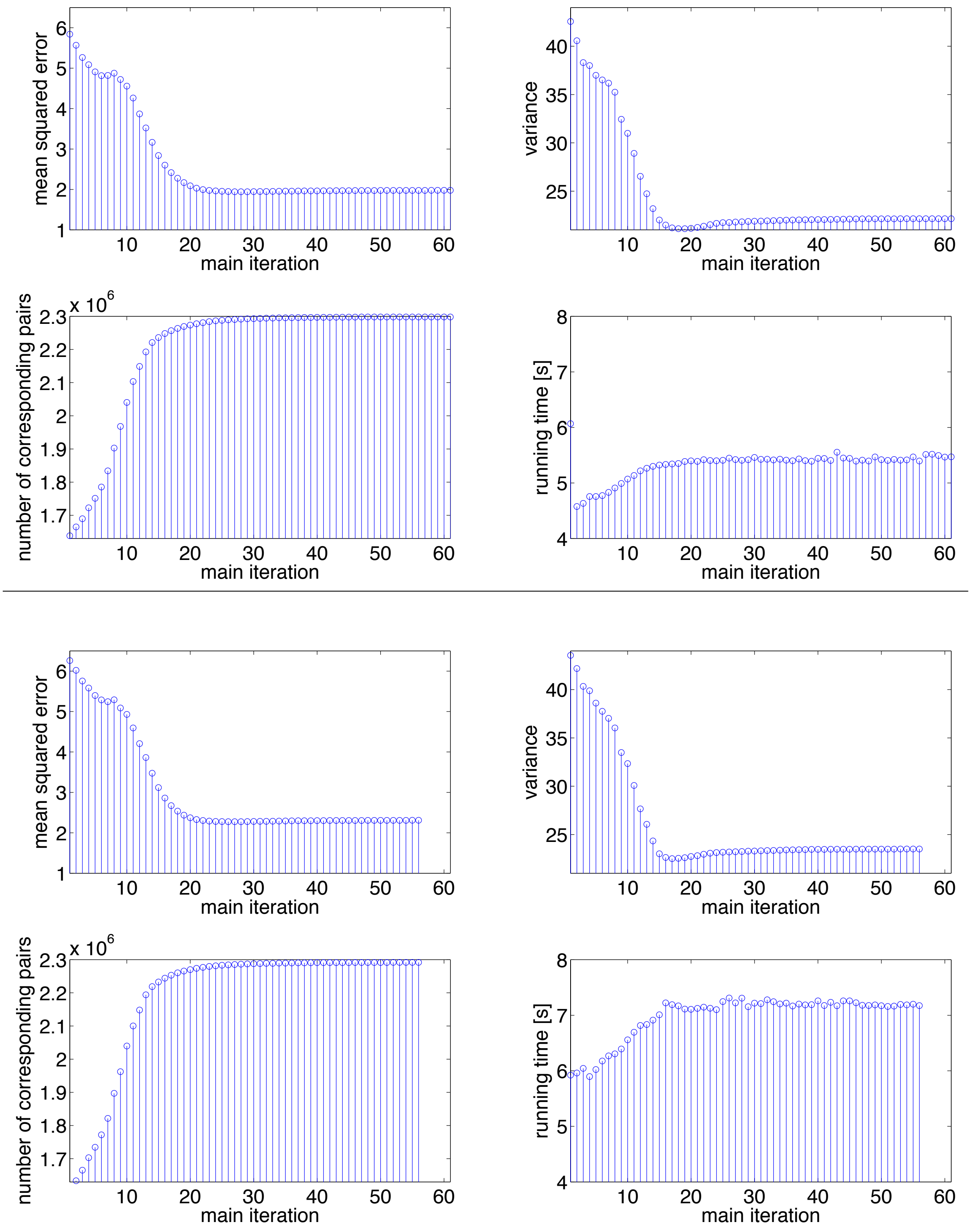


Figure 8. Computational results for the museum point cloud; top: projection to bounding rectangles (using one Gauss-Newton step, 61 iterations, running time: 326 s); bottom: projection to wall and roof polygons (using one Gauss-Newton step, 56 iterations, running time: 392 s). The increasing number of correspondences prevents the mean squared error from being decreasing strictly.