RWTH AACHEN UNIVERSITY

Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehrstuhl für Informatik VIII (Computergraphik, Computer Vision und Multimedia)
Mobile Multimedia Processing
Prof. Dr. Bastian Leibe

Master Thesis

OpenStreetSLAM: Combining visual SLAM with cadastral maps

Benito van der Zander
Matrikelnummer: 293994

September 2011

Erstgutachter: **Prof. Dr. Bastian Leibe**
Zweitgutachter: **Prof. Dr. Leif Kobbelt**

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

Aachen, den November 26, 2011

(Benito van der Zander)

Contents

Chapter 1

Introduction

In the last years several systems have been published that can process a video sequence to understand the 3D scene by either estimating the movement of the camera or building a 3D model of the scene and the visible objects, but all of these approaches are limited to the information provided by the sequence itself.

On the other hand with OpenStreetMap and Google Maps large data sets of cadastral maps have been created that contain the position and kind of almost all streets and buildings of the whole world.

We therefore strive to forge a system that can combine the local data provided by an image sequence together with the global data of the cadastral map to improve our knowledge about the local scene and even extend the map itself.
At a first step our system has to process the scene with the usual computer vision algorithms to find the path and create a 3D model, and then to align the path with the cadastral map to allow information flow between the scene and the map.
We can then display the information of the cadastral map to the user similar to an augmented reality fashion, find objects like buildings in the scene and associate them with the map to improve the scene understanding of the system, or to add the detected buildings to the cadastral map to complete missing information in it.

Our final system consists of four modules:

Visual Odometry: The visual odometry module takes a stereo image sequence as input and outputs a path describing the camera state of all frames in a local coordinate system. It estimates the camera path by finding feature points in the stereo images of each frame, triangulating 3D points from the features of the previous frame, matching the previous features against the current features, and computing a new camera position using the current 2D positions of the known 3D points.

It extends a visual odometry program previously implemented by us with an interface to accept path corrections provided by the cadastral map.

Reconstruction: The reconstruction module creates a dense 3D point cloud from the image sequence, for usage in the next stage. It basically uses an existing library

to estimate the disparity and thus the distance of each stereo pixel and calculates its 3D position from the the 2D position, the distance and the projection matrix of the camera. Afterwards it also computes normals for each 3D point and finds ground planes in the reconstruction.

Object Detection: The object detection is performed in the 3D point cloud by treating the objects as planes and searching such planes in the 3D model. Thereby it projects the complete reconstruction on the ground plane which converts buildings and rows of cars to 2D lines, detects these lines in the projected map and finally finds and classifies the plane corresponding to each line.

Matching: All our algorithms using the cadastral maps are subsumed under the matching module. The fundamental component is a positioning algorithm that compares the shape of the streets with the shape of the path and find the position of the camera in the cadastral map by searching for the most similar shape. This can be done online to correct the path while it is generated or offline to analyze an existing data sequence.

The next algorithm associates the detected objects with the buildings in the cadastral map and assigns the semantic information of the map – like street numbers – to the objects.

The last algorithm works the other way around, and adds the detected buildings at the estimated position to the cadastral map.

The remaining chapters of this thesis describe the details of each module, present an evaluation of the system and provide an outlook on possible extensions.

Chapter 2

Related work

Although our combined system is novel, some of the single components are very similar to existing publications:

Especially visual odometry is a well-studied problem, so there exist several related algorithms that can also estimate a camera path from a given image sequence. The most important ones are the system of Nistér[NNB06], the Libviso 1 library[KGL10] and its successor Libviso 2 [GZS11].
Our odometry is directly based on Nistér's paper, but we have extended it to be able to interact with a possible moving object detector by discarding areas of the image that the detector classifies as unreliable, and the cadastral map module by applying corrections to the camera position provided by another module during the path estimation.
All four odometry algorithms estimate the path by identifying identical points in the stereo images and in the images of the previous frame and compute the camera position from these matches with a RANSAC hypothesis/testing framework. The difference is that our and Nistér's system use the 3-point-algorithm [NS07] and Libviso a trifocal tensor to estimate the new camera state.
The 3-point algorithm is an algebraic, closed-form solution for the global camera translation and rotation that projects three certain 3D points given in the world coordinate system on three 2D points given in the image. The trifocal tensor presents a measurement model for point correspondences between frames and stereo images, so the actual camera state can be estimated with a non-linear Kalman filter given the positions of the features. Therefore Libviso can average over several feature points, while we can never use more than three points for a camera hypothesis, which potentially allows Libviso 1 to work better with noisy observations. However, iterating a non-linear Kalman filter introduces numerical errors and is very slow, so Libviso 2 also uses only three points for a new hypothesis.
Since these algorithms are very similar, our system can switch between our visualodometry and Libviso as odometry backend for most of the functionality.

Creating a 3D reconstruction from a known path is also an common problem, so there exist many previous solutions. Since we depend on the Libelas library, we basically reuse the system of Geiger described in [GRU10], combined with some standard

point cloud techniques. Libelas estimates disparity maps from the sequence which store for each pixel of an image the best matching pixel in the associated stereo image, by finding first safe matches and then estimating all other matches by optimizing a Bayesian energy function. We and Geiger convert this disparity then to a distance and compute the 3D point of the pixel.

No one so far has studied the combination of video sequences and publicly available cadastral maps, but Leung[LCH08] describes a related system that localize a camera in a map obtained from an aerial image. They first detect buildings in the 2D images by using a Hough transform to find lines in the images and then a vanishing point analysis to associate these lines with building facades. Afterwards they use a particle filter to find the most probable camera positions in their bird's-eye view map by defining a similarity score between the observed buildings and their map as well as a motion model for the particles. In contrast to us they use a single camera and neither create a visual odometry path nor a 3D model, and provide no bidirectional information flow which could extend the map.

Geiger again describes in [GLU11] a system to estimate the geometric and topological layout of streets crossings from a 3D reconstruction of a short image sequence. They define a generative graphical model for the layout of a junction, particularly consisting of the angles between the crossing streets, their width and the observations of the scene flow and an occupancy map. Latter map is a 2D grid that contains for every point on the ground plane, if this point is occupied, free or unobserved; which is a more sophisticated version of the ground plane projection we use in our object detection module. They then learn the model parameters from several training sequences, and perform inference using the reversible jump Markov Chain Monte Carlo algorithm.
Although such a street layout can be seen as a local cadastral map, they only describe their estimation system and do not mention cadastral or global maps, although it seems possible to use their system to localize oneself relative to the nearest junction in the global map or to directly extend cadastral street maps. However, in its current formulation their graphical model assumes that all streets are infinite long rays and can not handle sequences that contain multiple junctions.

A final approach using cadastral maps to create a reconstruction of a city is described in [SPF10]. They generate point cloud clusters from a large database of photos, and align these clusters with each other and a cadastral map. They assume that all points with a normal orthogonal to the ground plane normal are points on buildings and define an alignment score based on the inverse exponential distance between each building point and the nearest facade. This is very similar to our system, however they assume that a large part of the photos are GPS tagged and use distributed photos instead of a single video sequence. The usage of photos turns out to be an advantage, because human taken pictures are often centered on buildings and show all sides, while video sequences just happen to contain buildings in the background, which causes a lot of noise.

Chapter 3

Visual Odometry

In the visual odometry phase the exact movement of a stereo camera is estimated from a captured sequence of stereo images. Thereby a sparse reconstruction of safe stereo matches is created in each frame which is used in the next frame to estimate a camera position that projects the reconstructed points to the detected matches in the new images. Our visual odometry follows the system designed by David Nistér [NNB06].

In more details our system consists of these components:

Feature detection: A Harris corner detector is applied to the images to find the same points in all images independently.

Feature matching: These features are then mutually matched with the features of the associated stereo image and of the previous frame to find good stereo matches. The stereo match of each previous frame is triangulated to a 3d point using the known previous camera state.

Pose estimation: From the previous 3d position and the 2d position in the new frame of the features a set of new camera hypothesis is generated using the 3-point-algorithm. All hypothesis are scored with a metric based on the reprojection distance of the known points, and the best camera state is chosen as new state.

Kalman filter: Finally a constant-velocity Kalman filter is applied to the estimated camera state to reduce noise and smooth the resulting path.

3.1 Preprocessing

Before any visual odometry can be perfomed, some basic preprocessing has to be done. Each of the stereo cameras has to be calibrated by measuring the physical camera properties and defining a common coordinate system between both cameras.
Formally this means that intrinsic calibration matrices $K_l, K_r \in \mathbb{R}^{3\times3}$, extrinsic rotation matrices $R_l, R_r \in \mathbb{R}^{3\times3}$ and camera positions $T_l, T_r \in \mathbb{R}^3$ have to determined, such that

a world point $P \in \mathbb{R}^3$ is projected to $\begin{pmatrix} D_c p_{c,x} \\ D_c p_{c,y} \\ D_c \end{pmatrix} = K_c[R_c|-R_c T_c]P$ where p_c is the 2d position of the projected point and $D_c = |P - T_c|$ is the distance between the point and camera $c \in \{l, r\}$.
The calibration is usually given, but can be estimated by observing an object of known size and structure. We also assume that the initial position of left camera is at the origin of the coordinate system, i.e.: $T_l = 0_3$.
Such a calibration assumes that the camera projects all points linearly, but this is generally not true, because the form of the lens introduces a radial distortion. We can then measure this distortion and undistort the images by transforming them to a new image which satisfies the projection equation almost exactly, as the example of Fig. 3.1. It is necessary to remove this distortion to get precise results from the visual odometry.

It is also useful – although not essential – to rectify the images by transforming

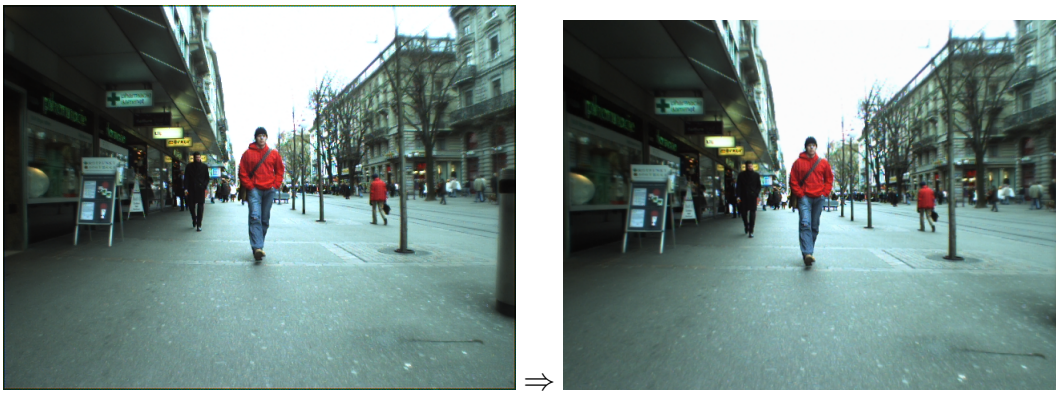


Figure 3.1: Example for an image undistortion

them so that the epipolar lines are parallel to x-axis. Then a projected point has the same y-coordinate in both stereo images which greatly simplifies the search for stereo matches, because one only has to search along the x-axis, and reduces the probability of mismatches. An example rectification is shown in 3.2.

3.2 Feature detection

To actually use the stereo information it is necessary to identify identical points in both images. Hence we search features in the images, points that have unique properties. There are many known feature detectors, and we have decided to use the Harris feature detector which detect corners in the images. It has been shown that the Harris corner detector has a high repeatability [SMB00] of finding the same points regardless of viewport changes and is therefore ideal to find features for stereo matches.

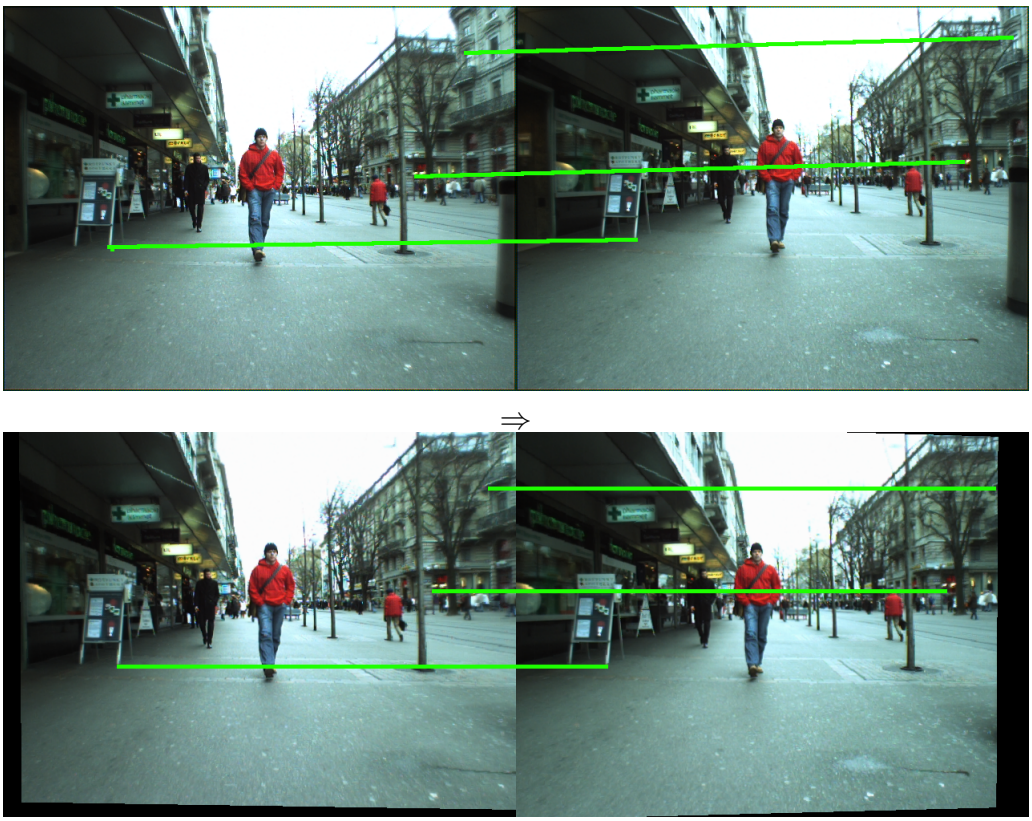


Figure 3.2: Example of an image rectification that aligns stereo correspondences with the x-axis

Basically the Harris detector considers for each point $p = (p_x, p_y)^t$ the matrix

$$A_{X,Y} = \begin{pmatrix} \left\langle I^x_{X,Y} \cdot I^x_{X,Y} \right\rangle & \left\langle I^x_{X,Y} \cdot I^y_{X,Y} \right\rangle \\ \left\langle I^x_{X,Y} \cdot I^y_{X,Y} \right\rangle & \left\langle I^y_{X,Y} \cdot I^y_{X,Y} \right\rangle \end{pmatrix},$$

where I^x, I^y are the image derivatives in x and y direction, $I^x \cdot I^y$ denotes the component-wise product between the two derivatives, and $\left\langle I^x_{X,Y} \right\rangle$ is the mean of all components of I^x over the window X,Y with area w^2 and $X = \{p_x - w/2, \dots p_x + w/2\}, Y = \{p_y - w/2, \dots p_y + w/2\}$.
The image derivatives describe the local change of the image when going 1 pixel in x or y direction, so the average over a $w \times w$ window can be seen as the change along w pixels. The eigenvectors of this matrix correspond to the direction of the change, and the eigenvalues to the strength of the change. So if both eigenvalues are large, the point is a corner and marked as a potential feature.
Faster than directly calculating the eigenvalues λ_1, λ_2 is it to calculate $\lambda_1\lambda_2 + \kappa(\lambda_1 + \lambda_2)^2 = \det(A_{X,Y}) + \kappa \operatorname{trace}(A_{X,Y})^2$ with an empirical parameter κ.
This also results in a simple score to directly compare the "cornerness" of two different pixels which helps to remove all weaker corners in the neighbour of a detected strong corner. This is necessary to prevent the creation of several features for a single point.
Usually the strongest Harris corners are concentrated on high entropy regions of the image, which is bad for visual odometry features, because it gives more weight to features on certain unstable objects than to the static scene background. To prevent this, the image is split into a grid of bins and Harris corners are separately detected in each bin, until each bin contains a certain count of features or until no features can be found anymore.

The feature detector can be combined with information from other system compo-

Figure 3.3: Detected features

nents, e.g. a pedestrian detector, which marks parts of the images as noisy or non-static and therefore useless for the visual odometry. These components have to provide a set of bounding boxes and the feature detector will just ignore all points inside these boxes.

3.3 Feature matching

For each feature position p a square patch of $\sqrt{n} \times \sqrt{n}$ pixels centered around p is extracted, as shown in 3.4. Each patch is represented by an n-dimensional vector in which each entry is the gray value of one of the pixels in the patch. It is also possible to use a $3n$-dimensional vector with color information or sub-pixel matching where the entries are combinations of the values of adjacent pixels, but this usually has only an almost negligible effect on the results.
The score of the similarity between two patches f^l and f^r is calculated using normal-

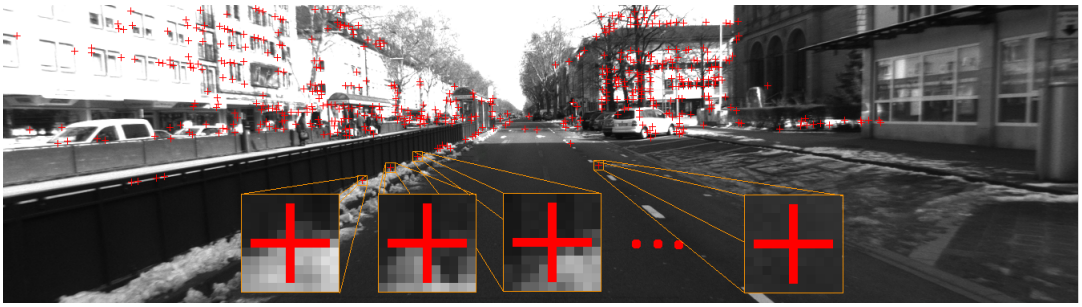


Figure 3.4: Illustration of extracted feature patches

ized cross correlation which defined as:

$$NCC(f^l, f^r) = \frac{1}{n} \sum_{k=1}^{n} \frac{\left(f^l(k) - \left\langle f^l \right\rangle\right)\left(f^r(k) - \left\langle f^r \right\rangle\right)}{\sigma(f^l)\sigma(f^r)},$$

where $f^l(k)$ is the k-th entry of the vector f^l, $\left\langle f^l \right\rangle = \frac{1}{n}\sum_{k=1}^{n} f^l(k)$ the mean and $\sigma(f^l) = \sqrt{\frac{1}{n}\left\langle (f^l - \left\langle f^l \right\rangle)^2 \right\rangle}$ the standard derivation of f^l.
The advantage of normalized cross correlation is that it invariant against affine changes, i.e. brightness or contrast changes. This can be proven by applying an affine transformation to the patch f. This creates a new patch with entries $f'_i = af_i + b$ which lead to the same factors in the sum of the cross correlation as the unmodified patch f:

$$\frac{f'(k) - \langle f' \rangle}{\sigma(f')} = \frac{af(k) + b - a\langle f \rangle - b}{a\sigma(f)} = \frac{a}{a}\frac{f(k) - \langle f \rangle}{\sigma(f)}.$$

As a trivial optimization each patch is normalized by precomputing factors $g^l(k) = \frac{(f^l(k) - \left\langle f^l \right\rangle)}{\sigma(f^l)}$, which simplifies the calculation of the correlation to:

$$NCC(f^l, f^r) = \sum_{k=1}^{n} g^l(k) g^r(k) = \left\langle g^l, g^r \right\rangle.$$

The NCC score is then used to find for each feature patch f_i^l of the left image the most

similar feature patch among all patches f_j^r of the right image in the neighbourhood around p_i^l; and similarly a match for f_j^r among all features f_i^l:

$$match_i^l = \underset{j \in N^r(p_i^l)}{\operatorname{argmax}} NCC(f_i^l, f_j^r),$$

$$match_j^r = \underset{i \in N^l(p_j^r)}{\operatorname{argmax}} NCC(f_i^l, f_j^r).$$

The neighbourhood $N^r(p_i^l)$ is the set of all indices of features in the right image which are near to the point p_i^l:

$$N^r(p_i^l) = \{j \in \mathbb{Z} || p_i^l - p_j^r| \leq d_2 \wedge |p_{i,x}^l - p_{j,x}^r| \leq d_x \wedge |p_{i,y}^l - p_{j,y}^r| \leq d_y\},$$

where d_2, d_x, d_y are thresholds for the corresponding norm. In rectified images all correct matches have the same y-position, so we choose $d_y \approxeq 0$ and $d_2 \approxeq d_x$. In unrectified images the epipolar lines are not parallel to the images axis and we tend to use only d_2 with $d_x = d_y = \infty$.
Here, high thresholds can lead to a high count of invalid matches, while low thresholds fail to find matches, when fast rotations occur. In an implementation we can use the fact that all features are contained in bins as optimization and only search for features that are in near bins.
To improve the robustness of the matches, we only accept mutual matches (i, j), i.e. matches that satisfy the condition:

$$match_i^l = j \wedge match_j^r = i.$$

We have now a set of stereo matches as well as the camera position – from the initial calibration or the previous frame estimation –, so we can find a 3D point to each matching pair. To triangulate the 3D position we take the usual projection equation for both images of a 3D point P in distances $D_i^l = |P_i - T_l|, D_i^r = |P_i - T_r|$ from each camera and camera projection matrices $P_l = K_l[R_l| - R_l T_l], P_r = K_r[R_r| - R_r T_r]$:

$$D_i^l \begin{pmatrix} p_{i,x}^l \\ p_{i,y}^l \\ 1 \end{pmatrix} = P_l \begin{pmatrix} P_{i,x} \\ P_{i,y} \\ P_{i,z} \\ 1 \end{pmatrix}, \qquad D_i^r \begin{pmatrix} p_{j,x}^r \\ p_{j,y}^r \\ 1 \end{pmatrix} = P_r \begin{pmatrix} P_{i,x} \\ P_{i,y} \\ P_{i,z} \\ 1 \end{pmatrix}.$$

The dependency on the unknown distance D_i^c can be removed by taking the crossproduct with the respective projected point, which simplifies the equations to:

$$0 = \begin{pmatrix} p_{i,x}^l \\ p_{i,y}^l \\ 1 \end{pmatrix} \times D_i^l \begin{pmatrix} p_{i,x}^l \\ p_{i,y}^l \\ 1 \end{pmatrix} = \begin{pmatrix} p_{i,x}^l \\ p_{i,y}^l \\ 1 \end{pmatrix} \times P_l \begin{pmatrix} P_{i,x} \\ P_{i,y} \\ P_{i,z} \\ 1 \end{pmatrix},$$

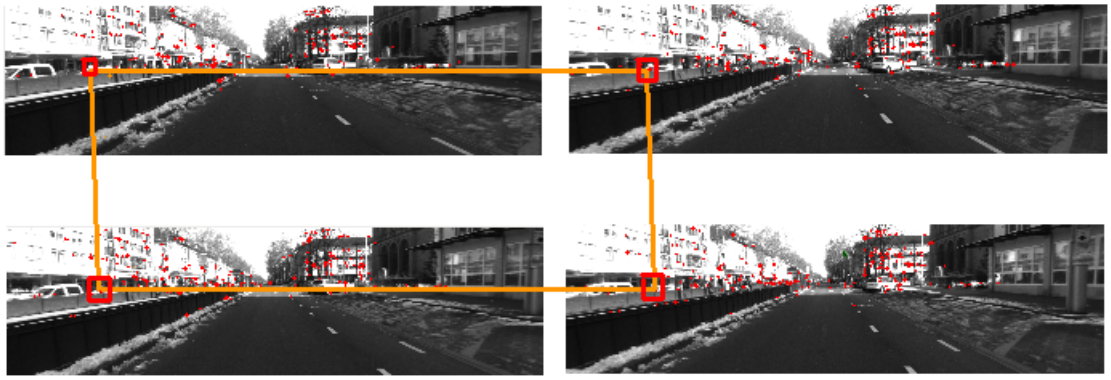


Figure 3.5: Matching features against the between frames and the corresponding next frame

$$0 = \begin{pmatrix} p^r_{j,x} \\ p^r_{j,y} \\ 1 \end{pmatrix} D^r_i \times \begin{pmatrix} p^r_{j,x} \\ p^r_{j,y} \\ 1 \end{pmatrix} = \begin{pmatrix} p^r_{i,x} \\ p^r_{i,y} \\ 1 \end{pmatrix} \times P_r \begin{pmatrix} P_{i,x} \\ P_{i,y} \\ P_{i,z} \\ 1 \end{pmatrix}.$$

The cross product with a known vector can be further reduced to a matrix multiplication which converts the equations to a normal linear equation system:

$$0 = \begin{pmatrix} 0 & -1 & p^l_y \\ 1 & 0 & -p^l_x \\ -p^l_y & p^l_x & 0 \end{pmatrix} P_l \begin{pmatrix} P_{i,x} \\ P_{i,y} \\ P_{i,z} \\ 1 \end{pmatrix}, \qquad 0 = \begin{pmatrix} 0 & -1 & p^r_y \\ 1 & 0 & -p^r_x \\ -p^r_y & p^r_x & 0 \end{pmatrix} P_r \begin{pmatrix} P_{i,x} \\ P_{i,y} \\ P_{i,z} \\ 1 \end{pmatrix}.$$

Stacking both equations creates a single matrix whose nullspace is spanned by $(P,1)^t$. Since the matrix usually contains some noise, the smallest eigenvector instead of a non-existing nullspace has to be calculated using singular value decomposition (SVD). This step is quite sensible to numerical precisions, even switching the SVD implementation (e.g. between GSL and Eigen) can cause the final estimated path to change by several metres.

This matching is repeated between the stereo images of the next frame, as well as between the current frame and the next frame. This results in a mutual matching across time and space as shown in Fig. 3.5.

As the neighbourhood of a possible feature matches in the next frame, all features in a certain distance around the position of the feature in the current frame could be used, i.e. $N^{new}(p^{old})$. However it is more reliable to project the known 3D point into the next frame. This is not actually possible, because it requires the knowledge of the camera state of the next frame, but a good estimate can be made

by predicting the camera position T^{new} of the next frame using a Kalman filter as described in section 3.5. Then we get the neighbourhood of features in the next frame as $N^{new}\left(\frac{(K[R|-RT^{new}]P)_{xy}}{(K[R|-RT^{new}]P)_z}\right)$. Through this matching we learn the 3D position P of the point belonging to a stereo match p^l, p^r in the next frame, which can then be used to estimated the new camera position by finding the projection matrices which project P to p^l and p^r using the algorithm from the next section.

3.4 Pose estimation

The current camera state can be estimated using a RANSAC hypothesis-testing framework, once a set of stereo matches with associated 3D position $S = \{(p_i^l, p_i^r, P_i)\}$ is known[1]. Thereby a set of camera hypothesis is generated from triples of points, scored with all the other points, and then the best hypothesis is chosen as new camera state.
Although a triple is sufficient to generate a camera hypothesis the algorithm chooses several quadruples (i_1, i_2, i_3, i_4) of random points and uses the fourth point to perform basic checks on the hypothesis.
The number N of generated quadruples is a compromise between speed and accuracy: A high N increases the probability to generate a correct hypothesis, but can be slow, since the runtime is linearly proportional to N.

The three projection equations

$$\begin{pmatrix} Dp_{i_k}^l \\ D \end{pmatrix} = K_l[R_l|-R_lT_l]\begin{pmatrix} P_{i_k} \\ 1 \end{pmatrix} \Leftrightarrow (K_l)^{-1}\begin{pmatrix} Dp_{i_k}^c \\ D \end{pmatrix} = [R_l|-R_lT_l]\begin{pmatrix} P_{i_k} \\ 1 \end{pmatrix}$$

with $k \in \{1,2,3\}$, are then solved for the camera state R_l and T_l of the left camera using the 3-point algorithm described in [NS07]. Due to the ambiguity of the point arrangement, the 3-point-algorithm can not return a definite solution, but it returns up to 8 possible solutions each leading to a separate hypothesis.
From the state of the left camera R_l, T_l, the state of the right camera R_r, T_r can be calculated, because the relative rotation $(R_l)^{-1}R_r$ and the relative translation $(R_l)^{-1}(T_l - T_r)$ remain invariant over all frames and are given by the initial calibration.
We then use the point P_{i_4} to perform the basic checks, i.e. to test if P_{i_4} is actually in front of all cameras. This is the case if and only if the forward vector f_c obtained from the third row of the rotation matrix R_c points in the direction of P_{i_4}, i.e. if and only if

$$\left\langle f_c, P_{i_4} - T_c' \right\rangle > 0.$$

This check removes all those invalid hypotheses, whose camera is rotated by 180 degrees, which are around half of the hypotheses.

[1] It is no longer necessary to differentiate between an index i of features in the left image and an index j of features in the right image like in the previous section, because we can assume that all matched points were sorted accordingly.

Another important check is to specify a maximal allowed velocity v_{max} for the camera and ignore all hypotheses that do not satisfy

$$|T' - T^{old}| < \frac{v_{max}}{fps}.$$

v_{max} should be large than the maximal possible true velocity, so we can choose around $10\frac{km}{h}$ for pedestrians and $300\frac{km}{h}$ for cars.

Instead of the distance to the old position, we can also use the distance $|T' - T^{pred}|$ between the new estimate and the position predicted by the Kalman filter, as described in Section 3.5.Then $\frac{v_{max}}{fps}$ becomes the maximal expected error of the Kalman filter which should be lower than the usual v_{max}. This removes more wrong hypotheses, but could remove correct hypotheses if fast, unexpected accelerations occur.

After the basic checks have removed obviously wrong hypotheses, it is necessary to choose the best of the remaining hypotheses H. Therefore for each hypothesis $(R'_l, T'_l, R'_r, T'_r) \in H$ a score has to be calculated, for which we use the reprojection error of all known points, the distance between the 2d true position and the 2d position according to the hypothesis. There are several possible ways to create a score, based on the reprojection error, e.g.:

$$SCORE_{outlier_1}(R'_l, T'_l, R'_r, T'_r) = \left| \left\{ (P_i, c) | \left| \frac{(K[R'_c| - R'_c T'_c] P_i)_{xy}}{(K[R'_c| - R'_c T'_c] P_i)_z} - p_i^c \right| < e \right\} \right|$$

$$SCORE_{outlier_2}(R'_l, T'_l, R'_r, T'_r) = \left| \left\{ P_i | \left| \frac{(K[R'_l| - R'_l T'_l] P_i)_{xy}}{(K[R'_l| - R'_l T'_l] P_i)_z} - p_i^c \right| < d \wedge \left| \frac{(K[R'_r| - R'_r T'_r] P_i)_{xy}}{(K[R'_r| - R'_r T'_r] P_i)_z} - p_i^c \right| < e \right\} \right|$$

$$SCORE_{error}(R'_l, T'_l, R'_r, T'_r) = \sum_{P_i, c} \left| \frac{(K[R'_c| - R'_c T'_c] P_i)_{xy}}{(K[R'_c| - R'_c T'_c] P_i)_z} - p_i^c \right|$$

$SCORE_{outlier_1}$ and $SCORE_{outlier_2}$ define a maximal allowed error e and count all inliers . The difference between them is that $SCORE_{outlier_1}$ counts inliers to both cameras independently, while $SCORE_{outlier_2}$ only counts points which are inliers of both cameras simultaneously.

$SCORE_{error}$ calculates the sum of all reprojection errors and hence is sensible to outliers, but this aspect can be improved by summing not over all possible (P_i, c), but only over the best 50% and ignoring the other half. It has the advantage that it doesn't not depend on a correctly chosen distance threshold.

Each score requires the calculation of several square roots which is quite slow, if performance is very important, it is possible to replace it with the logarithm or the absolute.

After a scoring function has been chosen, it has to be evaluated for each hypothesis in order to find the best hypothesis:

$$(R_l^{new}, T_l^{new}, R_r^{new}, T_r^{new}) \leftarrow \underset{(R_l', T_l', R_r', T_r') \in H}{\operatorname{argmax}} SCORE(R_l', T_l', R_r', T_r')$$

It is also possible to generate new hypotheses using only the inliers to the hypothesis $(R_l^{new}, T_l^{new}, R_r^{new}, T_r^{new})$ which could find a better hypotheses, because the probability to choose an invalid point is lower among the inliers than among all points. One could also use an adaptive RANSAC which discards points or hypotheses which have a low count of supporting points during the evaluation, before calculating all reprojection errors, and runs therefore faster.

The best hypothesis is then used as an observation for the Kalman filter described in the next section and the new state estimate returned by the Kalman filter is used as new camera state.

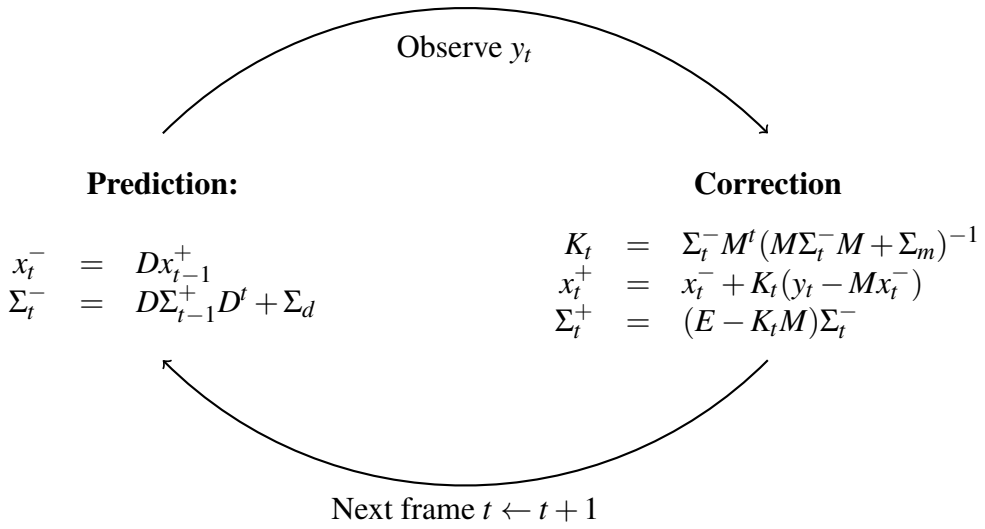
3.5 Kalman filter

The Kalman filter models the dynamic between the progression of a hidden state x and its associated partial observable y with linear equations. The task of the Kalman filter in the visual odometry is twofold: It smooths the estimated path by removing noise and provides predictions that are used as constraints on new hypotheses.

To apply a Kalman filter to a certain model, it is necessary to define a transition matrix D which describes the progression of the hidden state x over the time as $x_t = Dx_{t-1} + noise$ and a measurement matrix M which transforms the hidden state to the observable state y like $y_t = Mx_t + noise$. The noise of the progression and measurement can be specified with the covariance matrices Σ_d and Σ_m.

The Kalman filter then iterates through two steps: First a new state x_t^- is predicted from the old state x_{t-1}^+ and then it is corrected to a new state x_t^+ using the observation y_t. In each steps also the covariance matrices Σ_t^-, Σ_t^+ which describe the uncertainty of the corresponding states are updated.

Formally these two steps are described by the following equations[May79]



There are two important models, the 0-velocity model that assumes that the tracked object does not move at all and the constant velocity model that assumes that the object moves with a constant velocity.

In both cases the only observed state is the 3-dimensional vector of the object position

$$y = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix},$$

while the hidden state can be described by a 6-dimensional position/velocity vector[2]:

$$x_{0-vel} = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad x_{const-vel} = \begin{pmatrix} P_x \\ P_y \\ P_z \\ V_x \\ V_y \\ V_z \end{pmatrix}$$

.

Then the measurement matrix M just returns the first three rows of the state:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

[2]The 0-velocity vector will be implemented as a 3-dimensional vector, but a 6-dimensional vector simplifies the notation.

In the 0-velocity model the state does not change, so the update matrix D is just the identity matrix. In the constant velocity model the velocity remains unchanged, but is added to the position:

$$D_{0-vel} = E_{6,6} \qquad D_{const-vel} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The visual odometry only use the constant velocity model, but other parts of the system will use the 0-velocity model.

Any camera position estimated by the visual odometry is passed as observation to the constant velocity filter and the corrected position of the filter is then used as new camera position. Only the actual position is considered, the estimated rotation is always independent of the Kalman filter.

The constant velocity model is the closed simple model to the camera movement, because the camera only accelerates slowly and reaches a maximal velocity soon.

The 0-velocity model is useful to track stationary points and discard those which have a high uncertainty. It could be also used to validate the triangulated points in the visual odometry.

Sometimes the visual odometry completely fails and does not find any camera position, e.g. if frames are missing or the camera is obstructed, then the Kalman filter can propagate the camera straight forward independently of the odometry until latter recovers. Thereby the prediction of the Kalman filter itself is used as new, highly uncertain observation.

3.6 Implementation

This visual odometry has been implemented in a C++ command line utility, that takes the name of a calibration file, the names of all images and additional options as arguments and prints the estimated paths to stdout. This makes it easy to use in batch runs and to process the output in other applications.

The calibration file contains the matrices $K_l, K_r, [R_l|-R_lT_l], [R_r|-R_rT_r]$ in this order as row-major space-separated ascii matrices. We only tested it in cases that satisfy $T_l = 0_3$ and have a relative camera displacement along the x-axis. Fig. C.1 shows an example calibration.

The output consists of a list of world transform matrices

$$H_t = \begin{pmatrix} R_t & -R_tT_t \\ 0 & 1 \end{pmatrix}$$

relative to the initial positions for each frame t in the same format as matrices of the calibration file. The projection in projective coordinates of any point P in frame t with camera $c \in \{l, r\}$ is then $K_c[R_c| - R_c T_c]H_t P$.

We use the coordinate system shown in Fig. 3.6 where the up-vector is parallel to the y-axis, the right camera has a displacement along the positive x-axis, so that the camera path moves in the direction of the positive z-axis.

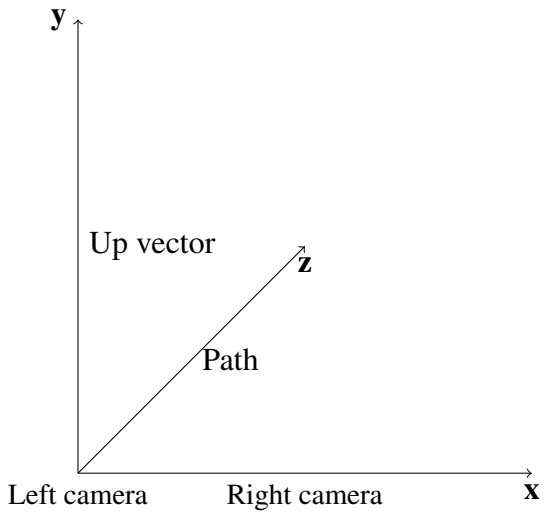


Figure 3.6: Coordinate system

Chapter 4

Reconstruction

In the reconstruction phase a dense point cloud is created from the image sequence by triangulating all pixels in the 3d space.

Such a reconstruction is a 3d representation of the scene and useful to process the scene in a global way, e.g. it can be used to create a visualization from different points of view or to detect three-dimensional objects like buildings or streets.

Our point cloud reconstruction is just a set of 3d points each annotated with a color and an estimated normal like a voxel model. There are other kinds of reconstructions namely meshes reconstructions which create a more solid, polygon representation of an object or a popup reconstructions which places parts of the images as 2d sprites in the 3d space. The point cloud reconstruction has the advantage that it is easy to create and does not make any assumption about the structure of the scene.

4.1 Creating the point cloud

The visual odometry has already created a reconstruction of the scene, but it is not a *dense* reconstruction, because it only contains the points that could be matched by the Harris corner detector with high reliability.

We therefore use another approach which is based on the Libelas library. Before the reconstruction is created, all images are preprocessed with Libelas to create disparity map for each frame. Libelas first finds a set of support points along the epipolar lines that can be reliable triangulated and then optimizes a Bayesian energy function for all other pixels to find the remaining disparities[GRU10].

Once the disparity maps are created the reconstruction of a single point is performed as follows:

The projection p of a 3d point P to a camera image is given by the usual projection equation:

$$\begin{pmatrix} Dp_x \\ Dp_y \\ D \end{pmatrix} = K[R|-RT] \begin{pmatrix} P \\ 1 \end{pmatrix} = KR(P-T),$$

where D is the distance between the point and the camera center.

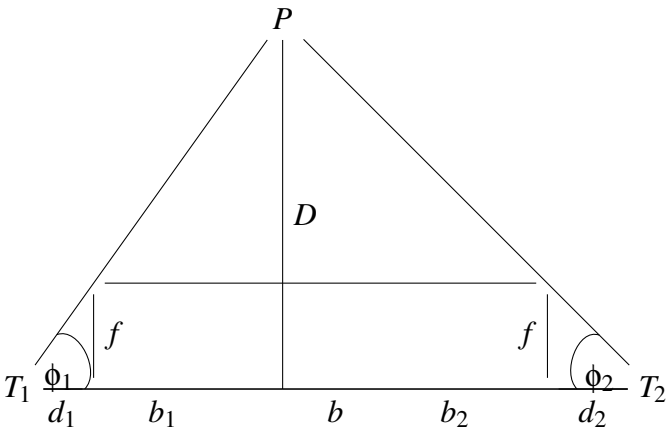
Since K and R are invertible $3x3$ matrices and known from the calibration and the odometry, this equation can be inverted to:

$$P = R^{-1}K^{-1}\begin{pmatrix} Dp_x \\ Dp_y \\ D \end{pmatrix} + T$$

The distance D is given by $D = \frac{fb}{d(p)}$ with the focal length f, the baseline b from the calibration and the disparity $d(p)$ calculated by libelas as shown in Fig. 4.1. This leads to the final reconstruction equation:

$$P = \frac{fb}{d(p)}R^{-1}K^{-1}\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} + T$$

In some cases Libelas fails to estimate a disparity, e.g. if an object is occluded in one of the stereo images. Then no point can be added to the point cloud.



$$d = d_1 + d_2 = f\frac{d_1}{f} + f\frac{d_2}{f} = f\cot\phi_1 + f\cot\phi_2 = f\frac{b_1}{D} + f\frac{b_2}{D} = \frac{f}{D}(b_1 + b_2) = \frac{bf}{D}$$

Figure 4.1: Relationship between disparity and distance of a 3d point[SS06]

Therefore a dense reconstruction can be created by applying this equation to every pixel of every frame. In order to reduce the computation time and memory usage, however, only pixels on a $i \times i$ grid are sampled which results in an $O(i^2)$ performance increase.

4.2 Refinement

Another way to reduce the number of useless points is to merge identical points and remove points that were not reliable triangulated. This can be done in two steps, first the points are tracked over time and then the uncertainty of the point position along the tracked path is estimated with a Kalman filter.

To track a 3d point it is reprojected on the next frame and its new estimated disparity compared with the disparity obtained from the distance between the point and the new camera position. If these disparities are sufficient close, both 3d points can be fused into a single point[GZS11].

Thus we calculate with the estimated P at the next camera position R', T' a new projection

$$\begin{pmatrix} D'p' \\ D' \end{pmatrix} = KR'(P - T')$$

and test if

$$\left| \frac{d'(p') - \frac{bf}{D'}}{d'(p')} \right| < \varepsilon$$

holds. If it does not, the point is discarded. Otherwise the last fused point is added to the point cloud, i.e. $P' = \frac{fb}{d'(p')} R^{-1} K^{-1} (p_x, p_y, 1)^t + T$, if P' is not again fused with another point. ε is some threshold ensuring that only near points are fused. We can also set $\varepsilon = \infty$ to only check if the disparity $d'(p')$ exists.

To reduce the noise of the reconstruction further it is useful to employ a 0-velocity Kalman filter as described in section 3.5. This model assumes that the true position is stationary and the Kalman filter only updates its covariance matrices which represents the uncertainty of the observation. All points that have been fused into a single point are then taken as observations of the same true point. If the covariance of this point becomes too large, the point is discarded, which should eliminate noise caused by moving objects or misidentified fusions.

After the position of all points in the point cloud has been estimated as shown in figure 4.2, more global properties can be calculated, namely the normals and the ground planes:

4.3 Normals

To compute the normal n of a point P, we assume that the associated surface of the point is approximately planar, and take as point normal the normal of a plane fitted to the 32 nearest points around P[SPF10].

If the points are exactly given, the normal n of the plane is the nullspace vector of the covariance matrix

$$COV = \sum_{v \in V} (\langle V \rangle - v)(\langle V \rangle - v)^t$$

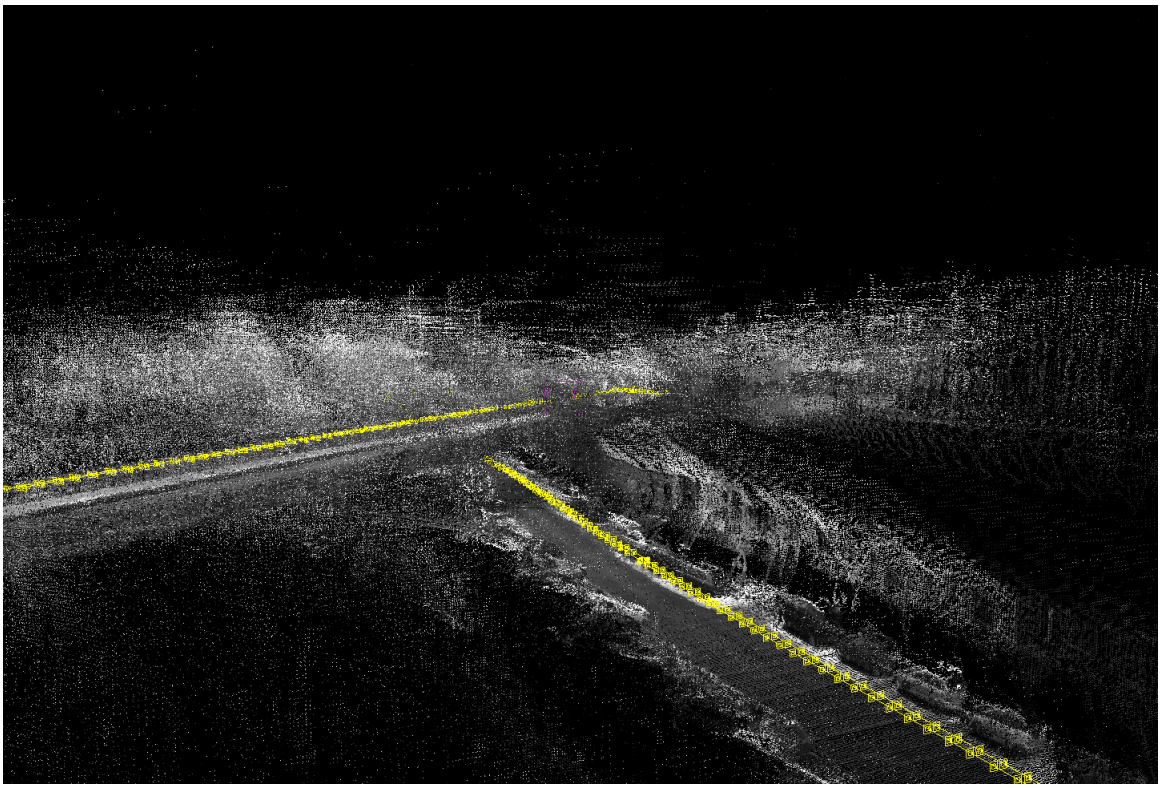


Figure 4.2: Reconstruction with path

with the mean $\langle V \rangle = \frac{\sum_{v \in V} v}{|V|}$, because $\langle V \rangle$ is then a point on the plane, and $\langle V \rangle - v$ a vector on the plane which satisfy

$$COV \cdot n = \sum_{v \in V} (\langle V \rangle - v)(\langle V \rangle - v)^t n = \sum_{v \in V} (\langle V \rangle - v)0 = 0.$$

If the points are noisy, the nullspace of COV does not exist, but it can be approximated by the smallest eigenvector of COV, which leads to a least-square estimation of the normal.

Finding the neighbourhood for each point is quite slow and can be optimized by using an octtree that splits the point cloud into spatial related parts and searching only in near nodes.

To create a simple octtree an axis aligned bounding box is placed around all points which becomes the root node of the octtree and is split at its center into 8 subnodes of the same size. Each subnode contains a subset of the point cloud and is again split in the same way if it contains more points than a certain threshold. Fig. 4.3 shows a 2d projection of an octtree where each node is split until it contains maximal two points.

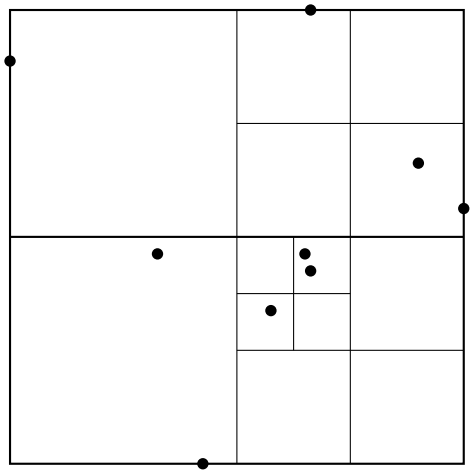


Figure 4.3: Quadtree as 2d octtree example

The nearest neighbours are usually found in the same node or in neighbouring nodes and can be found by continuing the search recursively in the children or the parent node, until the distance to all other nodes is greater than the distance to the already found points. A possible alternative way to perform the splitting is to split the node at the mean or the median of the contained points instead of the node center.

Because the point cloud contains several outliers the bounding box is only fitted to around 99% of the points and the remaining points are discarded. Thereby it is centered at the mean of the point cloud and enlarged, until its projection on each axis covers 99% of the points. The corresponding size b_k of the box is the minimal b_k satisfying

$$|\{v \in V || \langle V \rangle_k - v_k| < b_k\}| \approx 0.99|V|$$

and can be found by linear (or binary) search.

After all normals have been calculated, they can be visualized as short lines emerging from each point as shown in figure 4.4.

4.4 Ground plane

Finding the ground plane, or any plane, requires the estimation of the plane normal and an offset along that normal.

It appears that most of the reconstructed points are laying directly on the ground and have a correctly reconstructed normal, so the average or the most common point normal should correspond the ground plane normal. However, a faster way to find the normal is to assume that the camera has a constant height and fit a plane to the camera positions, like described above. To make the least square estimation robust against outliers, the camera path has to be split into several short pieces that each have a separate normal fitted to them. Then the normalized mean of all these local normals can be used as the global normal. [1]

Once the normal is known, we can use the fact that most points belong to the ground by projecting all points on a 1-dimensional line parallel to that normal. This results in a 1-dimensional histogram, and the bin which contains the most points corresponds to the searched plane offset. A resolution of around 10cm seems to be reasonable for the histogram, and to improve the precision the offset can be set to the average offset of all the points in that bin. As an optimization points above the camera position may be ignored.

These steps can be performed once to create a global ground plane, if it exists, or repeatedly to create local ground planes for each group of frames like the example in figure 4.5. We can also use these ground planes as hypotheses in a hypotheses-testing framework, but this is not necessary.

[1] With a correct calibration this usually turns out to be the y-axis.

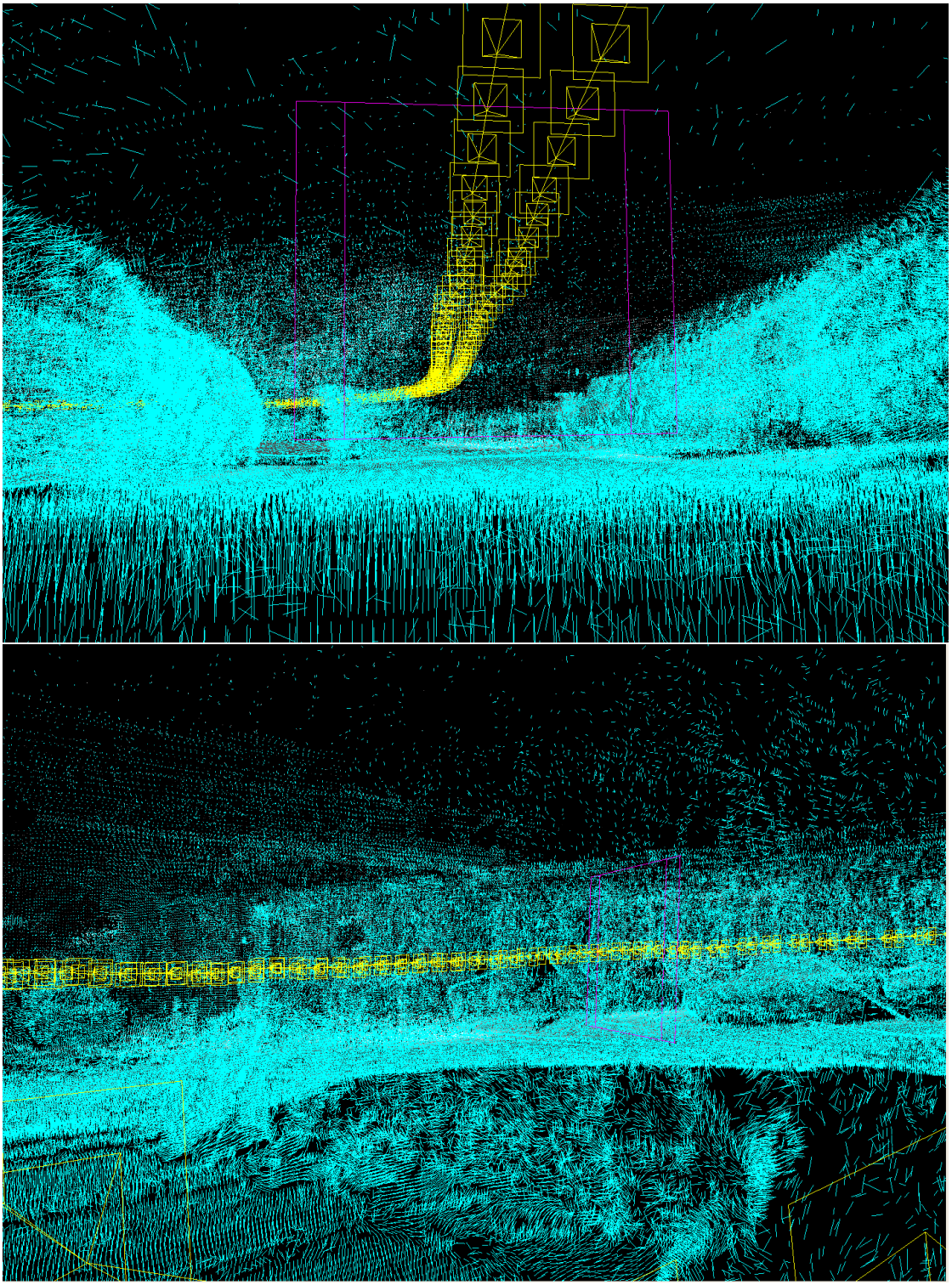


Figure 4.4: Normals of the ground plane and a car

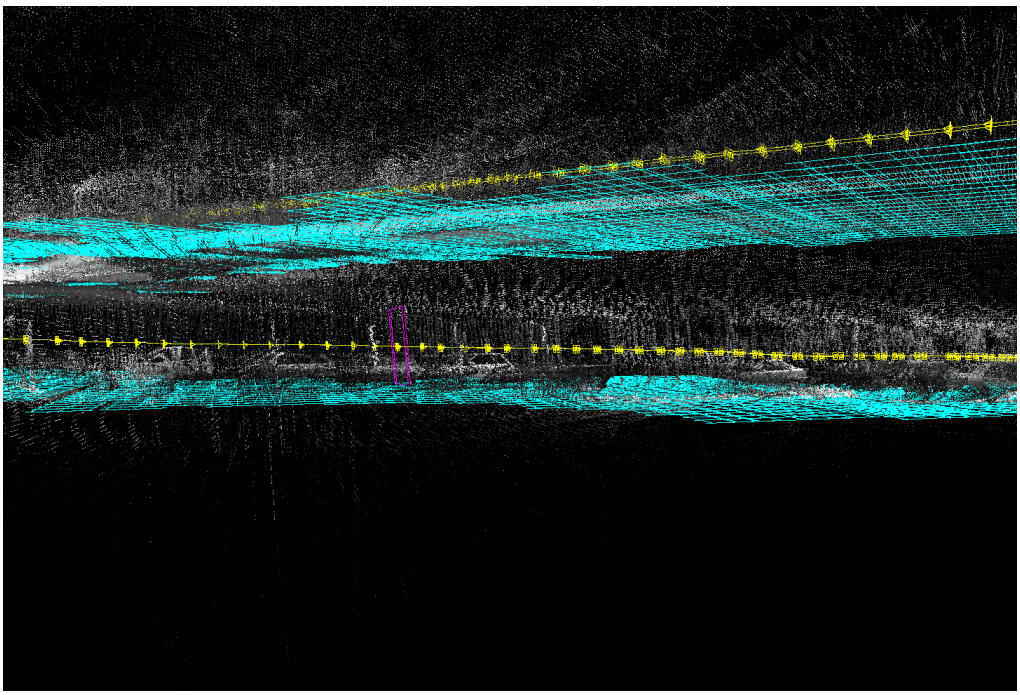


Figure 4.5: Ground planes estimated for a bridge sequence

Chapter 5

Object Detection

In the object detection phase the point cloud of the reconstruction is analyzed to find objects, sets of points that form together a single entity.

We are mostly interested in finding buildings that can be matched with the cadastral map, so we can assume that all searched objects are flat planes, since only the planar facades of the building can actually be seen. This reduced the problem of detecting objects to finding planes which are orthogonal to the ground plane.

Projecting all points on the ground plane, simplifies this problem again to detecting lines in the projected map, because the projection of an orthogonal plane is a single line.

All these lines can be found with a standard Hough transform, but a Hough transform combined with an MDL concept [BLK10] turned out to be more accurate.

Each of the detected lines is a hypothesis for a plane which has to be matched against the complete point cloud to find its 3-dimensional position and to classify it as building or car. This classification is necessary, because we only need buildings to match the scene against the cadastral map, but the line detection finds cars as well as buildings.

5.1 Line detection

Projecting the point cloud on the ground map creates a point density histogram, which can be seen as a gray-value image. To remove noise and make the image suitable for line detection, we apply a threshold to it, so that it becomes an edge-like binary map, where all non-zero bins are probably part of some line.

The standard Hough transform treats each point in this binary map as a vote for a set of lines in the voting space of all possible lines[Lei]:

A certain 2d line can be specified by the angle of the normal θ and the offset ρ of the line projected on the normal. Then each point $(x,y)^t$ on the line (θ,ρ) satisfies

$$\begin{pmatrix}\sin\theta\\ \cos\theta\end{pmatrix}^t \begin{pmatrix}x\\ y\end{pmatrix} = \rho.$$

On the other hand each line (θ,ρ) that contains the point $(x,y)^t$ has to satisfy the same condition.

Therefore, if θ has a discrete resolution, it is possible to find all possible lines through $(x,y)^t$ by enumerating all possible angles and calculating the respective offset:

$$\left\{\theta | \left(\theta, \begin{pmatrix}\sin\theta\\ \cos\theta\end{pmatrix}^t \begin{pmatrix}x\\ y\end{pmatrix}\right)\right\}.$$

In this way all possible lines for all points of the binary image can be calculated and those lines which have the largest number of supporting points are the true lines.

Since θ and ρ are discrete values with resolutions $\Delta\theta$, $\Delta\rho$, there only exist $\frac{\theta^{max}}{\Delta\theta}$ and $\frac{\rho^{max}}{\Delta\rho}$ possible values for θ and ρ. So the search of the lines with the largest support can be efficiently implemented with a voting matrix V of size $\frac{\theta^{max}}{\Delta\theta} \times \frac{\rho^{max}}{\Delta\rho}$ where the element $V(\theta,\rho)$ is incremented for each possible line (θ,ρ), as shown in Fig. 5.1. Then the element (θ,ρ) with maximal value $V(\theta,\rho)$ corresponds to the strongest line.

$\Delta\theta$ and $\Delta\rho$ are parameters that have to be chosen carefully. The smaller $\frac{\theta^{max}}{\Delta\theta}$, $\frac{\rho^{max}}{\Delta\rho}$ are, the more similar line hypotheses are merged together which is important to remove noise from the points. However, if they become too small, similar, but distinct lines are also fused together.

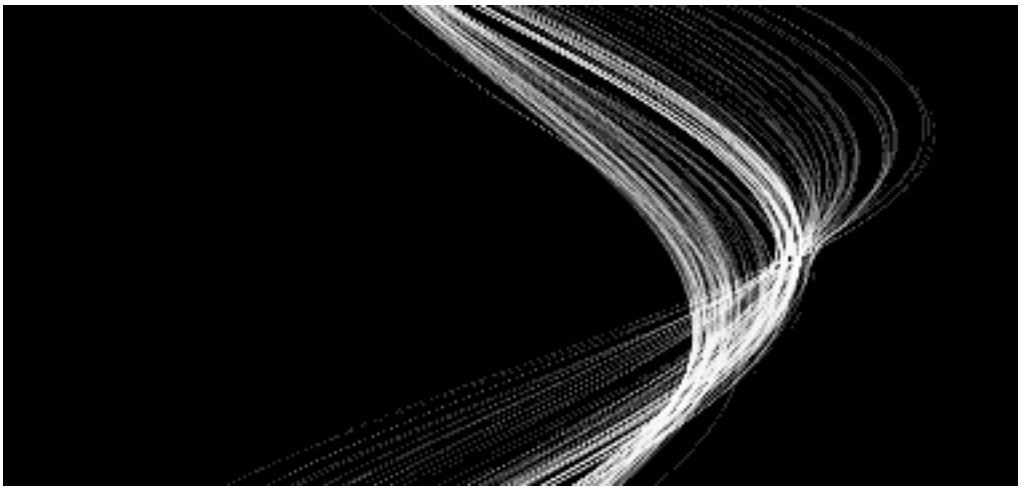


Figure 5.1: Hough voting space

As the maximal $V(\theta,\rho)$ corresponds to the strongest line, one could assume that the second maximal value is the second strongest line. But this is generally not true, because the second maximal value just consists of the votes of noisy points which should have voted for the maximal bin.

Thus in the Standard Hough Transform all votes in a small window around (θ, ρ) are removed from the voting matrix, similarly to the nearest neighbour suppression of the Harris corner detection. The window size is a parameter of the system and has to be carefully chosen, like the $\frac{\theta^{max}}{\Delta\theta}$ and $\frac{\rho^{max}}{\Delta\rho}$ parameters. Figure 5.2 shows the maximal bins chosen by the Standard Hough Transform.

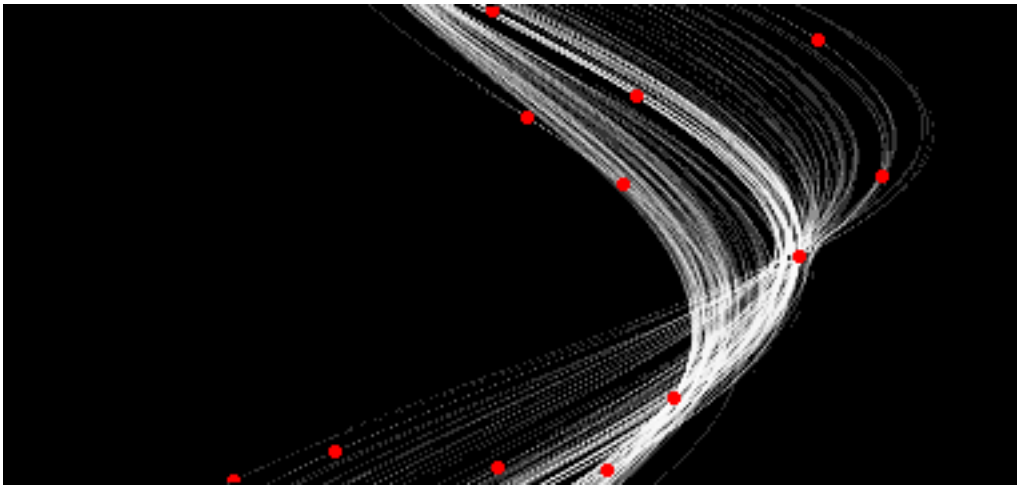


Figure 5.2: Hough voting space with peaks chosen by the Standard Hough Transform

A more reliable Hough Transform that was described in [BLK10] and provided as library is based on the idea to explain all points with a line and keep track of the assignments between points and lines, so that each vote for a new line also becomes a vote against the line currently assigned to this point:
The voting is changed, such that a point does not only vote for lines leading exactly through the point, but also for lines which just miss the point. The votes can be considered as the log likelihood that the point $(x,y)^t$ belongs to the line (θ, ρ) and is based on a score that penalizes more distant lines[1]:

$$\log P_{x,y}(\theta, \rho | I) = -c \cdot \left| \begin{pmatrix} \sin\theta \\ \cos\theta \end{pmatrix}^t \begin{pmatrix} x \\ y \end{pmatrix} - \rho \right|^e,$$

where c is a weighting factor and e an distance exponent, usually 1 or 2.
Then after the strongest line has been found by choosing the maximal bin of the voting matrix, like in the Standard Hough Transform, all points are assigned to this line and a new voting matrix is created. Thereby the vote cast for a certain new line (θ, ρ) is the score gained by assigning the point $(x,y)^t$ to this new line, minus the score lost by removing the previous assignment to the line $(\theta^{prev}, \rho^{prev})$:

[1] They also suggest to include the angular distance between the line and the tangent of the point, but the ground plane histogram I is too noisy to calculate the derivatives.

$$V_{x,y}^{new}(\theta,\rho) = \max(\log P_{x,y}(\theta,\rho|I) - \log P_{x,y}(\theta^{prev},\rho^{prev}|I), 0)$$

This finds a new strongest line $(\theta^{new},\rho^{new})$, and the points that are more explained by this line than the previous one – i.e. that satisfy

$$\log P_{x,y}(\theta^{new},\rho^{new}|I) > \log P_{x,y}(\theta^{prev},\rho^{prev}|I)$$

– are assigned to it. This is repeated until a maximal count of lines has been found, or the maximal score increase of a new line is less than a threshold λ.
They show that this corresponds to an approximation of a minimal descriptor length (MDL) line detector which finds the lines that minimize the bit count of a description of the image, when -$\log P_{x,y}(\theta,\rho|I)$ bits are required to describe a point assigned to a line and λ bits to describe a unassigned point.

Any Hough transform can only find infinite long lines in the binary image, but to actually localize the objects it is necessary to find line segments. Directly voting for line segments is theoretical possible, but it would require a 4-dimensional voting space, which is not feasible. Therefore the lines of the Hough transform have to be postprocessed and split into line segments.

All points of the binary image that are near to a line are projected upon this line creating a a 1-dimensional histogram on the line. Then a threshold t is applied to the histogram h which reduces noise and converts the histogram to a binary histogram. All non-empty bins in the histogram that have a distance less than d to each other are then connected to close small holes in the line caused by missing points or the thresholding. Formally the filtering of the histogram h to a new histogram h' can be written as:

$$h'(j) = \begin{cases} 1 & \exists i,k : i \leq j \leq k \wedge k - i \leq d \wedge h(i) \geq t \wedge h(k) \geq t \\ 0 & else \end{cases}$$

In this new histogram line segments can be easily found by splitting the histogram at empty bins. To reduce the noise further, all line segments that are shorter than a minimal length l or consist mostly of non-real points, i.e. of bins with $h(j) < t$, are discarded.

This whole postprocessing runs in linear time relative to the line length, and the accuracy depends highly on the correct choice of the various parameters.

Since we are mostly interested in finding buildings and cars, which can not intersect with each other unless a crash occurred, we can optionally apply another check to reduce the number of wrong line segment hypotheses: All line segments are checked for intersections and if two line segments intersect, the shorter line segment is removed.

Figure 5.3 and Fig. 5.4 show the final line segments detected with the Standard and MDL Hough transform.

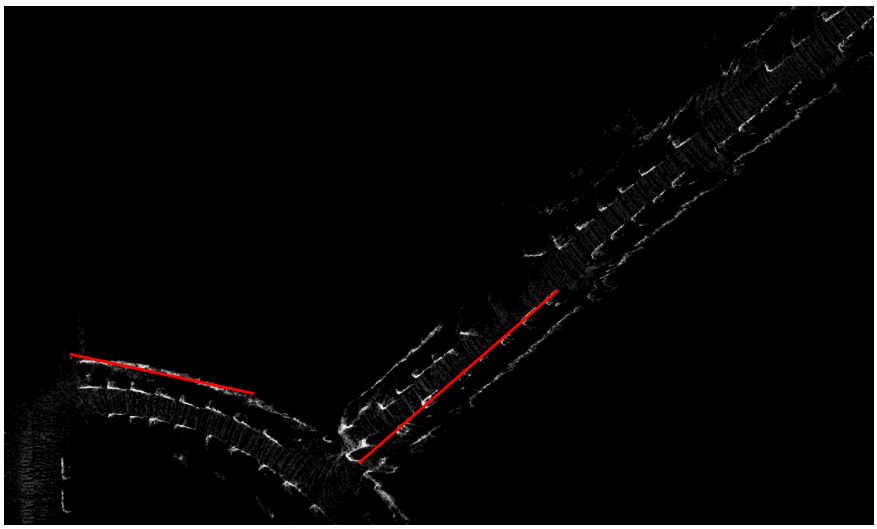


Figure 5.3: Line segments found with the Standard Hough Transform in a difficult map

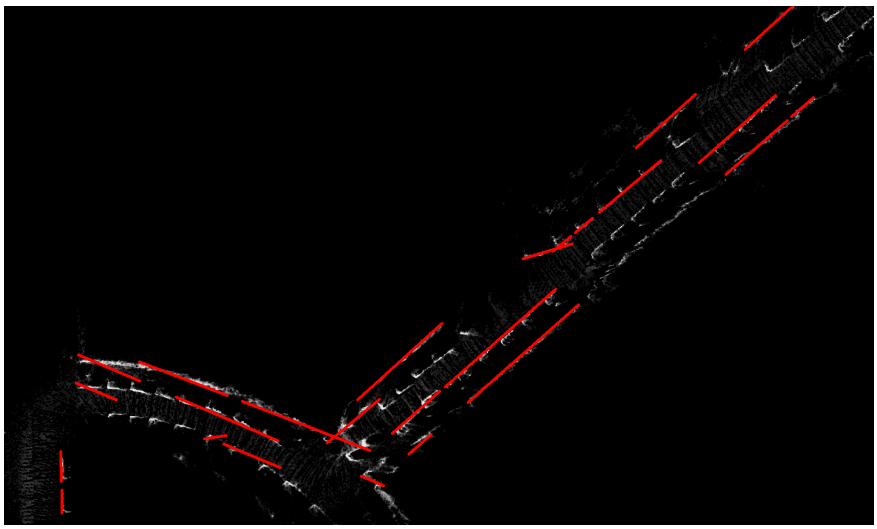


Figure 5.4: Line segments found with the MDL Hough Transform in that difficult map

5.2 Plane detection

In order to detect planar objects the line segments of the previous step are used to construct a plane part, and the distributions of the points in a bounding box around this plane are used to classify these planes as building or car objects by comparing the distribution with two reference distributions.

Each line segment is described by its start and end position $(s,t) \in (\mathbb{R}^3)^2$ on the ground plane, and can be extended to a plane part by keeping (s,t) as the bottom edge of the plane part and using a copy of the line segment moved along the normal $n(gp)$ of the ground plane as its top edge.

Such a rectangular part of a plane can be described by its center $o_c \in \mathbb{R}$ and edge vectors $o_x = \frac{t-s}{2}, o_y$, so that it has the four corners $o_c \pm o_x \pm o_y$ and the length $2|o_x|$ is the length $|t-s|$ of the line segment. o_y has to be parallel to the ground plane and the length $|o_y|$ will become the height of the object.
We also add the normal $o_z = \frac{o_x \times o_z}{2|o_x \times o_z|}$ of the plane to the description which results in a bounding box around the plane part with corners $o_c \pm o_x \pm o_y \pm o_z$. The length of this box along o_z is exactly one metre and we assume that only points within this bounding box contribute to this object.

All these point in the bounding box are then projected on the object plane to create a 2-dimensional histogram that describes the point distribution within the box. We then project this histogram on its y-axis by summing each row, which results in a 1-dimensional histogram h as the distribution of height distributions of the points, i.e. point density in each height of the object.
This projection can be optimized by using the octtree generated during the reconstruction and skipping all points in nodes which do not intersect the bounding box.

From this distribution h we can extract the height of the object, and its class by comparing it with reference distributions for buildings and cars. A detected building has to match the building distribution, a car the car distribution and miss-detected planes will match neither.

There exist various functions to compare two distributions u and v, we use the Bhattacharyya score and the Chi^2 score, which are defined as[Lei]:

$$bhattacharyya(u,v) = \sum_i \sqrt{u(i)v(i)}$$

$$\chi^2(u,v) = \sum_i \begin{cases} \frac{(u(i)-v(i))^2}{v(i)} & v(i) \neq 0 \\ 0 & else \end{cases}$$

The Bhattacharyya score is excellent to estimate the general similarity between two different distributions, while the χ^2 score computes the similarity between distribution that are assumed to stem from the same data and can be interpreted as the probability that two distribution are identical. Thence we can use the Bhattacharyya score to classify the object as car or building, and the χ^2 score to check if this classification is actually correct.

However, there usually exist an offset between the true and measured distribution, because the objects are not directly on the ground plane, e.g. if they are located on a hill or a bridge, and we have to remove this offset by shifting the distributions until the score is maximized. Thus we define a movement operator $\rightarrow$ that shifts all elements of a distribution vector, as follows:

$$(v \rightarrow \Delta)(x) = v(x - \Delta)$$

The search for the offset and the best matching class can then be written as:

$$(\Delta, ref) \leftarrow \underset{\Delta, ref \in CLASSES}{\operatorname{argmax}} \; bhattacharyya(h, ref \rightarrow \Delta),$$

where $CLASSES$ is the 2-element set of all reference distributions.
Afterwards the χ^2 test of the match is performed by checking, if $\chi^2(h, ref \rightarrow \Delta)$ is above a certain threshold. If this test fails, the plane is discarded as invalid.

To create the reference distributions we assume that the points of buildings and cars are evenly distributed, and only differ in the height of the object. An evenly point distribution of height H is given by

$$dist_H(i) = \frac{1}{H} \begin{cases} 1 & 1 \leq i \leq H \\ 0 & else \end{cases}.$$

So, if we call the expected height of a building $B \approxeq 8m$ and the height of a car $C \approxeq 1.5m$, we get $CLASSES = \{dist_B, dist_C\}$.

It is also possible to learn these distributions by (manually) classifying the detected planes and averaging all distributions for each class. To normalize the offset of each distribution h, they are shifted to zero mean, i.e. $0 =^! \sum_i i \cdot h(i - \Delta) = \sum_i (i + \Delta) \cdot h(i) = \sum_i i \cdot h(i) + \Delta \sum_i h(i) = \sum_i i \cdot h(i) + \Delta$.
The resulting distributions look similar to Gaussian distributions of different variance, as you can see in Fig. 5.6, but perform worse than the artificially chosen.

We can also use the distribution score to improve the MDL based line detection. Remember that $V(\theta, \rho)$ was the sum of all log likelihoods voting for a line (θ, ρ), so $\exp V(\theta, \rho)$ is the probability that a certain line existing in the map.
For each object resulting from a line we also have the Chi^2 score $\chi^2(h, ref_{h_T} \rightarrow \Delta)$, which lies in the interval $(0, 1)$, since ref_{h_T} and h are probability distributions, and can be considered as the probability that this object is valid.
Therefore $\exp V(\theta, \rho) \chi^2(h, ref_{h_T} \rightarrow \Delta)$ is the combined probability that a line exists and creates a valid object, and we can modify the voting matrix by adding $\log \chi^2(h, ref_{h_T} \rightarrow \Delta)$, or a linear approximation $f \cdot \chi^2(h, ref_{h_T} \rightarrow \Delta)$, to each bin which is a candidate for a new line.
This should remove votes for useless lines and allow the points to vote for other lines which results in more detected objects.

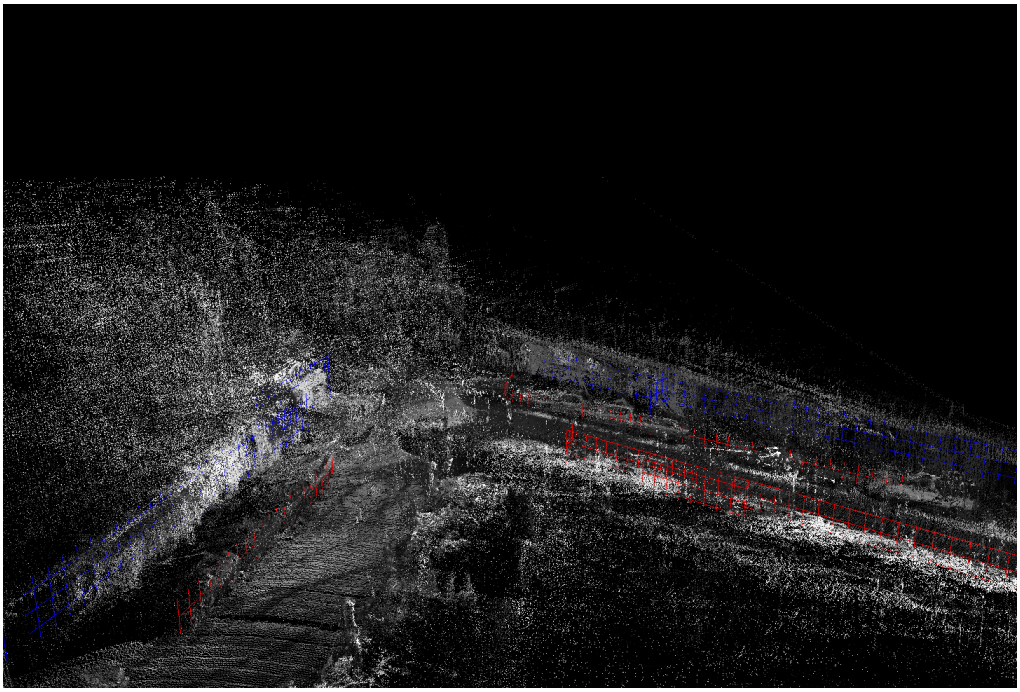


Figure 5.5: Detected cars and buildings

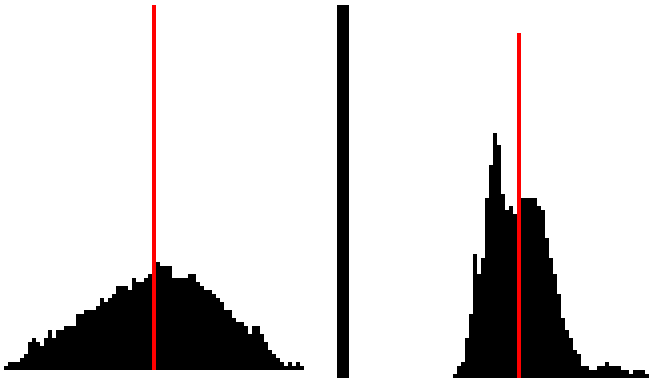


Figure 5.6: Learned distributions for buildings (left) and cars (right)

5.3 MDL framework

In the object detection MDL framework each voxel of the point cloud is explained by an object or the background, like the voting bins by the detected lines of the MDL line detector described in Section 5.1.

We follow the approach of [LLS08], which defines the savings of an accepted hypothesis h, as being proportional to

$$S_h \sim S_{area} - k_1 S_{model} - k_2 S_{error},$$

where S_{area} is the area – in our case the volume – covered by the hypothesis h, S_{model} the cost of the hypothesis and S_{error} the cost to correct the errors made by the hypothesis. k_1 and k_2 are system parameters that specify respectively the cost of adding a new hypothesis to the model and the required weight of supporting points for a hypothesis .

The savings of two[2], overlapping hypothesis becomes then the sum of the saving of both hypothesis minus the savings lost by the interaction of both:

$$S_{k,h} \sim S_k + S_h - S_{area}(k \cap h) + S_{error}(k \cap h)$$

Using this definitions one can calculate the total saving of a set m of chosen hypotheses by summing the savings of each hypothesis $h \in m$ and the savings of each pair $(h,k) \in m$. If we then consider m as a vector of indicator variables, such that $m(h) = 0$, if $h \notin m$, and $m(h) = 1$, if $h \in m$, we can write these summations as a vector-matrix product $m^t Qm$ with some matrix Q. Then the problem is reduced to finding the m that maximizes $m^t Qm$, by solving a so called quadratic "Boolean optimization problem".

[LLS08] gives a definition of the matrix Q based on defining S_{error} as negative likelihood of all object pixels, however their equations are not directly applicable to our problem, because their system only deals with object detection in 2D images. But we can generalize it to 3D by replacing the expected area of an object by its expected volume $V_e(h)$, and their probability $p(\mathbf{p} = fig.|h)$ that a pixel $\mathbf{p}$ belongs to figure fig given the hypothesis h through the probability $P(p|h)$ that a point p belongs to the object given the hypothesis h.

After this modifications the entries of Q are given by:

$$Q(h,h) = -k_1 + \frac{1}{V_e(h)} \sum_{p \in h} ((1-k_2) + k_2 P(p|h))$$

$$Q(k,h) = Q(h,k) = \frac{1}{2V_e(h)} \sum_{p \in h \cap k} (-(1-k_2) - k_2 P(p|w(h,k)))$$

Like before $Q(h,h) = S_h$ is the saving of a single hypothesis and $Q(k,h) = S_{k,h}$ the saving of a pair of hypotheses. The sums $\sum_{p \in h}$ and $\sum_{p \in h \cap k}$ are respectively taken over

[2]They do not handle any higher order overlaps.

all points in the bounding box of the hypothesis h or the intersection of the boxes of h and k.

Finding the globally maximal $\operatorname{argmax}_m m^t Q m$ requires exponential time, so they suggest to use a greedy search that repeatedly adds the new local maximal hypothesis, until no local improvements are possible.

If we define e_h as the set-vector that only contains the hypothesis h:

$$e_h(k) = \begin{cases} 1 & k = h \\ 0 & else \end{cases}$$

we can write the greedy search step that adds a new maximal hypothesis h to a set-vector m, as follows:

$$m' \leftarrow \operatorname*{argmax}_{h:m(h)=0} (m + e_h)^t Q (m + e_h)$$

This is repeated until no m' that increases the savings can be found anymore. By caching the calculations of previous steps, $(m \cup \{h\})^t Q (m \cup \{h\})$ can be calculated in amortized constant time.

Before we can actually calculate the matrix Q, we still have to define the expected volume $V_e(h)$ of a hypothesis h, the probability $P(p|h)$ that a point p belongs to an object given the hypothesis h and what these hypotheses h even are.

We could consider only the objects detected by the algorithms of the previous section as hypotheses, however in the MDL framework we can also include ground plane hypotheses that explain the points on the ground plane and prevent them from being assigned to object hypotheses.

To generate these ground plane hypotheses, we basically take the local ground planes of Section 4.4 and surround each with a bounding box similarly to the boxes around the plane objects of the previous section. The normal h_z of the local ground plane is parallel to the normal $n(gp)$ of the global ground plane.

We also have to add a property $h_T \in \{B, C, G\}$ to each hypothesis h that describes its assumed type as building, car or ground plane..

To calculate then the expected volume $V_e(h)$ of a hypothesis h we assume that all objects have a fixed length and height depending on their type, but leave the size along the normal variable, because it is mostly a system parameter.

$$V_e(h) = 2|h_z| \begin{cases} 10^2 & for\ buildings,\ h_T = B, \\ 3^2 & for\ cars,\ h_T = C, \\ 10^2 & for\ ground\ planes,\ h_T = G. \end{cases}$$

Keeping the size constant favors larger objects which can explain more points.

We assume that the surface of the objects is mostly flat, so the probability that a point belongs to an object has to decreases very fast with increasing distance. This can be formulated as a falling exponential function, as suggested in [SPF10], which also maps the distance to a probability-like value in the interval $[0,1]$:

$$P(p|h) = -\frac{1}{f_{h_T}} \exp(-f_{h_T}|p-h_c|).$$

f_{h_T} is a parameter that describes how fast the probability declines with increasing distance, and depends on the assumed class of the hypothesis, i.e. it has to be larger for more flat objects like buildings and smaller for cars. The multiplication with $\frac{1}{f_{h_T}}$ normalizes the integral of $\int \frac{1}{f_{h_T}} \exp(-f_{h_T}x)\,\mathrm{d}x$ to 1, so that it becomes a valid probability density.

It is also possible to include the normal $n(p)$ of the point p in the score. When examining the 3D model one sees that the normals of the points on the ground plane are almost all parallel to the plane normal $n(gp)$, but the normals of the points on the objects deviate heavily. We therefore only use the normals to distinguish between ground plane points and objects points:

$$P(p|h) = -\frac{1}{f_{h_T}} \exp(-f_{h_T}|p-h_c|) \begin{cases} \langle n(p), n(gp)\rangle^2 & for\ ground\ planes,\ h_T = G \\ (1-\langle n(p), n(gp)\rangle^2) & else \end{cases}$$

$\langle n(p), n(gp)\rangle$ is the cosine between these two normals and squaring it increases the weight of the normal condition and ensures that the value is within the interval $[0,1]$.

As described so far the MDL framework can only accept or reject the hypotheses generated from the line segments and ground planes, but most of them are independent and a MDL framework becomes most useful, when different, conflicting hypotheses exist. Therefore we split each single hypothesis into several overlapping hypotheses which are then all subjected to the MDL framework:
A building hypothesis of length $L = 2|h_x|$ is split into parts of length $2^{l+2} \leq L$ and shifted to all possible non-overlapping positions for a certain length. If we write these splits as tupels from a starting to an ending offset, which denotes the complete unmodified hypothesis as $(0,L)$, the set of added hypotheses becomes:

$$\{(2^{l+2}i, 2^{l+2}\cdot(i+1)) | 2^{l+2} \leq L \wedge 2^{l+2}i < L\}$$

A car hypothesis is only split in parts of 4 metres each, because longer cars are too rare to be worth consideration:

$$\{(4i, 4(i+1)) | 4i < L\}$$

A ground plane hypothesis is not split at all, because it can already be considered to be a splitted part of a global ground plane.

To split a hypothesis to a tupel (s_i, s_{i+1}) the lengths along h_y and h_z and the center position h_c along the h_y, h_z axis has to be kept constant, while the length along the h_x axis becomes $s_i - s_{i-1}$ and the center along h_x the center of the interval $[s_i, s_{i-1}]$:

$$o_x^i = \frac{h_x}{|h_x|}\frac{s_i - s_{i-1}}{2} \quad \text{and} \quad o_c^i = h_c - \frac{h_x}{|h_x|}\left\langle \frac{h_x}{h_x}, h_c \right\rangle + \frac{h_x}{|h_x|}\frac{s_i + s_{i-1}}{2}$$

Splitting the hypotheses even more could be beneficial for the accuracy of the result, however it would slow down the system due to the $O(n^2)$ running time needed to create the matrix Q and choosing the best vector m.

Chapter 6

Matching

In the final matching phase we combine the visual odometry, the reconstruction and the detected objects with the cadastral map.

First we find the camera position in the map, either offline, estimating the position from a given path, or online, tracking the camera starting at a given initial position and correcting the path of the odometry. This will localize the camera globally using just local data and the map, and could replace traditional loop-closing algorithms or even GPS tracking systems.

Once the geodetic position of the camera is known, we combine the semantic information of the map with the data of the odometry and reconstruction.

We can create a 3D model of the map and project all buildings in the image sequence, to mark stores and other points of interest. The images can then be projected the other way around on the facades of the buildings to create a texturized 3D model.

The matching procedure also helps to improve the object detection in the reconstruction by finding matches between the detected buildings and the true buildings of the cadastral map. Each building can either be found in the OpenStreetMap and annotated with semantic informations, or it can be added as new building to the map, which leads to a novel, unique way of creating cadastral maps.

6.1 Offline Positioning

In the offline setting the cadastral map and the complete estimated path of the visual odometry are given and we find the location of this path in the cadastral map. This is useful to localize a "mislocalized"/kidnapped robot or to replace traditional loop-closing algorithms, and is a necessary step for the later matching algorithms. It can also be used as an alternative to GPS tracking, e.g. if the robot navigates in caves without GPS reception or if the US has disabled GPS due to a world war III.

To localize the path we employee shape matching techniques that compare the shape of a set of streets to the shape of the path and return the streets with the most similar shape.

There are various possible scores for judging the similarity of a certain path/map alignment, based on the distance $D(p) = |p - n(p)|_1$ between the position of one frame p to the nearest street point $n(p)$:

$$SCORE_{OUTLIER}(p_i) = -|\{i|D(p_i) < t\}|,$$

$$SCORE_{CHAMFER_1}(p_i) = -\sum_{i=1}^{n} D(p_i),$$

$$SCORE_{CHAMFER_2}(p_i) = -\sum_{i=1}^{n} \max(D(p_i), t)$$

$$SCORE_{STREET}(p_i) = -\sum_{i=1}^{n} \left|p_i - p_i'\right|_2,$$

where p_i' is defined as

$$p_i' = \lim_{k\to\infty} p_{i,k}'',$$

with

$$p_{i,0}'' = p_{i-1}', \quad p_{i,k+1}'' = \begin{cases} p_{i,k}'' + v_{i,k} & D(p_{i,k}'' + v_{i,k}) < t, \\ p_{i,k}'' + (v_{i,k,x}, 0)^t & D(p_{i,k}'' + (v_{i,k,x}, 0)^t) < t, \\ p_{i,k}'' + (0, v_{i,k,y})^t & D(p_{i,k}'' + (0, v_{i,k,y})^t) < t, \\ p_{i,k}'' & else, \end{cases}$$

where $v_{i,k} = \frac{n(p_i) - p_{i,k}''}{|n(p_i) - p_{i,k}''|}$ is the normalized direction from the point $p_{i,k}''$ to the nearest street point $n(p_i)$.

$SCORE_{OUTLIER}$ counts the number of frame positions that are more than d units away from a street, i.e. the number of frames that are not on the street. $SCORE_{CHAMFER_1}$ and $SCORE_{CHAMFER_2}$ are Chamfer scores that sum all these distance, whereby the summed distance values of $SCORE_{CHAMFER_2}$ are limited to a maximal distance t.

$SCORE_{STREET}$ models the interaction of the path and the street by creating a new path p_i' that follows the true path, but is constrained to stay on the street. Basically $p_{i,k}''$ tracks a point moving from the position of the previous frame p_{i-1}' to the nearest street point $n(p_i)$. In each frame $p_{i,k}''$ moves greedy into the direction of $n(p_i)$, until it reaches the boundaries of the street.

Therefore this score not only compares the shape of the path with the street, but also ensures that the streets of the examined map part are connected, and that it is actually possible to move on this street along the given path.

In all scores it is possible to either use the Manhattan distance $|x - x'| + |y - y'|$ or the Euclidean distance $\sqrt{(x-x')^2 + (y-y')^2}$. The Euclidean is more accurate, but the Manhattan distance is faster too calculate and it is simple to create a distance map as a matrix D that contains at each entry $D(x,y)$ the Manhattan distance from (x,y) to the nearest street: After a coordinate system and resolution are chosen that transform

all coordinates of the map into integer values, an initial bitmap like Fig. 6.1 can be produced by drawing the streets as lines on D, creating:

$$D(x,y) = \begin{cases} 0 & (x,y) \text{ on street} \\ \infty & else \end{cases}$$

This bitmap can then be transformed into a distance matrix – as shown in Fig. 6.2 – by

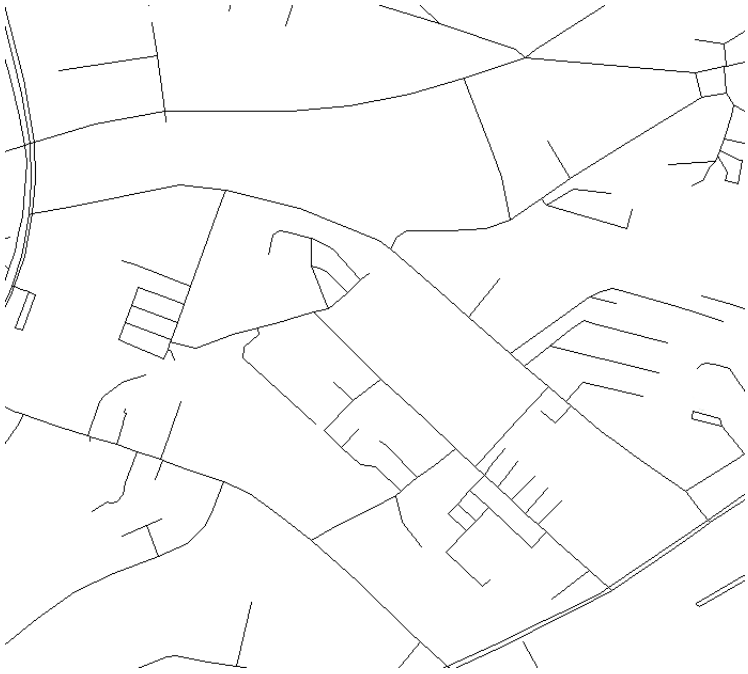


Figure 6.1: Simplified cadastral street bitmap (around UMIC)

successively taking the incremented distance from the entry above/left of each entry and afterwards from the below/right[Lei]:

$$D(x,y) \leftarrow \min(D(x,y), D(x-1,y)+1, D(x,y-1)+1) \quad \text{from top-left to bottom-right}$$

$$D(x,y) \leftarrow \min(D(x,y), D(x+1,y)+1, D(x,y+1)+1) \quad \text{from bottom-right to top-left}$$

This algorithm can be extended by collating not only the distance, but also the position of the nearest street to create a matrix/tensor D^e where each element is the position of the nearest street. D^e can be used to approximate a Euclidean distance map, or to create an angle map α that stores the angle of the nearest street.

The angle between the street and the direction of the path can be included in the score by replacing the distance $|p_i - n(p_i)|_1$ with $|p_i - n(p_i)|_1 + w|\alpha_i - \alpha(p_i)|_\circ$ where w is a weighting factor and $\alpha, \alpha(p_i)$ the angle of the normal of the path or the street. The angular distance function $|\beta|_\circ = \min(|\beta| \mod \pi, \pi - |\beta| \mod \pi)$ returns the minimal

Figure 6.2: Distance map showing the distance to the nearest street (around UMIC)

angular difference of each quadrant. However, including the angle does not really increase the accuracy, probably because a turning car does not actually move in the direction of the streets.

Once a scoring function *SCORE* has been chosen, the best alignment of the path with the map can be found by enumerating all possible transformed paths as follows:

$$(x, y, \theta) = \underset{(x,y,\theta)}{\operatorname{argmax}} SCORE(R_\theta p_i + (x, y)),$$

where (x, y) is a translation of the path and R_θ a rotation matrix that rotates the path around the angle θ.

Enumerating all possible translations and rotations is of course very slow, so we apply two optimizations: First, we choose a path position p_s and only check for translations, that move p_s exactly on a street and rotations around p_s:

$$(x, y, \theta) = \underset{(x,y,\theta) \wedge D(x+p_{s,x}, y+p_{s,y})=0}{\operatorname{argmax}} SCORE(R_\theta(p_i - p_s) + p_s + (x, y))$$

Second, we add the constraint that the path and the street have to be parallel at the position of p_s, because it makes no sense to check for path transformations that do not align the path with the street. If a_s is the angle of the tangent of the path at p_s and

$\alpha(x,y)$ of the street at position (x,y), the search equation finally becomes:

$$(x,y,\theta) = \underset{(x,y,\theta)\wedge D(x+p_{s,x},y+p_{s,y})=0\wedge\theta=\alpha(x+p_{s,x},y+p_{s,y})-\alpha_s+(\pi\pm\pi)/2}{\operatorname{argmax}} SCORE(R_\theta(p_i-p_s)+p_s+(x,y))$$

When choosing p_s it is important to chose a frame where the camera does not rotate, so that the local orientation corresponds to the orientation of the street. Thus we chose the most stable point p_s as follows:

First the absolute angle α_i of all frames of the path is calculated:

$$\tan\alpha_i = \frac{p_{i+1,x}-p_{i,x}}{p_{i+1,y}-p_{i,y}}$$

Then the most stable point is the position of the frame which has the smallest rotational change:

$$s \leftarrow \underset{i}{\operatorname{argmin}} |\alpha_{i+5}-\alpha_{i-5}|$$

A possible extension might be to include the buildings found in the object detection in the score to increase the matching accuracy, but in our system the object detection is not yet reliable enough for this and the cadastral maps of our data sets does not always contain all buildings.

6.2 Online Positioning

In the online positioning setting the camera position is tracked in the cadastral map while the visual odometry is performed. Such an online tracking can detect and corrects errors in the visual odometry, e.g. if the vehicle leaves the street or runs into buildings. On large sequences it should also prevent the accumulation of small errors and present an alternative to usual loop-closing algorithms. Finally it can also create nice looking visualizations of the path.

We assume that the starting position and angle is exactly known, either from an initial oracle or from an offline positioning applied to previous frames. Then we simulate a camera moving on the streets following the odometry, but constrained to stay always on the street, similar to the calculation of the $SCORE_{STREET}$ of the previous section.

The position correction algorithm is interposed with the visual odometry algorithm: After a new camera hypothesis has been chosen, it is corrected with the cadastral map and the corrected position is used as final estimate for the current frame and passed as new observation to the Kalman filter.

Formally we are given the previous camera position p_{i-1} and a new hypothesis p_i and have to find a corrected position p_i' by first updating the simulated camera to move in the direction of the nearest street point of p_i and then correcting the position accordingly. For the first step we could proceed exactly like in the $SCORE_{STREET}$ calculation of the previous section, but this requires the creation of a distance map for

all possible positions which can be quite slow[1], so we use a different algorithm that works directly with the vector-graph representation of the cadastral map.

The cadastral map can be seen as a set of street segments $\{(s,t)\}$ each connecting points s and t.

To find the nearest point on a such a street segment we just project the camera position p on the street segment:

$$n_{s,t}(p) = \begin{cases} s & \langle t-s, p-s\rangle < 0 \\ t & \langle t-s, p-s\rangle > |t-s|^2 \\ \frac{1}{|t-s|^2}(t-s)\langle t-s, p-s\rangle + s & else \end{cases}$$

In the first two cases the projected point is not actually within the line segment and the corresponding line end node is returned, in the last case we return the projection itself.

In the very first frame we can find the nearest street point $n(p)$ of the complete map by projecting the point p on all street segments and choosing the street segment with the smallest distance:

$$(s_0,t_0) \leftarrow \operatorname*{argmin}_{(s,t)} |n_{s,t}(p) - p|$$

In later frames we track the simulated camera over the streets. We assume that it can move freely on a street segment, but can only leave it and jump to a new street segment, if the distance to the new position is less than ε. This corresponds to the constraints of a real camera moving on the streets, and also allows estimation errors up to ε.

If $|n_{s_{i-1},t_{i-1}}(p_i) - s_{i-1}| > \varepsilon$ and $|n_{s_{i-1},t_{i-1}}(p_i) - t_{i-1}| > \varepsilon$ the simulated camera is far from the end nodes of the current street segment and is therefore constrained to stay on this segment, so we set $(s_i,t_i) \leftarrow (s_{i-1},t_{i-1})$.

Otherwise we check all streets N adjacent to the nearest edge node s or t if they are nearer to p_i than the current street, under the constraint that the simulated camera may not move more than ε from the current position at $n_{s_{i-1},t_{i-1}}(p_i)$:

$$(s_i,t_i) \leftarrow \operatorname*{argmin}_{(s,t)\in N \wedge |n_{s,t}(p_i) - n_{s_{i-1},t_{i-1}}(p_i)| \le \varepsilon} |n_{s,t}(p_i) - p_i|$$

This street segment (s_i,t_i) is the new nearest globally nearest street segment[2] and the point $n_{s_i,t_i}(p_i)$ is the new globally nearest street point.

$n_{s_i,t_i}(p_i)$ lies exactly on the center of the street, although a real street has a finite width w. Therefore we do not use $n_{s_i,t_i}(p_i)$ directly as the nearest point, but $n'_{s_i,t_i}(p_i)$

[1] In the last section, the distance map was actually an optimization because we were testing almost all possible positions anyway, but here we only consider a single position.

[2] In an actual implementation we have to repeat this until the street does not change anymore, because there may exist street segments shorter than ε.

which is allowed to deviate up to w units from the center in the direction of p_i, as follows:

$$n'_{s_i,t_i}(p_i) = \begin{cases} p_i & |n_{s_i,t_i}(p_i) - p_i| < w \\ p_i + (n_{s_i,t_i}(p_i) - p_i)\frac{|n_{s_i,t_i}(p_i)-p_i|-w}{|n_{s_i,t_i}(p_i)-p_i|} & else \end{cases}$$

This $n'_{s_i,t_i}(p_i)$ can then be used to correct the camera position p_i estimated by the odometry to match more closely the cadastral map. We could set $p'_i = n'_{s_i,t_i}(p_i)$ to force the camera directly on the street, however sudden, large changes can confuse the odometry.

Thus we use an affine combination of the estimated and corrected position which drags the camera slowly towards the street: ,

$$p'_i = (1-f)p_i + f n'_{s_i,t_i}(p_i),$$

where f is another system parameter, defining the weight of the correction.

It also turns out that a turning camera often leaves the street altogether and the odometry reacts very sensible to positional changes during a turning phase. Therefore we compare the angles of the average of the last forward vectors and do not correct the position if the angles have changed more than a certain threshold.

The time necessary to perform these corrections is negligible compared to the time of the odometry, so if the visual odometry runs in real time, the visual odometry with correction will also run in real time.

An possible extension to the online positioning is to prevent the camera from moving into buildings. Then we would model the buildings as force emitters that apply a standard inverse square force on the camera, like a two equally charged sphere on each other. This results in an acceleration

$$a = -\frac{1}{m}\frac{1}{|r|^2}\frac{r}{|r|}$$

of the camera, where m is the mass of the camera, i.e. a parameter that controls the influence of a building on the camera, and r the vector from the camera to the building.

To include this acceleration in the Kalman filter, we have to use a constant-acceleration model with a 9-dimensional hidden and observable state of (position, velocity, acceleration). The position is observed as before, the velocity can be estimated from the distance between two estimated positions, and the acceleration is the sum of the accelerations vectors of all buildings. One could also remove the velocity from the observable state. The update matrix just increase the velocity by the current acceleration and the position by the current velocity.

6.3 Semantics

After the path has been aligned with the cadastral map, we can enhance our understanding of the scene with semantic information the sequence alone can not provide.

We begin by creating a 3D model of the cadastral map. The coordinates of any OpenStreetMap object are given in geodetic coordinates, latitude and longitude, and can be converted to a local world coordinate system as described in Appendix B.2.

Streets are given as a sequence of connected coordinates and can easily be represented by drawing OpenGL lines from each coordinate to the next one. Buildings are given in a similar way as a sequence of building corner coordinates and could also be represented as a line. However, we chose a more intuitive approach and visualize the buildings as solid objects. Thus each building line is replaces by a OpenGL quad that has the line as bottom edge and the line translated parallel along the y-axis as top edge. The normals of the quad are chosen to point away from the building center, which is the mean of all building corners. On the top and bottom of the building we place polygons of all corners to close it.

An OpenStreetMap generally does not contain building heights or altitudes, so we assume that all buildings start at ground level and have a height of 15 metres. Nevertheless we could use this information, if we have other maps that provides it.

OpenStreetMap also supports real semantic information, like the positions of bakeries or pharmacies. However, these information are not linked to the respective buildings, and just say that at a certain position an object with certain semantic properties can be found. To find the relationship with the buildings, we describe the buildings as polygons on the ground level and search for each building all contained semantic points. Each match is then added as an annotation to the building and visualized by changing the texture of the building to a common symbol corresponding to the semantic information. All remaining semantic points outside of buildings are rendered as floating globes, shown in Fig. 6.4.

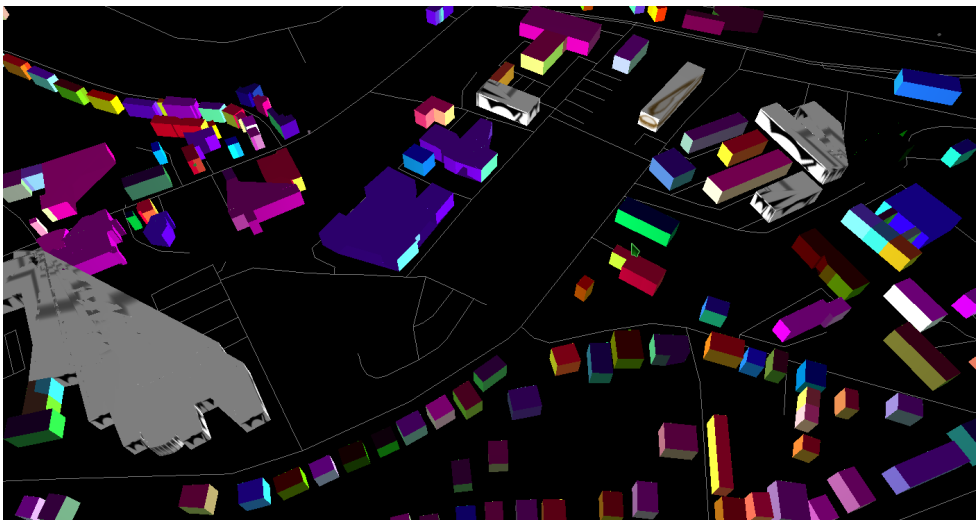


Figure 6.3: 3d model of the OpenStreetMap around UMIC

Now we have a 3D model of the cadastral map in the world coordinate system and

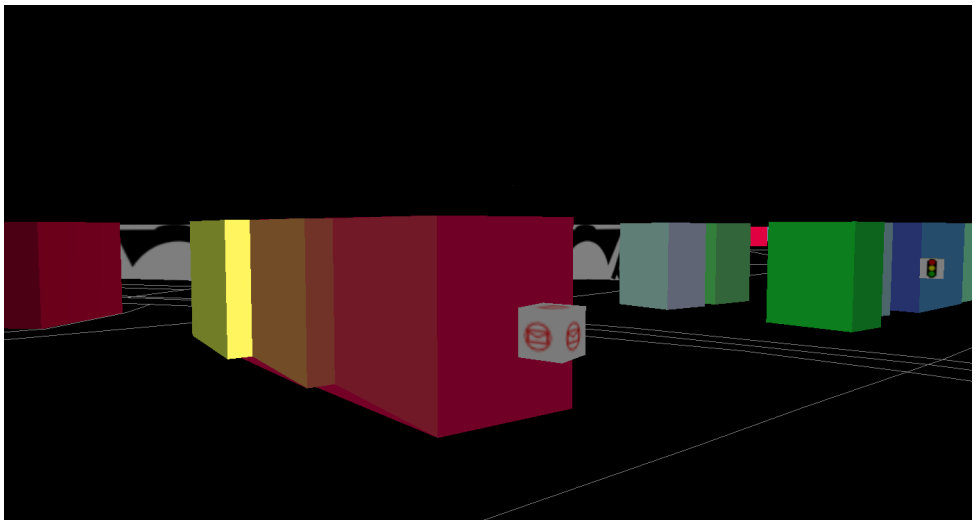


Figure 6.4: Visualization of a mail box and a traffic light, with an university in the background

the camera projection matrices $K_c[R_c|-R_cT_c]$ that project from the world coordinate system to the images, so we can project this model on the image sequence, either using our software renderer or OpenGL.

The software renderer basically sorts all building faces according to their distance from the camera, projects the faces to polygons on the images using the projection matrix and subtracts the polygons of near faces from the polygons of far faces. This has the advantage that we know for each face the exact projected polygon, but we cannot easily render textures and the number of intersecting polygons and their complexity each increases linearly which causes the rendering to run in $O(n^3)$.

In the OpenGL renderer we cannot directly set the projection matrix, because OpenGL always performs an additional viewport transform. We can however set $[R_c|-R_cT_c]$ as modelview matrix and K' as projection matrix where K' is a 4×4 variation of K that moves the distance from the 3rd to the 4th coordinate and inverts the viewport transformation. Then we can render the complete scene on the images, including streets and the building textures with semantic information.

This images can then be displayed to the user like in Fig. 6.5 or Fig. 6.6 to shown him where the desired buildings are located in an augmented reality fashion.

We can also match the objects detected in the object detection phase with the OpenStreetMap to assign semantic information to each object. Therefore we search for each object a building that is similar to the object, i.e. a building that is near enough to intersect with the object and is parallel to the object.

Each detected object is represented by a bounding box of size $o_x, o_y, o_z \in \mathbb{R}^3$ centered at $o_c \in \mathbb{R}^3$, i.e. the eight corners of the box are $o_c \pm o_x \pm o_y \pm o_z$. o_y is parallel to

Figure 6.5: Buildings with house numbers projected on a frame of an image sequence

Figure 6.6: Marking a pharmacy

the ground plane and $|o_z|$ very small, so that the box is approximately a plane[3].

To allow for alignment errors/noise we increase the size of each bounding box slightly by δ units along each axis $a \in \{x, y, z\}$, as follows:

$$o_a = o_a \frac{|o_a| + \delta}{|o_a|}$$

To compare this with the buildings, we write the faces of each building as planes of size $f_x, f_y \in \mathbb{R}^3$ centered at $f_c \in \mathbb{R}^3$, and convert them into similar bounding boxes by introducing another axis $f_z = (f_x \times f_y)\frac{\delta}{|f_x \times f_y|}$.

The two bounding boxes $o_c, o_x, o_y, o_z \in \mathbb{R}^3$ and $f_c, f_x, f_y, f_z \in \mathbb{R}^3$ do not overlap, if there exist a separating axis which is an axis where the intervals of the projections of both boxes are separated[Wik]. The interval of a projected bounding box b on any axis n is:

$$|\langle n, b_c \rangle| \pm (|\langle n, b_x \rangle| + |\langle n, b_y \rangle| + |\langle n, b_z \rangle|)|$$

Two intervals $r_1 \pm m_1$, $r_2 \pm m_2$ are separate if and only if $r_1 + m_1 < r_2 - m_2 \vee r_1 - m_1 > r_2 + m_2$ which leads to a simple test if n is a separating axis.

To definitely prove that two boxes intersect we check all axis o_a, f_a with $a \in \{x, y, z\}$ and the cross product between all pairs of them. To disprove an intersection it is sufficient to find a single separating axis which can usually be found after just three tests.

Additional to the intersection test we check if the building and the object are parallel by defining a maximal rotation threshold γ and checking the condition

$$\cos(f_z, o_z) = \frac{|\langle f_z, o_z \rangle|}{|f_z||o_z|} > \cos(\gamma).$$

If we compare all object with all buildings in this way, we can assign the semantic information of the buildings to the matching objects.

However, usually the object detection cannot differentiate between a single house and a row of houses, so that each the object spans across several buildings. Thus we detect one-to-many matches between an object and the buildings, and split the object to create a new single object for each matching building.

Thereby, we project all building facades that are potential matches on a one dimensional histogram along an normalized axis $\frac{o_x}{|o_x|}$, using the projection equation described above. In each bin of the histogram the building id and distance are stored and nearer buildings override farer buildings. After all buildings have been projected, we can easily extract the split positions $s_1, \ldots, s_n$, the bins at which the matched building id changes and which mark the positions where the object has to be split. We then create a new object for each split interval as described in Section 5.3, by keeping o_y, o_z and

[3]Remember that only one face of the building can be seen and reconstructed.

o_c along o_y, o_z constant, but setting the new length to $s_i - s_{i-1}$ and the center along o_x to the center of the interval $[s_i, s_{i-1}]$:

$$o_x^i = \frac{o_x}{|o_x|}\frac{s_i - s_{i-1}}{2} \quad \text{and} \quad o_c^i = o_c - \frac{o_x}{|o_x|}\left\langle \frac{o_x}{o_x}, o_c \right\rangle + \frac{o_x}{|o_x|}\frac{s_i + s_{i-1}}{2}$$

In future system it could be possible to use the semantic information in the visual odometry or the path matching phase to increase the result performance. Some ideas are to detect traffic lights, pedestrian crossings or the count of lanes in the images and compare these detections with the information of the map. Or to use the maximal allowed speed on a street to put constraints on the odometry.

The display of the buildings information in the images can also be further enhanced, e.g. the OpenStreetMap contains the opening time of stores and the reprojection could show if a store is currently opened.

6.4 Map improvement

Our system is also able to use the information provided by and estimated from the stereo image sequence to enhance or extend the cadastral map.

The most important map improvement is too add new buildings to the OpenStreetMap, which is possible with just the given image sequence and the old map, since the positioning tells us the position of our path in the map and the object detection finds the buildings in the local world coordinate system.

An object given as a bounding box with size $o_x, o_y, o_z \in \mathbb{R}^3$ centered at $o_c \in \mathbb{R}^3$ can be added to the map as follows: First we have to enlarge the object along the o_z axis, because the current bounding box only describes one face of the building. Because the camera can only see a face that it is the front side of a building, we search the nearest camera position c_n and enlarge the house in the direction $d_a = \frac{o_c - c_n}{|o_c - c_n|}$ away from the camera parallel to o_z. We assume a standard house size of $W = 10m$, so we have to move the center $\frac{W}{2}$ units in the direction of d_a and rescale the o_z vector to $\frac{W}{2}$ units:

$$o_z' = o_z \frac{W}{2|o_z|}\operatorname{sign}\langle o_z, d_a\rangle \quad \text{and} \quad o_c' = o_c + o_z \operatorname{sign}\langle o_z, d_a\rangle \frac{W}{2}$$

.

The new bounding box is a building of which we can project all corners $o_c' \pm o_x \pm o_y \pm o_z'$ on the ground plane by keeping only the points on the bottom of the building, which changes the corners to $o_c' \pm o_x - o_y \pm o_z'$.

Then we transform these coordinates into geodetic coordinates as explained in Appendix B.2 and add the final corners as new building corners to the OpenStreetMap.

Altogether we can suggest following work flow for improving OpenStreetMaps:

1. Drive through the streets whose buildings are missing in the OpenStreetMap

2. Let our system automatically estimate the path, create the reconstruction and detect buildings in it
3. Manually remove all incorrect building objects and add missing ones, in the interactive 3d visualization
4. Use the matching algorithms to position the path in the map and remove already known buildings automatically.
5. Export the map as new OpenStreetMap

This allows someone with just a stereo camera to extend OpenStreetMaps in an accurate way, and does not require any aerial support like GPS or satellite images, unlike the methods currently used.

The image sequence also allows us to add textures to the 3d model which are not obviously not contained in a cadastral map, as shown in Fig. 6.7. Thereby we determine for every pixel of every frame the point on the building that is projected on this pixel by intersecting a camera ray with the building face and create a texture from all these pixels by projecting them from the building face into the texture space.

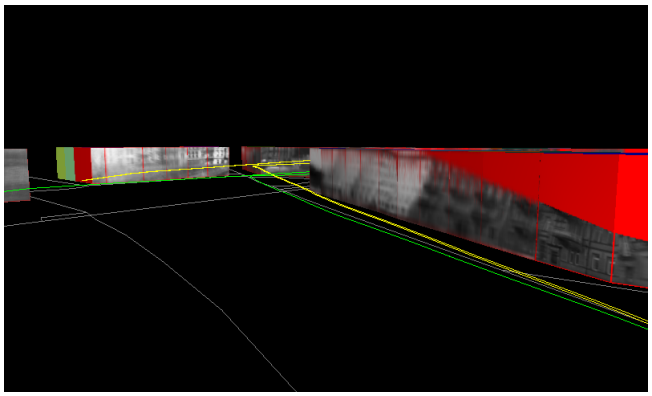


Figure 6.7: Textures from the images added to the OpenStreetMap

The texture coordinate system of a building face maps $(0,0)$ and $(1,1)$ to opposite corners, so we can define a projection matrix from a point on the face in world coordinates to texture coordinates, by placing a camera aligned to the face directly before the face. If the face is given like before as $f_c, f_x, f_y, f_z \in \mathbb{R}$ with corners $f_c \pm f_x \pm f_y$ and normal f_z, this matrix becomes:

$$K_t[R_t|-R_tT_t] = \begin{pmatrix} \frac{1}{2|f_x|} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2|f_y|} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix} \left(\begin{array}{c|c} \begin{matrix} r^t \\ u_y \\ -n^t \end{matrix} & -\begin{pmatrix} r^t \\ u^t \\ -n^t \end{pmatrix}^{-1} (f_c+n) \end{array} \right)$$

where $r = \frac{f_x}{|f_x|}$, $u = \frac{f_y}{|f_y|}$, $n = \frac{f_z}{|f_z|}$ are the normalized forward, up and normal vector[4].

All points P on the plane of this building face have to satisfy the usual plane equation:

$$n^t P - n^t f_c = 0$$

In Section 4.1 it was shown that all points P on the ray from the camera origin through the pixel $(p_x, p_y)^t$ of the image plane are given by:

$$P = R^{-1}K^{-1}\begin{pmatrix} Dp_x \\ Dp_y \\ D \end{pmatrix} + T,$$

where D is the distance D of the point. Inserting this in the plane equation leads to a basic geometric equation that can easily be solved for D:

$$n^t(R^{-1}K^{-1}\begin{pmatrix} Dp_x \\ Dp_y \\ D \end{pmatrix} + T) - n^t f_c = 0$$

$$Dn^t(R^{-1}K^{-1}\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} + T) - n^t f_c = 0$$

$$D = \frac{n^t f_c}{n^t(R^{-1}K^{-1}\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} + T)}$$

D can again be inserted in the ray equation which gives us P, the point on the building that was projected to (p_x, p_y). By projecting P with $K_t[R_t| - R_t T_t]$ in the texture coordinate system we therefore learn the texture pixels that we have to update with the color of the pixel (p_x, p_y).

We can either intersect the ray with all building faces or use the output of the software renderer and only intersect the ray with the face that was projected on this pixel. In any cases we repeat it for all pixels to get color information for all building textures, and average the colors for each pixel on texture. In our system the camera images have a higher resolution than the texture images, so we do not have to worry about missing pixels in the textures.

A possible alternative to the rays is to use both projection matrices directly and create a homography that transforms the pixels from the image coordinates to the texture coordinates[Lei].

[4] In the implementation the matrix is slightly more complex, because we share the same texture for all faces of a building.

Chapter 7

Evaluation

To evaluate our system we apply it to several image sequences and compare the results with the ground truth of the sequence.

The requirements for a data set to be useable in our case are quite high: The sequence has to consist of stereo images with a known calibration and stable, non-jerking camera movement. A ground truth is required to evaluate the quality of an estimated path and this ground truth must also contain GPS data in order to evaluate the quality of the offline matching. And finally the sequence has to be recorded at a place with buildings and good OpenStreetMap support, the map must contain streets as well as building information. Especially latter is still missing in many OpenStreetMaps.

So we perform almost all our evaluation with the large Karlsruhe dataset [KGL10] [SPF10] which consist of several, different sequences recorded from a car in Karlsruhe. Each frame of a sequence consists of two undistorted and rectified stereo images of size 1344×372 and for each sequence a GPS ground truth is given by a GPS/IMU system. This system provides a geodetic and local Cartesian camera position, and a yaw angle[1]. In relation to the camera the GPU/IMU unit is translated by $(1.6, 0.6, -0.05)^t$ metres and tilted by -4.6 degree. Each Karlsruhe sequence has a unique id generated from the recorded ISO-date and the drive number, like 20090908d10.
The final advantage of the Karlsruhe sequences is that the associated OpenStreetMaps contain all streets and most of the buildings.

To evaluate the visual odometry and the online matching component, we compare the results of our normal visual odometry and OpenStreetMap corrected visual odometry with the ground truth and the results of the Libviso libraries[KGL10] on multiple sequences.

The testing system is implemented in the visualizer and will load the images of a sequence, the cadastral map as well as configuration files describing the parameters for the separate odometry programs, and then automatically execute all four odometries in

[1] Actually it provides all Euler angles, but roll and pitch are invalid due to a bug in their software.

parallel. During the path estimation, the paths are visualized, similar to Fig. 7.1, and afterwards all created paths and the command lines necessary to reproduce each path are written to the file system.

Before the visual odometry with online matching can be performed, it is necessary to align the initial camera state with the cadastral map. We do this by automatically taking the starting position from the GPS ground truth, and manually rotating the ground truth path until it is aligned with the streets of the map. One could also use the offline path matching algorithm for this alignment, but we want to test each component independently.
To use Libviso it was necessary to write a small wrapper program, that performs the necessary library calls and reads/writes the files in our format.

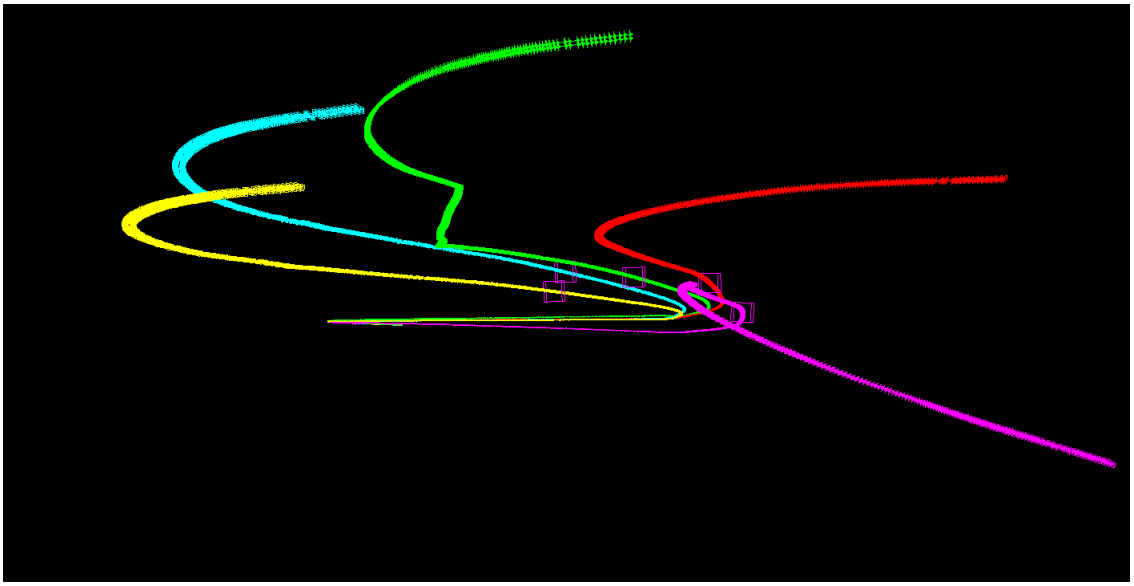


Figure 7.1: A tentacle monster, a visualization of the estimated paths. The ground truth is green, our uncorrected VO yellow, our corrected VO blue, Libviso 2 and 1 are violet and red

Finally the visualizer will create a plot of the distance between all estimated paths and the ground truth in each frame, calculate the average distance and standard deviation of the distance, and create a table containing this information for each sequence and algorithm. The distance average is first taken over all frames of a sequence, and then over the average of the sequences, so that frames in longer sequences have less weight. This compensates the fact that the distance between the estimated path and ground truth is always larger in longer sequences due to the accumulation of errors.

According to the final average distance of Table 7.1 our algorithms performs worse than libviso, and especially the map-corrected version does not work at all.

However, this result is mostly caused by the completely failed odometry of sequence 20100304d32. If we exclude this sequence the average distances to the ground

Sequence id	Our VO		Our VO + OSM		Libviso 2		Libviso 1	
20090908d10	6.1	± 3.5	3.3	± 1.5	6.6	± 6.9	5.2	± 5.3
20090908d12	17.3	± 7.1	13.9	± 5.4	25.9	± 13.0	17.7	± 8.9
20090908d15	27.6	± 17.0	8.4	± 4.3	5.7	± 2.3	3.7	± 1.2
20090908d21	5.5	± 2.4	3.2	± 1.0	4.7	± 2.7	4.7	± 2.3
20091214d15	4.6	± 1.7	5.4	± 2.0	18.3	± 14.2	22.3	± 16.4
20100304d32	71.1	± 88.6	226.0	± 154.5	26.1	± 16.3	25.8	± 15.6
20100304d33	6.1	± 4.7	7.5	± 5.7	6.4	± 4.8	4.5	± 3.0
20100305d23	41.7	± 33.7	12.1	± 6.0	10.5	± 7.0	13.6	± 12.1
20100309d20	9.2	± 3.2	18.3	± 5.0	14.9	± 10.1		
20100317d46	4.9	± 2.1	7.5	± 4.5	6.7	± 3.3		
average	19.4	± 16.4	30.5	± 18.9	12.5	± 8.0	12.1	± 8.1

Table 7.1: Average distance of each path

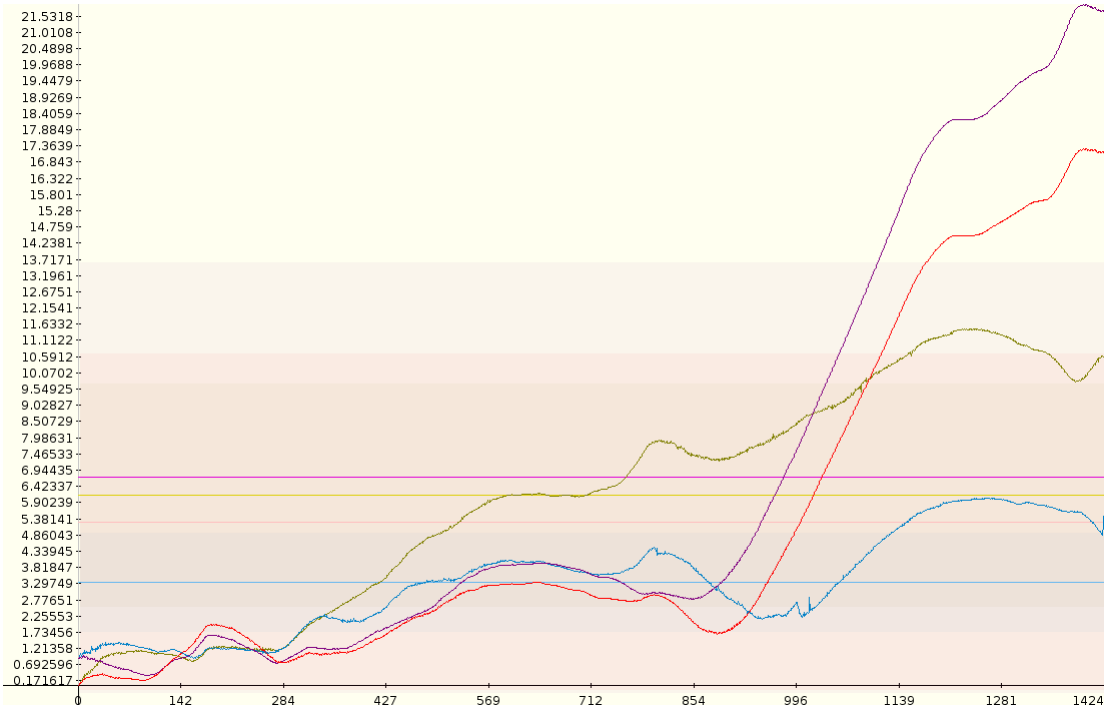


Figure 7.2: Plot of the paths of Karlsruhe sequence 20090908d10

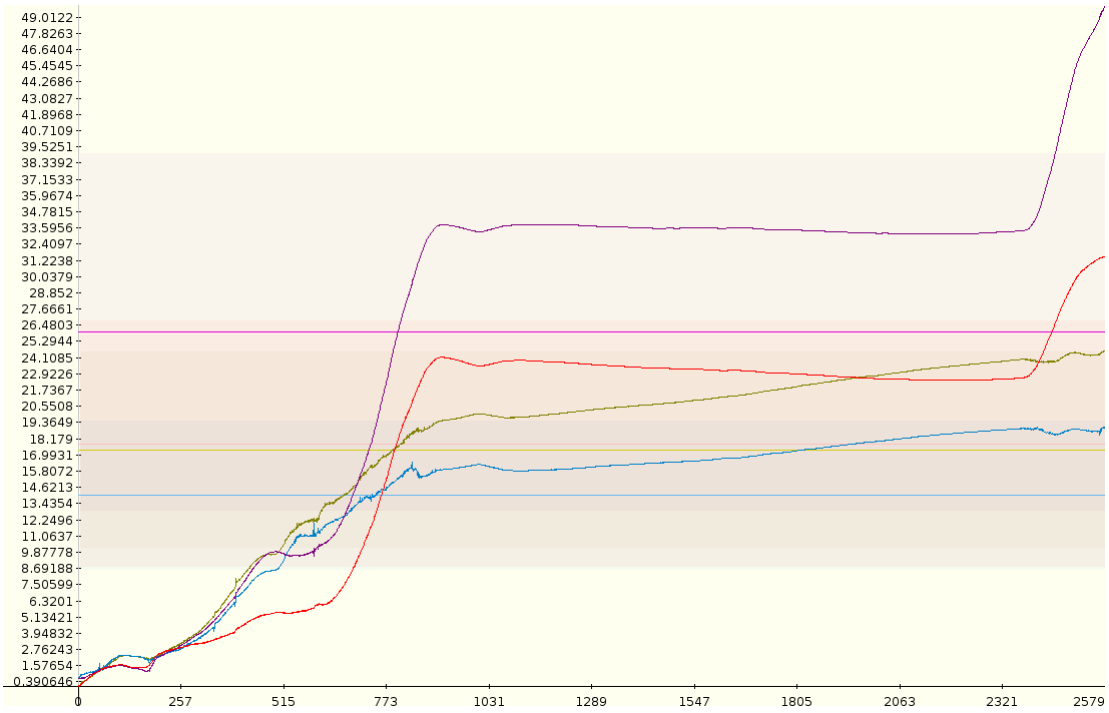


Figure 7.3: Plot of the paths of Karlsruhe sequence 20090908d12

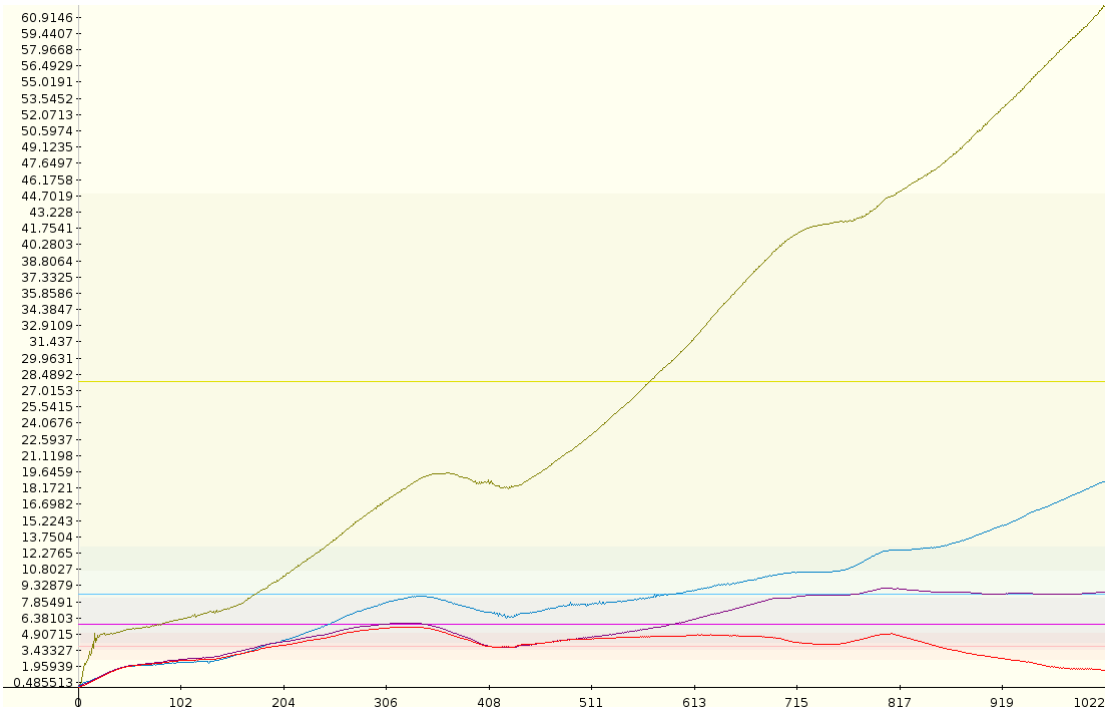


Figure 7.4: Plot of the paths of Karlsruhe sequence 20090908d15

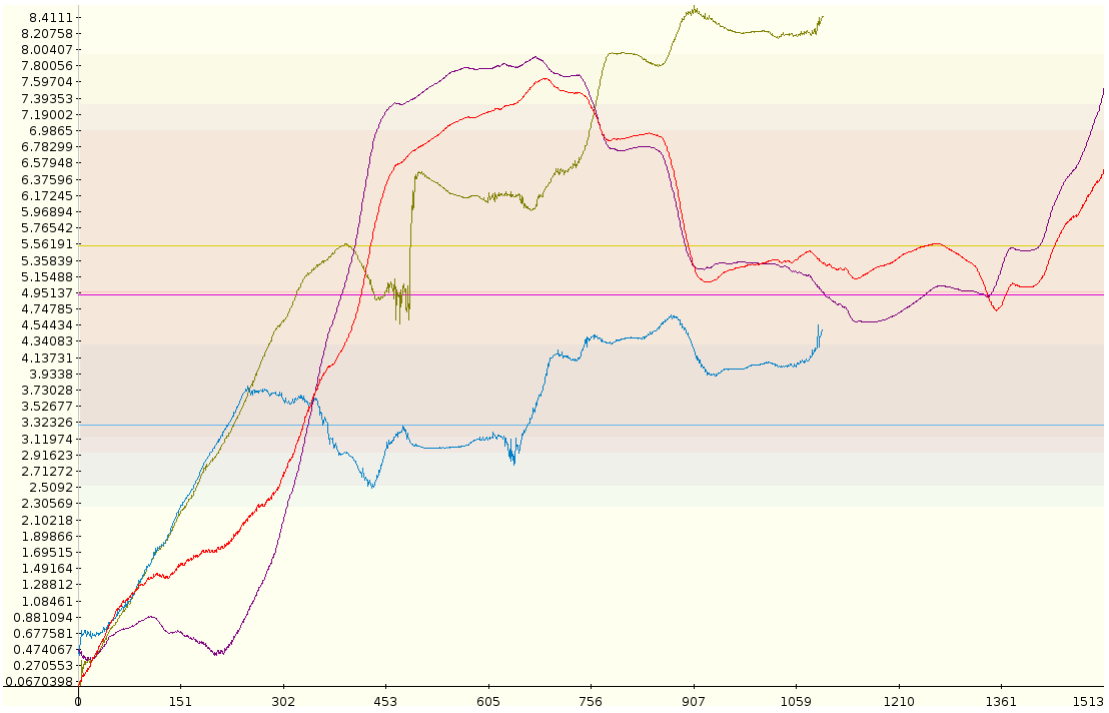


Figure 7.5: Plot of the paths of Karlsruhe sequence 20090908d21

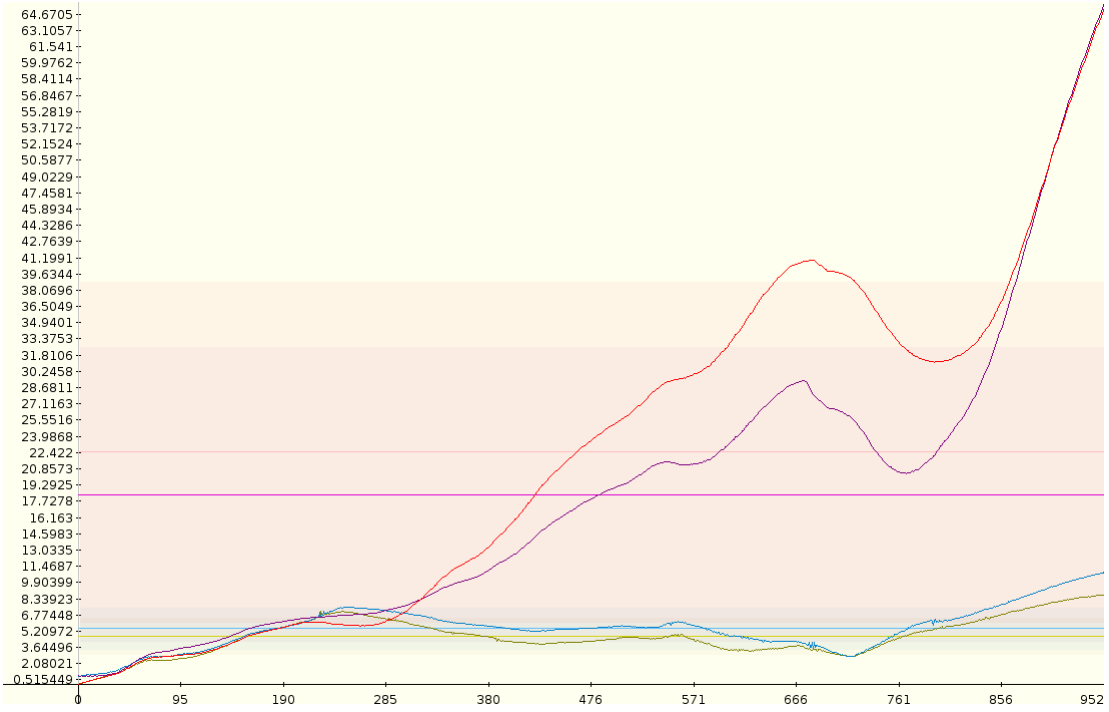


Figure 7.6: Plot of the paths of Karlsruhe sequence 20091214d15

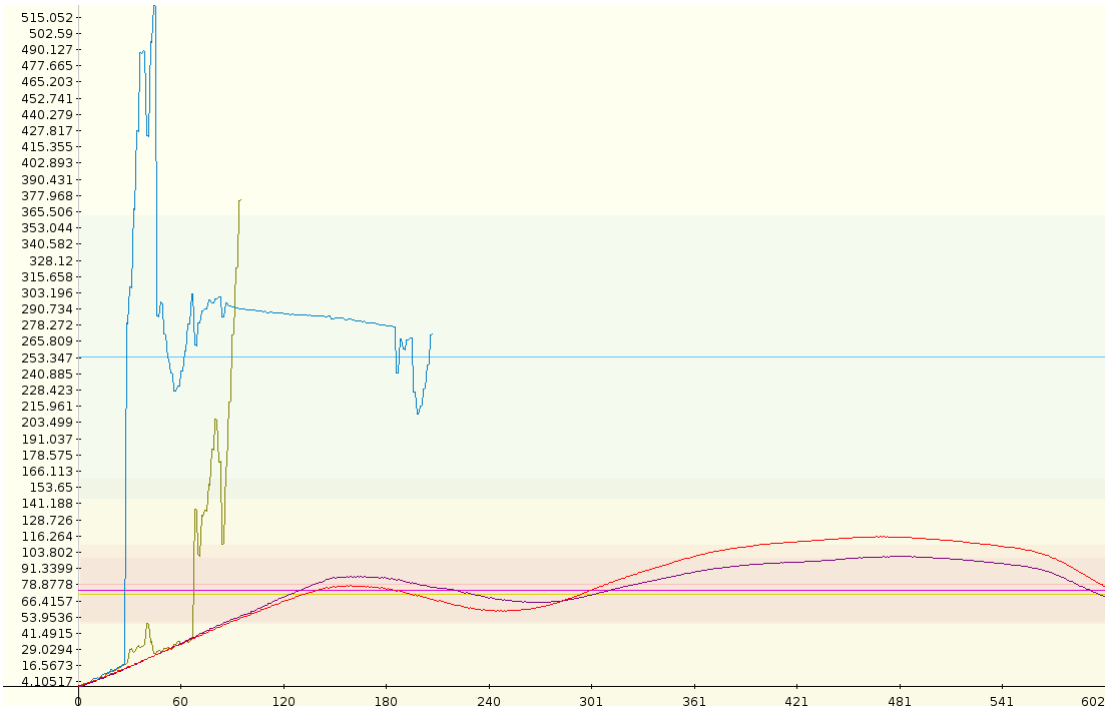


Figure 7.7: Plot of the paths of Karlsruhe sequence 20100304d32

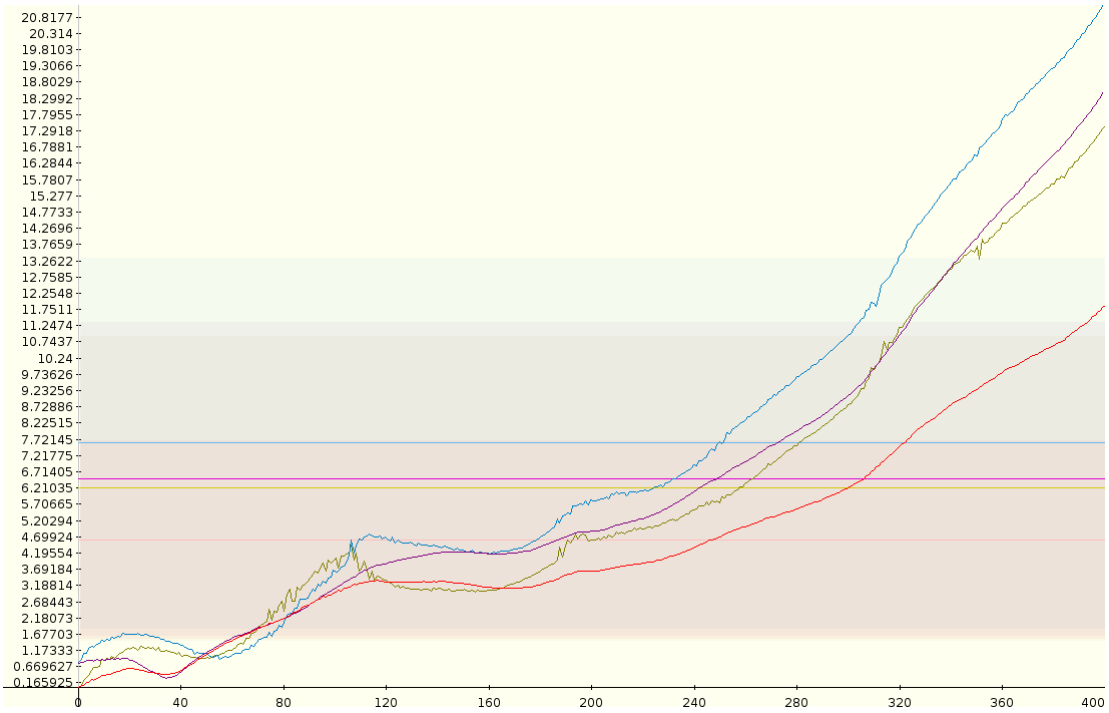


Figure 7.8: Plot of the paths of Karlsruhe sequence 20100304d33

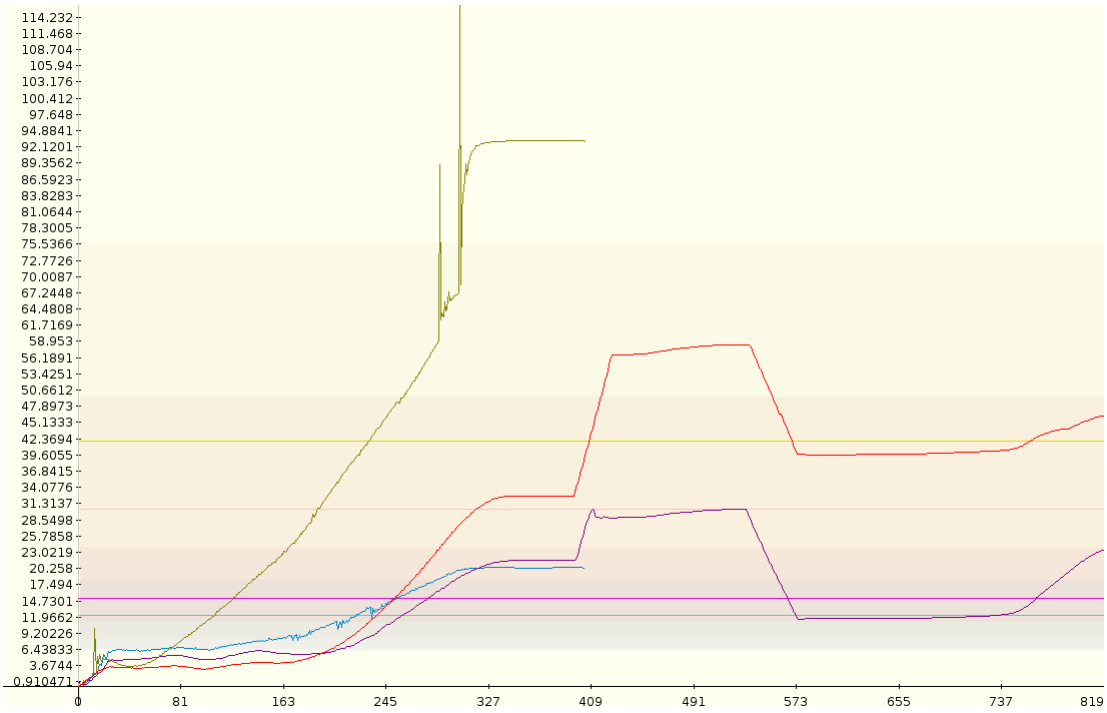


Figure 7.9: Plot of the paths of Karlsruhe sequence 20100305d23

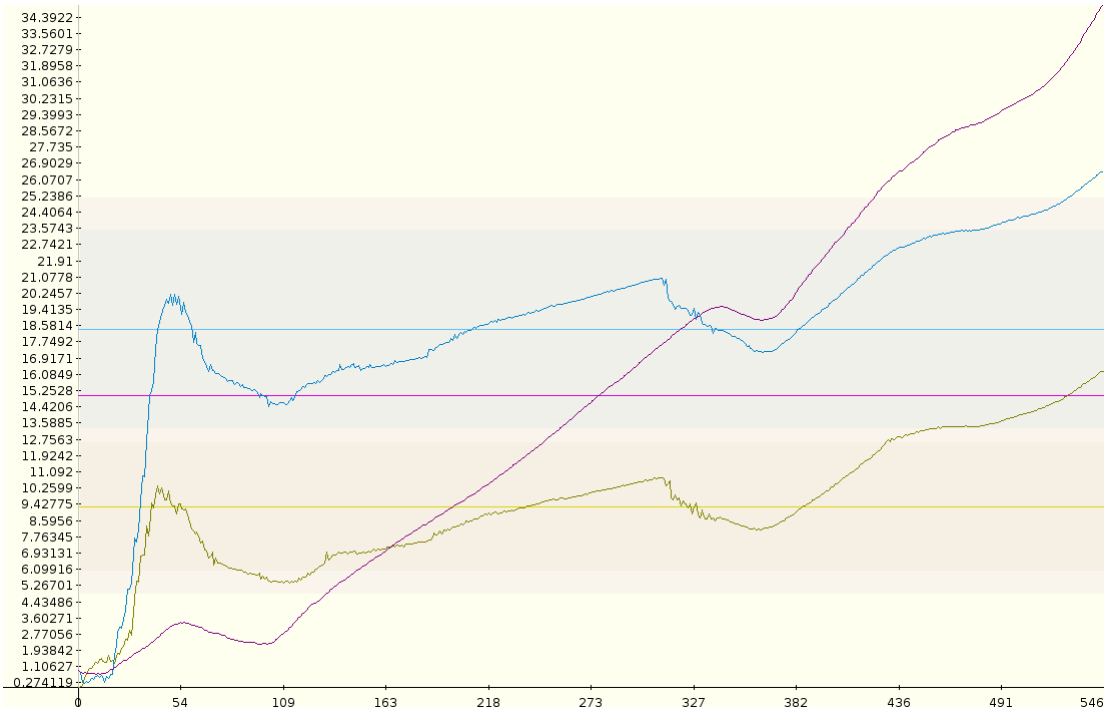


Figure 7.10: Plot of the paths of Karlsruhe sequence 20100309d20

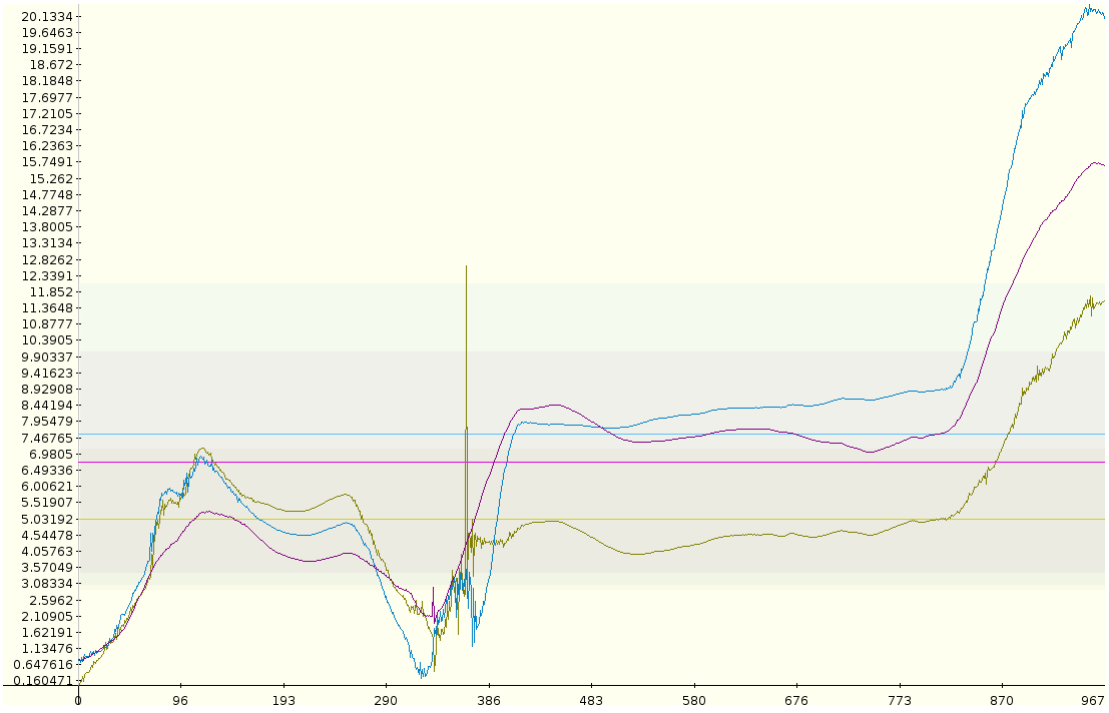


Figure 7.11: Plot of the paths of Karlsruhe sequence 20100317d46

truth is 13.7m for the uncorrected odometry, 8.2m for our corrected odometry, 11.0m and 10.6m for Libviso 2 and 1. So the corrected version becomes the best of the four tested algorithms, which shows that access to a cadastral map can be very helpful.

Excluding this sequence is also justified by the fact that the path is aborted after a third of the sequence, because our odometry detects that it has become lost, as you can see in Fig. 7.7. It is also the sequence where Libviso performed worst, which further reduces the possible bias of removing it.
Nevertheless, one also has to notice the fact that the corrected odometry version was developed with access to the Karlsruhe sequences, so it may be optimally tuned to theses sequence and perform worse on other sequences. However, the same might be true for libviso, since the Karlsruhe sequences form the dataset provided with libviso.

Figure 7.2 to Fig. 7.11 show plots of the distance between the camera position of the ground truth and of each estimated path over all frames, where our basic path is golden, our corrected path is blue, and the paths of Libviso 2 and 1 are respectively red and violet.

Systematical testing of the reconstruction component is not really feasible, since we do not have a ground truth of the 3D point cloud, but it is also not really needed, since the quality of the reconstruction depends mostly on the well-known libelas li-

brary. Therefore we just run the reconstruction on different sequences and check, if it looks "nice". It does, as seen in Fig. 7.12

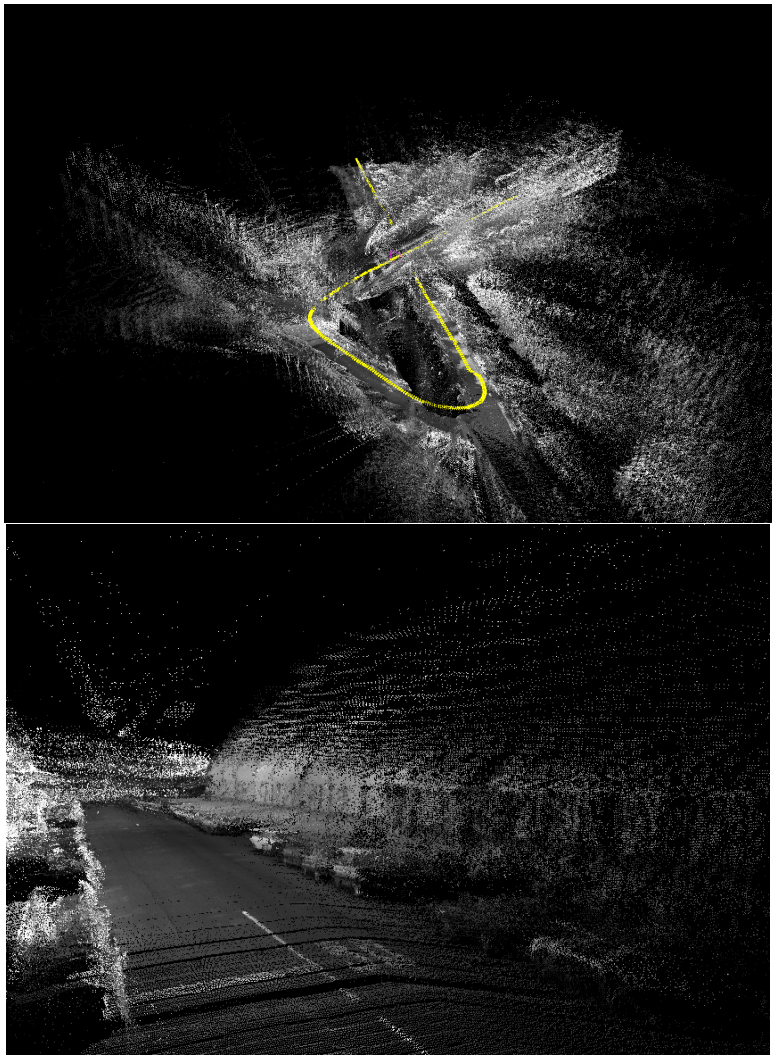


Figure 7.12: Looking on and backwards to a scene

A simple test when viewing the reconstruction is to look backwards along the estimated path, if the scene is not distorted and still recognizable, the reconstruction is probably good.

A scientific evaluation of the object detection is also difficult, because we neither have a ground truth for the objects nor a clear definition, what we consider to be an object. So we perform the object detection for several sequences and manually check the results.

For each detected object we check the following properties:

Support: If this object is supported by the point cloud, as every valid object has to be. If there are no supporting points we call the object invalid, if it has some supporting points, but also holes, we call it partial(ly invalid).

Completeness: If the detected object is not too small or only covers a part of the true object. To identify the true object, we use human intuition about the position of the next building facades and nears cars.

Overlap: If the detected object does not overlap with another. Every pair of overlapping hypotheses is only counted once.

Missclassified: If the classification of the object as building or car is correct, i.e. a missclassified building object is actually a car.

We search manually for buildings and cars in the reconstruction and count how many of these objects were not detected which we note as "missing" objects. Buildings that should be there according to the image sequence, but are due to noise even for a human not visible in the point cloud, are not counted as missing.
The results are listed in Fig. 5.5, and show that buildings that have been detected are mostly correct, but many buildings are not found. However, car detection are much more unreliable with many incomplete, partial and missing cars.
This is explained by the fact that very often a row of multiple cars is explained by a single detected car, or a single car in a row is covered by two detections. And since the detection is based on line detection, cars that are not near to another one, are not found at all.

Since it is possible to interactively edit the set of hypotheses in the visualizer, we could remove invalid detections, correct slightly misaligned detections and add all missing objects to create a ground truth for a certain reconstruction. This ground truth could then be used in future evaluations to compare the performance of modifications to the object detector in an automated testing framework.

The offline positioning is evaluated by repeatedly executing the algorithm for several sequences with different parameters and comparing the result with the GPS information provided the groundtruth. The distance from the position estimated for the first frame to the first frame of the ground truth is used as single score for the quality of an aligment. We ignore the rotation because it is not directly given by the ground truth, and it is unlikely that the positioning algorithm would find the correct position with a wrong rotation. Enter We use a testing framework that takes a list of configuration files which contain the different parameters for the positioning algorithm and a list of estimated paths with cadastral maps, and calls the positioning algorithm for all pairwise combinations of parameters and sequences. Such an exhaustive approach is here

		Building objects					
Sequence	Missing	Count	Invalid	Partial	Incomplete	Overlap	Missclassified
090908d10	2	4		1			
090908d12	4	5			1		
100309d19	2	9				3	
100309d20	6	0					
100309d23		3	1		1	1	
100309d51	1	5	1				
100309d81		0					
		Car objects					
Sequence	Missing	Count	Invalid	Partial	Missclassified		
090908d10	14	21	2	8	1		
090908d12	21	63	15	5	1		
100309d19	12	41		4			
100309d20		8	2	4	2		
100309d23	1	37	25				
100309d51	16	24	4	10	3 (bridge)		
100309d81	~ 10	12	4		3		

Table 7.2: Evaluation of the object detection

feasible, because the path and the cadastral map are very small compared to a large image sequence or a complete point cloud.

We have evaluated it for 7488 cases, and the results shown in Fig. 7.13 view that the parameters do not really matter. The error distance of a certain sequence is almost the same for all parameters, so the quality of the result depends mostly on the path itself.
This is easy to understand, because the positioning algorithm can only perform well if the path contains distinct turns, and will always fail if the camera just moves straight ahead, because there exist many unrelated straight streets in a city.

Table 7.3 to Table 7.6 show a clearer interpretation of the data points of Fig. 7.13. We assume that the positioning algorithm has worked correct, if the distance between estimate and true position is less than 15m and count for all tested parameters the percentage of correct estimated sequences. In Table 7.3 and Table 7.5 the algorithm is performed using the ground truth path, in Table 7.4 and Table 7.6 it uses the path given by our visual odometry. In the first two tables Table 7.3 and Table 7.4 we only use small, easier cadastral maps of sizes around $300 \times 300m$, in Table 7.3 and Table 7.4 also larger maps with sizes up $800 \times 600m$ are included.

It appears that using the Chamfer scores results in a basic, robust performance, changing the parameters has almost no effect on the quality of the estimation, however, the results are not optimal. The Outlier score has a high dependency on the chosen threshold, which causes it to perform either really good or really bad. And finally the force-on-street score is the golden mid, not so sensible to parameter changes, but

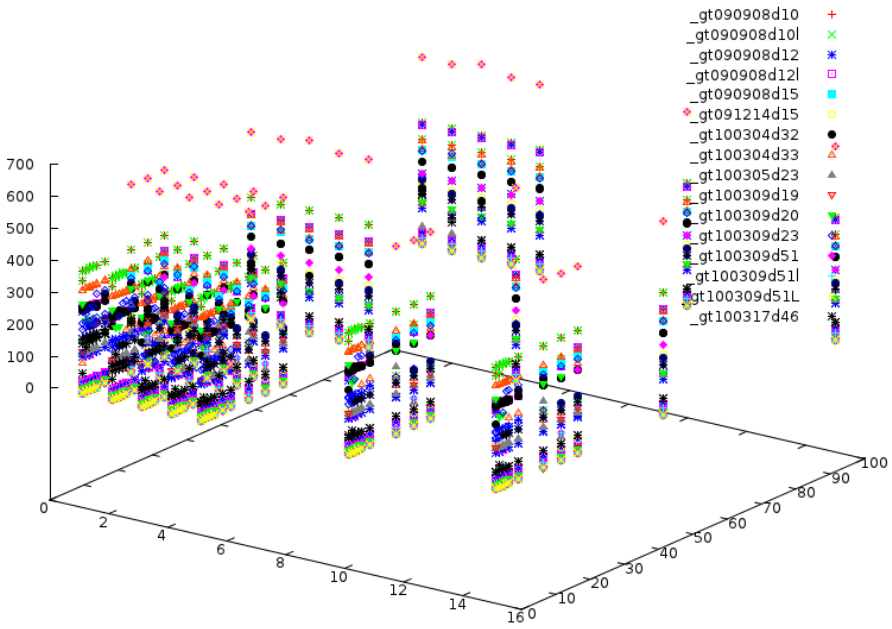


Figure 7.13: The error distance (on z-axis) of applying the positioning algorithm to a sequence with different map scale (1 to 15) and different thresholds (1 to 100)

also not as high scoring. It might however perform comparatively better on more difficult maps, so it would be wise to repeat this evaluation with more sequences and less different parameters.

scale:	1	2	3	4	5	10	15
CHAMFER							
1.00	50%	50%	50%	50%	50%	50%	50%
OUTLIER							
inlier							
1.00	44%	44%	44%	44%	44%	44%	44%
2.00	44%	44%	44%	44%	44%	44%	44%
3.00	44%	44%	44%	44%	44%	44%	44%
4.00	50%	50%	56%	56%	56%	56%	56%
5.00	56%	56%	56%	56%	56%	56%	56%
7.50	56%	56%	56%	56%	56%	56%	56%
10.00	/	/	/	/	38%	/	/
15.00	19%	19%	19%	19%	19%	19%	19%
20.00	6%	12%	12%	6%	6%	6%	6%
25.00	0%	0%	0%	0%	0%	0%	0%
50.00	0%	0%	0%	0%	0%	0%	0%
100.00	0%	0%	0%	0%	0%	0%	0%
FOS							
max jump							
1.00	44%	50%	50%	50%	50%	44%	44%
2.00	56%	56%	56%	56%	56%	56%	56%
3.00	50%	56%	56%	56%	56%	56%	56%
4.00	50%	50%	50%	50%	50%	50%	50%
5.00	50%	50%	50%	50%	50%	50%	50%
7.50	50%	50%	50%	50%	50%	50%	50%
10.00	/	/	/	/	50%	/	/
15.00	50%	50%	50%	50%	50%	50%	50%
20.00	50%	50%	50%	50%	50%	50%	50%
25.00	50%	50%	50%	50%	50%	50%	50%
50.00	50%	50%	50%	50%	50%	50%	50%
100.00	50%	50%	50%	50%	50%	50%	50%

Table 7.3: Evaluation of different scores, percent of positions found with less than 15 m error on 192 test cases for each pair of parameters, aligning the ground truth path with the cadastral map

scale:	1	2	3	4	5	10	15
CHAMFER							
1.00	44%	56%	50%	56%	50%	50%	50%
OUTLIER							
inlier							
1.00	19%	25%	19%	19%	19%	12%	12%
2.00	38%	38%	44%	31%	31%	31%	31%
3.00	44%	50%	44%	50%	44%	44%	44%
4.00	50%	56%	50%	56%	50%	50%	50%
5.00	56%	62%	56%	62%	56%	56%	50%
7.50	44%	50%	44%	50%	44%	44%	44%
10.00	/	/	/	/	44%	/	/
15.00	19%	25%	25%	25%	25%	25%	25%
20.00	12%	12%	12%	12%	6%	12%	12%
25.00	12%	12%	6%	6%	6%	6%	6%
50.00	0%	0%	0%	0%	0%	0%	0%
100.00	0%	0%	0%	0%	0%	0%	0%
FOS							
max jump							
1.00	19%	6%	19%	13%	12%	25%	25%
2.00	38%	44%	38%	44%	38%	38%	38%
3.00	56%	50%	38%	50%	44%	44%	44%
4.00	50%	50%	44%	50%	44%	38%	44%
5.00	50%	56%	44%	50%	44%	44%	44%
7.50	50%	56%	50%	56%	50%	50%	50%
10.00	/	/	/	/	50%	/	/
15.00	50%	50%	50%	56%	50%	50%	50%
20.00	50%	50%	50%	56%	50%	50%	50%
25.00	50%	50%	50%	56%	50%	50%	50%
50.00	50%	50%	50%	56%	50%	50%	50%
100.00	50%	50%	50%	56%	50%	50%	50%

Table 7.4: Evaluation of different scores, percent of positions found with less than 15 m error on 192 test cases for each pair of parameters, aligning the estimated path with the cadastral map

scale:	1	2	3	4	5	10	15
CHAMFER							
1.00	50%	50%	50%	50%	50%	50%	50%
OUTLIER							
inlier							
1.00	44%	44%	44%	44%	44%	44%	44%
2.00	44%	44%	44%	44%	44%	44%	44%
3.00	44%	44%	44%	44%	44%	44%	44%
4.00	50%	50%	56%	56%	56%	56%	56%
5.00	56%	56%	56%	56%	56%	56%	56%
7.50	56%	56%	56%	56%	56%	56%	56%
10.00	/	/	/	/	38%	/	/
15.00	19%	19%	19%	19%	19%	19%	19%
20.00	6%	12%	12%	6%	6%	6%	6%
25.00	0%	0%	0%	0%	0%	0%	0%
50.00	0%	0%	0%	0%	0%	0%	0%
100.00	0%	0%	0%	0%	0%	0%	0%
FOS							
max jump							
1.00	44%	50%	50%	50%	50%	44%	44%
2.00	56%	56%	56%	56%	56%	56%	56%
3.00	50%	56%	56%	56%	56%	56%	56%
4.00	50%	50%	50%	50%	50%	50%	50%
5.00	50%	50%	50%	50%	50%	50%	50%
7.50	50%	50%	50%	50%	50%	50%	50%
10.00	/	/	/	/	50%	/	/
15.00	50%	50%	50%	50%	50%	50%	50%
20.00	50%	50%	50%	50%	50%	50%	50%
25.00	50%	50%	50%	50%	50%	50%	50%
50.00	50%	50%	50%	50%	50%	50%	50%
100.00	50%	50%	50%	50%	50%	50%	50%

Table 7.5: Evaluation of different scores, percent of positions found with less than 15 m error on 256 test cases for each pair of parameters, aligning the ground truth path with the cadastral map

scale:	1	2	3	4	5	10	15
CHAMFER							
1.00	44%	56%	50%	56%	50%	50%	50%
OUTLIER							
inlier							
1.00	19%	25%	19%	19%	19%	12%	12%
2.00	38%	38%	44%	31%	31%	31%	31%
3.00	44%	50%	44%	50%	44%	44%	44%
4.00	50%	56%	50%	56%	50%	50%	50%
5.00	56%	62%	56%	62%	56%	56%	50%
7.50	44%	50%	44%	50%	44%	44%	44%
10.00	/	/	/	/	44%	/	/
15.00	19%	25%	25%	25%	25%	25%	25%
20.00	12%	12%	12%	12%	6%	12%	12%
25.00	12%	12%	6%	6%	6%	6%	6%
50.00	0%	0%	0%	0%	0%	0%	0%
100.00	0%	0%	0%	0%	0%	0%	0%
FOS							
max jump							
1.00	19%	6%	19%	13%	12%	25%	25%
2.00	38%	44%	38%	44%	38%	38%	38%
3.00	56%	50%	38%	50%	44%	44%	44%
4.00	50%	50%	44%	50%	44%	38%	44%
5.00	50%	56%	44%	50%	44%	44%	44%
7.50	50%	56%	50%	56%	50%	50%	50%
10.00	/	/	/	/	50%	/	/
15.00	50%	50%	50%	56%	50%	50%	50%
20.00	50%	50%	50%	56%	50%	50%	50%
25.00	50%	50%	50%	56%	50%	50%	50%
50.00	50%	50%	50%	56%	50%	50%	50%
100.00	50%	50%	50%	56%	50%	50%	50%

Table 7.6: Evaluation of different scores, percent of positions found with less than 15 m error on 256 test cases for each pair of parameters, aligning the estimated path with the cadastral map

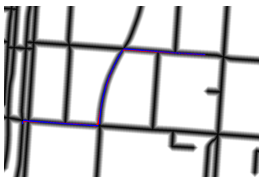


Figure 7.14: Positioning of the sequence 20090908d15

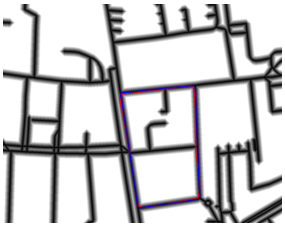


Figure 7.15: Positioning of the sequence 20091214d51

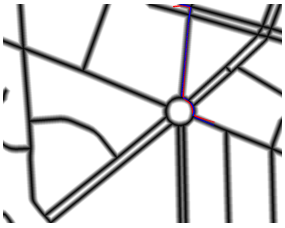


Figure 7.16: Positioning of the sequence 20100305d20

Figure 7.17: Positioning of the sequence 20100309d20

Figure 7.18: Positioning of the sequence 20100317d46

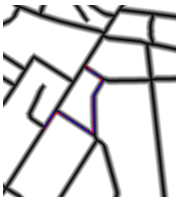


Figure 7.19: Positioning of the sequence karlsruhe19

Figure 7.20: Positioning of the sequence 20090908d10 with large map

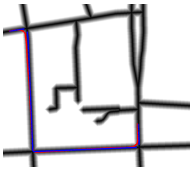


Figure 7.21: Positioning of the sequence 20090908d10

Figure 7.22: Positioning of the sequence 20090908d12 with large map

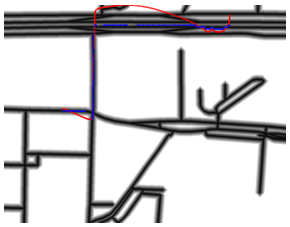


Figure 7.23: Positioning of the sequence 20090908d12

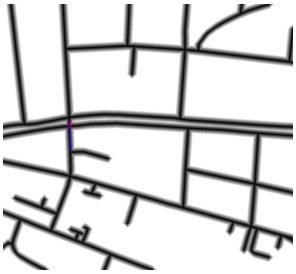


Figure 7.24: Positioning of the sequence 20100309d23

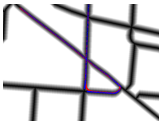


Figure 7.25: Positioning of the sequence 20100309d51

Figure 7.26: Positioning of the sequence 20100309d51 with large map

Figure 7.27: Positioning of the sequence 20100309d51 with LARGE map

Chapter 8

Conclusion and Future Work

We have shown that the combination of local visual scene information with global cadastral maps can actually improve the results and even extend the map.

The reimplementation of the uncorrected visual odometry and the reconstruction works as expected, and the novel inclusion of street information within the odometry shows that the cadastral map improves the accuracy of the path in most cases.

The object detection is able to detect many buildings and cars, although it still makes a lot of negative and positive errors, so we plan to improve it in future works. One important experiment would be use a semantic understanding library that can classify all pixels of an image according to a semantic class, and annotate each 3d point in the reconstruction with the semantic label. Then the object detection can be constrained to only use points with the correct labels, which should already improve the results.
Since the object detection depends on several system parameters, we should probably reevaluate it for different sets of parameter using an automated test framework with a manually created ground truth for each sequence. We should also perform this evaluation on more sequences.

The matching with the the cadastral map was proved to be helpful:
The offline positioning can align many paths with the cadastral map, even though it fails in some sequences and we should examine these sequences to improve the positioning algorithm. It is clear that the current idea cannot work for pure, straight paths, but once the object detection accuracy is improved, it might be possible to add the distance between the detected buildings and the true building to the score of an alignment. If we import a semantic image library, we could also directly judge the distance of all detected building points to the nearest building of the aligned map, without detecting which points form to an object. We can also try to detect more semantic objects like traffic signals in the image sequence and check if they exist at a candidate position.
The online positioning performed well, but we might be able to improve it even more by simulating a repulsive force emerging from the building facades with the Kalman filter to model the fact that the camera path may not intersect buildings in a natural

way.
The performance of the other matching components depends mostly on the quality of the estimated path and the object detection. If the path and the cadastral map are correct, the reprojected buildings are correct, too. Similar, if the path and the detect objects are correct, we can add valid buildings to the cadastral map.
So, if we improve the general system performance, we could try to establish our system as a new, official OpenStreetMap editor.

Appendix A

Notations

$K = K_l$	3×3 Intrinsic matrix of the left camera
K_r	3×3 Intrinsic of the right camera
$R = R_l$	3×3 Rotation matrix of the left camera
R_r	3×3 Rotation matrix of the right
$T = T_l$	Position of the left camera
T_r	Position of the right camera
$[R\|-RT]$	3×4 Projection from the global CS to the left camera CS
$t = -RT$	Rotated position of the left camera
$M_1 M_2$	Multiplication of the matrices M_1, M_2
M^t	Transpose of M
M^{-1}	Inverse of M
$m_{i,j}$	Element (i, j) of matrix m
$m_{i,\cdot}$	i-th row of matrix m
$m_{\cdot,j}$	j-th column of matrix m
m^t	transpose of matrix m
v_i	i-th entry of vector v
v_x, v_y, v_z	1st to 3rd entry of a vector p
$\langle v \rangle$	mean of the entries of vector v
$\sigma(v)$	standard derivation of the entries of vector v
$\langle v, v' \rangle$	scalar product between v and v'
$\|v\| = \|v\|_2 = \sqrt{< v, v >}$	Euclidean length of the vector v
$\|v\|_1 = \sum_{i=1}^{n} \|v_i\|$	Manhattan length of the vector v
$\text{argmax}_x f(x)$	The x that maximize $f(x)$, usually found by enumerating all x
$\text{argmin}_x f(x)$	The x that minimize $f(x)$, usually found by enumerating all x
$\begin{cases} v_1 & c_1 \\ v_2 & c_2 \\ \dots \end{cases}$	The value of the first satisfied condition, i.e. v_i iff $c_i \wedge \neg c_1 \dots \wedge \neg c_{i-1}$

Appendix B

OpenStreetMap

B.1 File format

To use and extend an OpenStreetMap our system has to be able to read and write osm files, wherefore the OpenStreetMap file format is here described:

An OpenStreetMap basically consist of a set of nodes which each specify a certain geodetic position on Earth and a set of ways that are ordered sequencecs of nodes and connect them together[1]. Each of them can be annotated with certain tags.

Way objects not only specify ways or streets, but everything in the OpenStreetMap that is 1- or 2-dimensional, i.e. buildings are also described by ways which connect all buildings corners.

The osm file is a xml file that just list all these objects as direct children of the xml root node.

Each osm node has a world wide unique id `id`, a latitude `lat`, and a longitude `lat`, and is stored in the osm file like this:

```
<node id="15357699" lat="49.0144314" lon="8.3601013"/>
```

There exist other attributes like author or date, but they are not relevant for our system.

A way also has an id[2], and just refers all connected node ids in order:

```
<way id="52276453">
 <nd ref="665298817"/>
 <nd ref="665298819"/>
 ...
</way>
```

[1] There also exist relations behave like ways, but can connect nodes and ways to provide meta information, like the route of a bus line. However, they are irrelevant for our system, since they provide abstract, invisible information.

[2] A way may have the same id as a node.

Each node and way object can have a semantic tag, like

```
<tag k="highway" v="residential"/>
```

for a small street. These tags are listed as xml child nodes to the xml node of the object.

Building tags that do not describe the fabric of a building are often not assigned to the way object of the building, but to another node which is then placed within the building, but not linked to anything. These single nodes are then rendered as fancy symbols by usual 2D renderers. Our 3D renderer, however, searchs the building that contain this node.

When creating a new osm file, we cannot set the real ids of nodes and ways, because they have to be globally unique. The solution is to use negative ids, which always specify local objects and are replaced by unique, positive ids when the map is uploaded to the OpenStreetMap server.

B.2 Coordinate systems

The coordinates in the OpenStreetMap are in the global geodetic coordinate system, but to process them we need a local Cartesian system. The geodetic coordinate system has been standardized by [NIM] and the conversion consist of a two step conversion from geodetic coordinates into Earth Centered Earth Fixed (ECEF) coordinates which belong a global Cartesian coordinate system and then from ECEF into local ENU coordinates.

Geodetic coordinates consist of latitude ϕ, longitude λ and altitude h.

ECEF coordinates consist of a vector (X,Y,Z) relative to Earth.

ENU coordinates consist of a vector (x,y,z)relative to an reference point at (X_p,Y_p,Z_p) in ECEF .and at ϕ_r,λ_r in geodetic coordinates

To do any conversion one need to know the shape of the Earth, as standardized by WGS84:

semi-major axis	a	$6378137.0m$
Reciprocal of flattening	$1/f$	$298.257223563m$
Semi-minor axis	$b = a(1-f)$	$6356752.3142m$
First Eccentricity Squared	$e^2 = 2f - f^2$	$6.69437999014x10^3$

The conversion formulas[3] are then:

[3] As reported in wikipedia, but confirmed by our experiments

B.2.1 Geodetic -> ECEF

$$\begin{aligned} X &= \left(\frac{a}{\chi} + h\right)\cos\phi\cos\lambda \\ Y &= \left(\frac{a}{\chi} + h\right)\cos\phi\sin\lambda \\ Z &= \left(\frac{a(1-e^2)}{\chi} + h\right)\sin\phi \end{aligned}$$

with $\chi = \sqrt{1 - e^2\sin^2\phi}$,

B.2.2 ECEF -> ENU

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -\sin\lambda_r & \cos\lambda_r & 0 \\ -\sin\phi_r\cos\lambda_r & -\sin\phi_r\sin\lambda_r & \cos\phi_r \\ \cos\phi_r\cos\lambda_r & \cos\phi_r\sin\lambda_r & \sin\phi_r \end{bmatrix} \begin{bmatrix} X - X_r \\ Y - Y_r \\ Z - Z_r \end{bmatrix}$$

B.2.3 ENU -> ECEF

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} -\sin\lambda & -\sin\phi\cos\lambda & \cos\phi\cos\lambda \\ \cos\lambda & -\sin\phi\sin\lambda & \cos\phi\sin\lambda \\ 0 & \cos\phi & \sin\phi \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} X_r \\ Y_r \\ Z_r \end{bmatrix}$$

B.2.4 ECEF -> Geodetic

This conversion is tricky and requires to solve "Bowrings geodetic" equation with the Newton-Raphson algorithm, which results in following iterative equation which will converge to the correct solution for κ:

$$\kappa = 1 + \frac{e^2 a\kappa}{\sqrt{X^2 + Y^2 + (1-e^2)Z^2\kappa^2}} = 0$$

if κ is known, we can calculate ϕ from the definition $\kappa = \frac{\sqrt{X^2+Y^2}}{Z}\tan\phi$ and h from $h = (1 - (1-\kappa^{-1})e^{-2})\sqrt{X^2 + Y^2 + Z^2\kappa^2}$

There exist a closed-form solution – the Ferrari's solution – for an approximation of this equation, but it is not accurate enough for our system.

Calculating λ however is easy, as you can see from the inverse conversion above, because $\tan\lambda = \frac{Y}{X}$

Appendix C

Raw data

```
679 0 660
0 679 187
0 0    1

679 0 660
0 679 187
0 0    1


 1 0 0 0.000000e+00
 0 0.996778878456 -0.0801989243289 0.000000e+00
 0 0.0801989243289 0.996778878456 0.000000e+00

 1 0 0 -5.70738241e+02
 0 0.996778878456 -0.0801989243289 0.000000e+00
 0 0.0801989243289 0.996778878456 0.000000e+00
```

Figure C.1: Calibration file of the Karlsruhe sequence

Bibliography

[BLK10] Olga Barinova, Victor Lempitsky, and Pushmeet Kohli. P.: On detection of multiple object instances using hough transforms, 2010.

[GLU11] Andreas Geiger, Martin Lauer, and Raquel Urtasun. A generative model for 3d urban scene understanding from movable platforms. In *Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, USA, June 2011.

[GRU10] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Asian Conference on Computer Vision*, Queenstown, New Zealand, November 2010.

[GZS11] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *IEEE Intelligent Vehicles Symposium*, Baden-Baden, Germany, June 2011.

[KGL10] Bernd Kitt, Andreas Geiger, and Henning Lategahn. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In *IEEE Intelligent Vehicles Symposium*, San Diego, USA, June 2010.

[LCH08] Keith Yu Kit Leung, Christopher M. Clark, and Jan P. Huissoon. Localization in urban environment by matching ground level video images with an aerial image. In *IEEE International Conference on Robotics and Automation*, Pasadena, USA, May 2008.

[Lei] Bastian Leibe. Lecture computer vision.

[LLS08] Bastian Leibe, Aleš Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *Int. J. Comput. Vision*, 77:259–289, May 2008.

[May79] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. 1979.

[NIM] NIMA. Technical report.

[NNB06] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23:2006, 2006.

[NS07] David Nistér and Henrik Stewénius. A minimal solution to the generalised 3-point pose problem. *J. Math. Imaging Vis.*, 27:67–79, January 2007.

[SMB00] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *Int. J. Comput. Vision*, 37:151–172, June 2000.

[SPF10] Christoph Strecha, Timo Pylvänäinen, and Pascal Fua. Dynamic and scalable large scale image reconstruction. In *CVPR*, pages 406–413, 2010.

[SS06] Katrina Sokolova and Barak Shilo. Experiments in stereo vision. 2006.

[Wik] Wikipedia. Bounding volume.