Peter Riegler-Nurscher, BSc

Tracking Patches of 3D Objects for 2D Augmented Reality Training Algorithms using LSD-SLAM

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme

Telematik

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dr. Vincent Lepetit

Institute for Computer Graphics and Vision

Graz, Austria, Jan. 2016

Abstract

3D Object detection is one of the major challenges in computer vision. Several methods based on object parts have come up in this area over the last years. These methods rely on many training samples of good quality. The following need for a simple and fast way of collecting training samples led to this project.

We present a method for easy gathering of training samples for part based 3D object detecting systems. The method tracks user selected object parts and stores them for later training. It is also capable of handling multiple video sequences for collecting samples under changing environmental conditions. The localization and tracking uses the recently proposed LSD-SLAM system and operates under its given conditions.

Additionally a different part based 3D object detection method is presented and evaluated. It is based on 2D part detection by using the Viola Jones algorithm. The detected parts are used to estimate the 3D position of the trained rigid object.

The implemented system and its resulting performance is compared with the existing object detection approach from [2] which uses Convolutional Neural Networks.

Kurzfassung

3D-Objekterkennung ist eine der größten Herausforderungen in der aktuellen Computer Vision. Viele Methoden basierend auf repräsentativen Teilen der zu erkennenden Objekte wurden in den letzten Jahren präsentiert. Diese Verfahren beruhen auf vielen und vor allem qualitativ hochwertige Trainingsdaten. Die Notwendigkeit einer Möglichkeit zum einfachen und schnellen Sammeln von diesen Trainingsdaten führte zu diesem Projekt.

Wir stellen eine Methode zum einfachen Sammeln von Trainingsbeispielen für 3D-Objekterkennungsysteme basierend auf Objektteilen vor. Das Verfahren lokalisiert und verfolgt von Benutzer ausgewählte Teile eines Objekts und speichert Abbildungen dieser für einen späteren Trainingsvorgang. Das vorgestellte System ist in der Lage verschiedene Videosequenzen, für das Sammeln von Trainingsdaten unter verschiedenen Umweltbedingungen, zu bearbeiten und untereinander zu registrieren. Die Lokalisierung und Verfolgung der einzelnen Objektteile basieren auf dem vor kurzem veröffentlichten LSD-SLAM System und operieren unter dessen Bedingungen.

Zusätzlich wird ein auf Objektteilen basierendes 3D-Objekterkennungssystem vorgestellt und evaluiert. Die 2D-Teileerkennung dieses Systems nutzt den Viola Jones Algorithmus. Die erfassten Teile werden für eine 3D-Positionsbestimmung des erfassten, starren Objekts herangezogen.

Das implementierte System und seine Leistung wurden erfasst und mit dem unter [2] vorgestelltem Objekterkennungsverfahren, welches auf gefalteten neuronalen Netzwerken beruht, verglichen.

Acknowledgments

First and foremost I would like to thank my supervisor, Prof. Vincent Lepetit for his guidance and encouragement, without which this project would not have been possible.

I also want to thank Mahdi Rad for his support and contribution to the evaluation of my work.

I would also like to thank my colleagues and friends at home which brought balance and happiness to my everyday life.

Last but not least I owe my deepest gratitude to my family for their moral support, understanding and love during all times.

Contents

List of Figures

List of Tables

1 Introduction

Contents

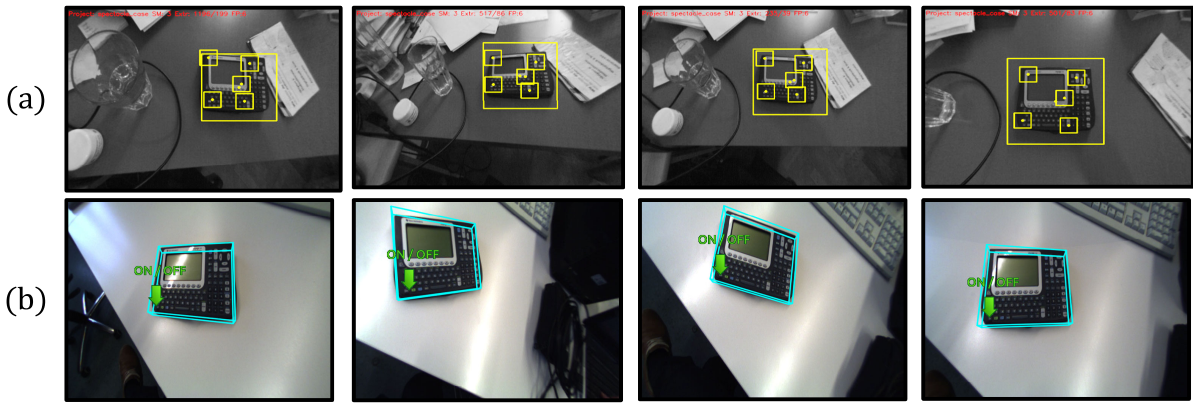


Figure 1.1: (a) The training sequence of an object. The user-selected parts are tracked over time under different views. (b) The test sequence of the trained detector. The pose of the object is augmented by a 3D box and an interesting 2D point is augmented by an arrow.

3D Object detection is one of the major challenges in computer vision. Its many use cases in augmented reality, robot vision and many more and its difficulty makes it one of the most researched areas in this field. 3D Object detection and tracking has undergone significant improvements in recent years. Many approaches are based on edge detection which is sensitive to clutter, while other approaches based on key points are sensitive to occlusions. Still other methods rely on depth sensors which might fail on metallic surfaces or outdoor scenes. Recently strong researched part based models try to cope with these problems as good as possible.

Crivellaro et.al. proposed a method extending a part based approach by virtual control points [2], to which the supervisor of this thesis also contributed. To overcome the issue of occlusion, this approach uses 3D control points around an object part to estimate the pose of it. Just one object part with its surrounding area has to be visible for the pose calculation. It makes this approach very robust to occlusions, efficient and accurate. The method has two detection stages, first the location of the object part is estimated, afterwards the location of the surrounding control points is detected. Both steps use a Convolutional Neural Network (CNN) which is trained using many sample images of the object.

A main challenge for all of these methods is to gather satisfying and enough training samples, to get good detection results. The work in [2] uses ARUCO (see Section 2.1) to register the sample images and to extract patches from them. In ARUCO fiducial markers are placed around the observed object. These markers are detected to calculate the pose of the camera in each frame. After integrating some previous knowledge of the objects structure, the parts can be tracked over time.

The drawbacks of this method are the need for printed physical markers and 3D knowledge of the object parts. Therefore the main purpose of this work is to replace ARUCO by a more convenient system for gathering the sample images.

We developed a method to extract image patches from user selected object parts, where the user selects the parts via a graphical user interface anywhere and at any time within a video stream. The method used for localisation and tracking is the LSD-SLAM system, which was proposed and released under the GPLv3 license lately. Figure 1.1a shows the tracking of the different parts over time. SLAM stands for Simultaneous Localization and Mapping and provides us with localization information and also with a 3D point cloud representation of the object and its environment. The system, which is developed in the course of this thesis, is also capable of using different video sequences showing the same object, these sequences are registered to each other by estimating their coordinate transformations. The training sequences are captured from different views and under different lightning conditions. If the object is movable, one can also use different scenes, in which the object is placed, to get training samples against various backgrounds. Patches of the object part, as well as its control points are taken for the training.

The gained patches can be handed over to the system proposed by Crivellaro et.al. in [2]. Alternatively, the same patches can be used for training the detection method proposed in our work:

Our detection system makes use of the Viola-Jones algorithm proposed in [35]. The Viola-Jones object detection framework was originally designed for 2D face detection, but we try to employ it for 3D object detection. We train one detector for each object part from the gained patches. Due to different background scenes while capturing the training observations, the influence of background is eliminated. Influence of lightning is inherently reduced by the Viola-Jones method. The cascade structure of the used method makes it very fast and even faster if it is tuned for our particular task.

The final step after training the object parts is to test the detectors. Each detector tries to find its part within the image and the system estimates the pose of the rigid object from the position of the parts by solving a Perspective-n-Point problem. To show the detection result, 3D and 2D augmentation is inserted into the scene. Figure 1.1b shows a sequence of augmented images during the testing phase.

1.1 Thesis Outline

- Chapter 2 gives a background on previous proposed works in the area of 3D object detection and tracking.
- In Chapter 3 important parts of the mathematical, algorithmic and software background, which are used in this work, are presented.
- The concepts of the system and subsystems implemented in this work are provided in Chapter 4.
- Important details on the implementation are given in Chapter 5.
- The performance regarding accuracy, robustness and efficiency of different configurations are provided in Chapter 6
- Finally, conclusions of the thesis and future works are presented in Chapter 7.

2

Related Work

Contents

2.1 Tracking Methods

As part of our offline stage we need to track and collect 2D patches of the object which represent the different parts over longer video sequences. The object parts might get strongly transformed due to the camera movements, which is intended to capture many different poses for the training stage. The tracking method on the other hand must be able to deal with these circumstances.

Several differrent approaches have been proposed to track features, patches and other salience points within a video sequence. Figure 2.1 by [32] shows the categories of monocular tracking.

One of the best known is the Kanade-Lucas-Tomasi feature tracker [29, 33]. In this method the general feature detection approach is extended by a spatial component to make the tracking more efficient. The tracker is based on three assumptions, brightness constancy, small motion and spatial coherence. These assumptions show the weaknesses and limitations of the tracker at the same time. The error shown in Equation 2.1 is minimized in the KLT approach.

The correlation measurement for the moving features are the sum of the squared differences. The equation includes a translation or affine model, represented by function f, which tries to model the motion δ_i from the features m_j between two frames.

$$E = \sum_j (\mathbf{I}_t(\mathbf{f}(\mathbf{m}_j; \mathbf{p}_i + \Delta_i)) - T(\mathbf{m}_j))^2 \tag{2.1}$$

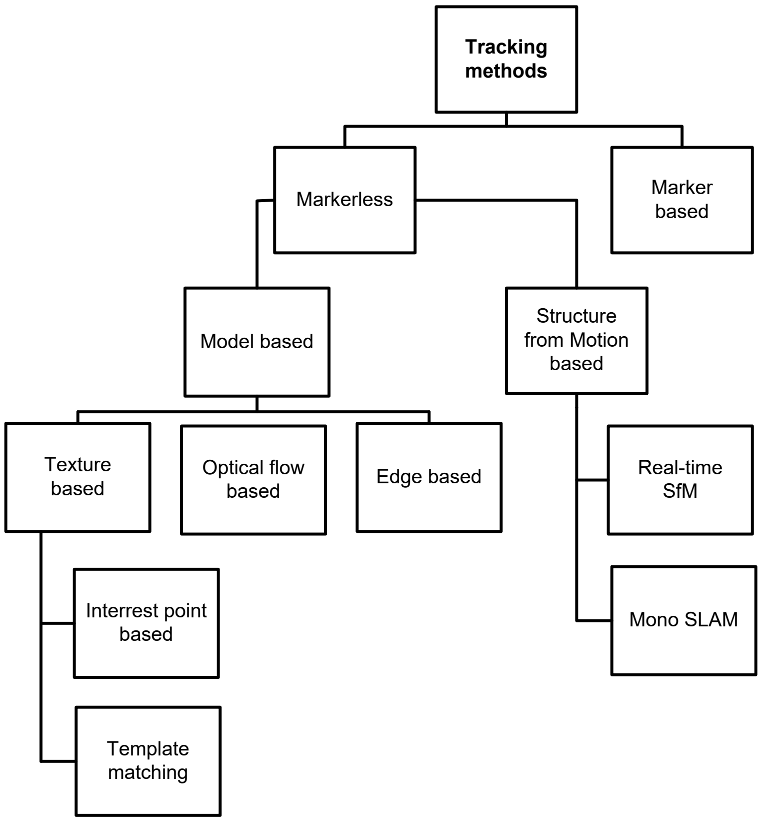


Figure 2.1: Classification of monocular tracking methods in different categories.[32]

Another drawback of the KLT tracker is the potential loss of all feature after a while. In that case the tracking must be initialized again, which might be impossible due to strong affine transformations. Moreover the features can drift over time due to accumulated errors and the tracking positions can get incorrect.

Drifting can be avoided by using fiducial markers. Those are artificial high contrast codes placed within the captured scene. The position of these fiducial markers is known and therefore the camera position can be estimated, they can be detected efficiently using simple thresholding approaches. Figure 2.2 shows different common fiducial marker layouts and their usage in real time environments is visible in Figure 2.3. In recent years there have been proposed fiducial marker systems like [9] which can also cope with partial occlusions of markers.

On the downside, to track parts on the examined object each position relative to the markers must be known, which requires previous knowledge of the physical dimensions of the object (e.g. a 3D model). Of course, the markers must be placed within the physical scene which might be difficult in some situations and might lead to inaccurate tracking results. Due to this issues the currently used ARUCO system should be replaced by an alternative system in the work of [2].

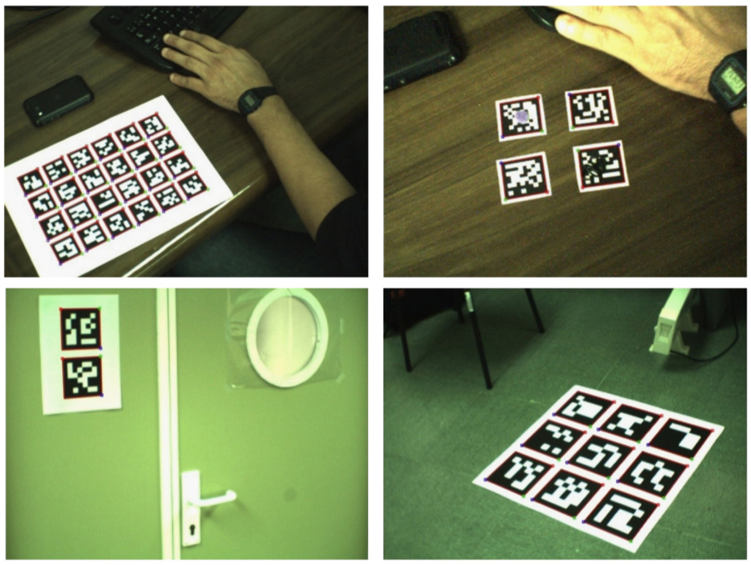


Figure 2.2: Examples of fiducial markers in different environments.[9]

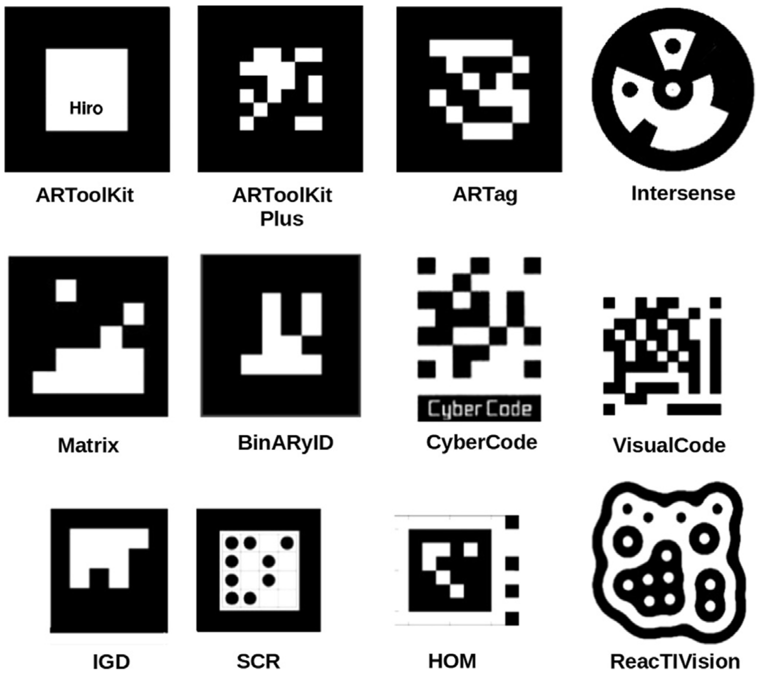


Figure 2.3: Examples of fiducial markers proposed in previous works.[9]

SLAM-Systems [5, 13, 23] provide a method to track points without 3D knowledge of the object or any physical markers. These systems track features or whole pixel maps and simultaneously calculate the current position of the camera using a single RGB camera. The SLAM approach is based on the structure from motion principle and therefore requires static scenes.

The proposed Parallel Tracking and Mapping (PTAM) [13] approach, tries to find salient features within the frames and tracks their position over time. It introduced a method to reduce the DOF per step by separating the camera pose calculation from the feature point tracking. Both processes run in parallel while the camera pose must be calculated fast and robust to get a good tracking experience, the mapping of the features must be rich and accurate.

In our case, we need a more dense tracking of features, because it has to be possible for the user to select any point within the image. Newcombe et al. proposed a dense tracking and mapping system (DTAM) [23] which overcomes the limitations of sparse feature points. Their system tracks the camera position with its 6 DOF and creates a full dense depth map. The composition of the depth map can be seen in Figure 2.4. In general the system tries to reduce the photometric cost volume C_r for one keyframe based on an reference input image I_r with a camera pose T_{rw}. The inverse depth d ranges between xi_{min} and xi_{max}. Ten to hundreds of video frames indexed as $m \in I(r)$ are used to calculate the values for one cost volume thus also for one keyframe.

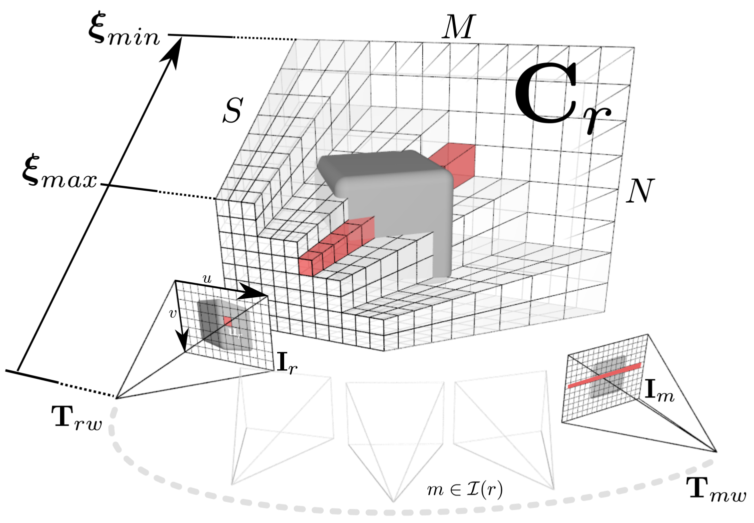


Figure 2.4: A keyframe r consists of a reference image I_r with pose $\mathbf{T}_{rw}$ and data cost volume $\mathbf{C}_r$. Each pixel of the reference frame $\mathbf{u}_r$ has an associated row of entries $\mathbf{C}_r(\mathbf{u})$ (shown in red) that store the average photometric error or cost $\mathbf{C}_r(\mathbf{u}; d)$ computed for each inverse depth $d \in D$ in the inverse depth range $D = [\xi_{min}; \xi_{max}]$. [23]

Formula 2.2 shows the calculation of a cost volume C_r which equals the average over the photometric error 2.3 from the overlapping images in $I(r)$. The photometric error is calculated between the reference image of the keyframe I_r and an overlapping frame I_m. A pixel u and a corresponding depth value are used in the function π to back-project to a 3D point in the coordinate system of frame I_m. This 3D point for its part is projected into the image I_m to get a gray-value for comparison with $I_r(u)$.

$$\mathbf{C}_r(u,d) = \frac{1}{|\mathcal{I}(r)|} \sum_{m \in \mathcal{I}(r)} \|\rho_r(\mathbf{I}_m, \mathbf{u}, d)\|_1 \tag{2.2}$$

$$\rho_r(\mathbf{I}_m, u, d) = I_r(u) - I_m(\pi(KT_{mr}\pi^{-1}(u,d))) \tag{2.3}$$

The resulting depth values are filtered using total variation optimization to get natural smoothed surfaces. These basic concepts are also used in the state of the art method LSD-SLAM [5] proposed in 2014. It provides a semi-dense depth map and tracking in real time. The system is capable of large-scale scenes and corrects accumulated drifts. Due to its capabilities we decided to use LSD-SLAM for our tracking system. In Section 3.3 we give an introduction into LSD-SLAM and its utilization.

2.2 Object Detection Methods

Over the recent years many different Object Detection methods have been proposed and further advanced. In this section we discuss some representative works and further focus on part based methods.

One of the most researched approaches relies on edges, see [11, 14, 21], but they are sensitive to large occlusions and clutter. Other approaches are based on keypoints, like [31, 34, 37], which are easier to match than edges, on the down side the use of keypoints requires textured objects. Some methods using stereo information, like [24], have also been proposed, but the requirement of a stereo configuration limits the possible use cases.

Methods based on regions have also been proposed, they join a 2D segmentation with the 3D pose estimation problem, see [26, 27]. However partial occlusion are hard to handle with this approaches.

In recent times, inexpensive 3D sensors, like the Kinect sensor, have entered the market, they reveal many new approaches, like in [4, 15, 30], exploiting the resulting RGB-D information. Our work focuses only on monocular images, which excludes these sensors.

Recently proposed part based models are based on using object parts and also their relation among them. Many part based methods are inspired by the work of Feizenszwalb et al. and the proposed Deformable Part Model [6]. These techniques focus mostly on category rather than instance recognition. The Deformable Part Model (DPM) uses not just local part informations of an object, but also its spatial arrangement. In principle the approach is based on the Dalal-Triggs detector which uses a single filter on histogram of oriented gradients (HOG) features. In addition to a coarse root filtering (Dalal-Triggs), a fine filtering on higher resolution and for each part individually, is performed. The definition of the parts is based on an unsupervised manner and uses heuristics such as high gradient energy. Figure 2.5 shows a single component person model with five parts.

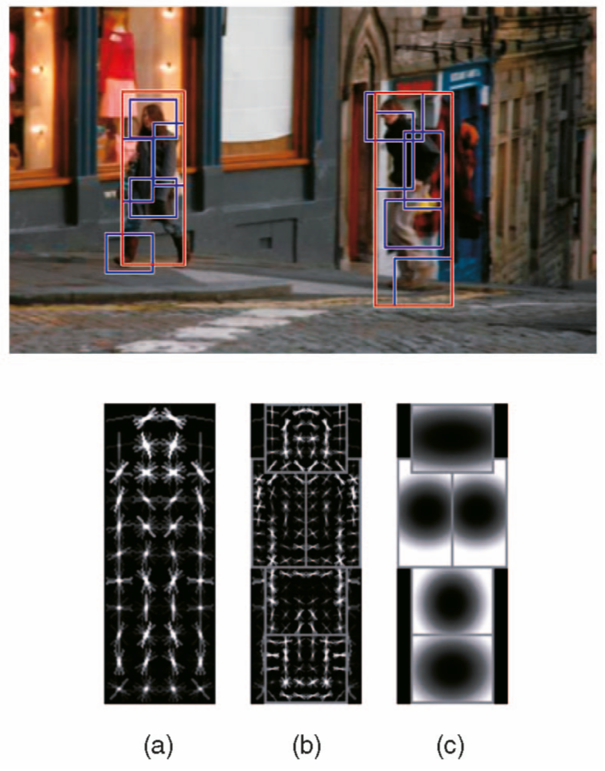


Figure 2.5: Part Based Model example: Single component person model, defined by (a) a coarse root filter, (b) several higher resolution part filters, and (c) a spatial model for the location of each part relative to the root.[6]

Another part-based methods try to outperform DPM by using not only the deformable parts model but also its underlying 3D geometry [25, 30]. Moreover the proposed geometry-driven deformable part-based models inherently allow to infer the 3D properties from a single 2D image.

To get 3D information about the object's geometry the system cited in [30] is trained with labeled RGBD images and follows a supervised manner, while the approach in [25] uses CAD models to incorporate the geometry information. Payet and Todorovic proposed a method based on image contours as basic feature and on higher level bag of boundaries (BOB) to detect 3D objects [24]. By using contours instead of feature points there is no need for interior textured objects as in many part-based methods required.

Lim et al. proposed a fine pose parts-based model (FPM) [20], also using CAD models for fine pose estimations. The method reaches state-of-the-art in terms of accuracy by increasing the number of training samples. The higher number of samples is achieved by rendering many different views of the object thereby the number of real images can be very low. Xiang et al. try to handle significant viewpoint changes in their recently proposed work [37]. Thier improvement of the DPM is based on modeling the 3D aspect parts and their relationship. So it is possible to predict the visibility and shape of these parts in any viewpoint under different cases of self occlusion.

3 Mathematical, Algorithmic and Software Background

Contents

3.1 Accurate Part based 3D Object Detection using Virtual Control Points

The 3D object detection method proposed in our initial paper [2] is based on control points. These virtual points are arranged around a salient point in 3D space, which are trained by a 2D patch detection method. In the detection phase the main parts are searched and in a second stage the virtual control points, as trained before, are detected in the surrounding area. If just one part is visible, the pose of the object can be estimated from the control points by solving a Perspective-n-Point problem. That makes this method very robust to occlusions and also very efficient.

The training methods proposed in [2] for the detection of parts and also for detection of the control points are convolutional neural networks (CNN). The orientation and location of these points can be chosen randomly in general, but in practice an alignment along the coordinate axes is a good choice. In Figure 3.1 an exemplary arrangement of control points around the main point is shown.

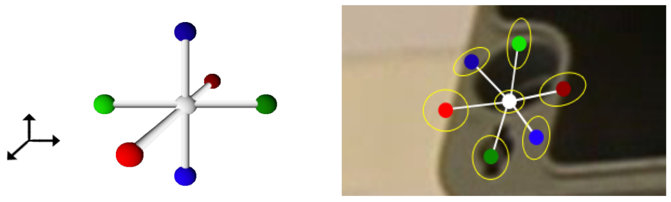


Figure 3.1: The formation of virtual control points around an part in 3 orthogonal directions.[2]

3.1.1 Part Training and Detection

The parts within the image are detected by a Convolutional Neural Network (CNN) [16]. It is trained by the collected patches of the parts which should be 32x32 pixel. The patches are selected randomly around the center point, therefore our tracking system must extract 64x64 pixel sized patches to include all possible patches. The architecture of the CNN is visible in Figure 3.2.

The training itself is done by optimizing the negative log-likelihood over the parameters w of the CNN:

$$\widehat{w} = \arg\min \sum_{j=0}^{N_P} \sum_{\mathbf{q} \in \tau_j} -\log \mathrm{softmax}(\mathrm{CNN}_w^{\text{part-det}}(\mathbf{q}))[j] \tag{3.1}$$

where $\mathbf{q}$ is an image patch, N_P the number of parts and τ_j a training set made of image patches centered on part j.

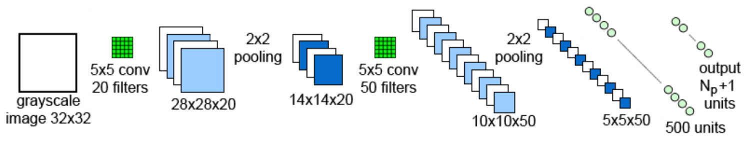


Figure 3.2: Architecture of CNN $^{\text{part-det}}$ for part detection. The last layer outputs the likelihoods of the patch to correspond to each part or to the background.[2]

At run-time the CNN delivers clusters of large values around the center point of a part. A gaussian smoothing is applied to filter the output and the local maximums of the clusters are candidates for the location of parts.

3.1.2 Control point Training and Detection

After the detection of parts, the control points are evaluated. The training of the control points is done again by a CNN, but with another architecture (see in Figure 3.3). The size of the training patches is 64x64 and they should include the control points. The output layer of the network is made of $2N_V$ neurons, which predict the 2D locations of the control points. N_V stands for the number of control points of the part, which is usually 7.

For each part a CNN is trained by minimizing the squared loss of the predictions over the parameter w:

$$\widehat{w} = \arg\min \sum_{(\mathbf{q},\mathbf{w})\in\nu_j} \left|\left|\mathbf{w} - \mathrm{CNN}_w^{\text{cp-pred-}j}(\mathbf{q})\right|\right|^2 \tag{3.2}$$

where ν_j is a training set of image patches centered on part j and the 2D locations of the control points are concatenated in a vector $\mathbf{w}$.

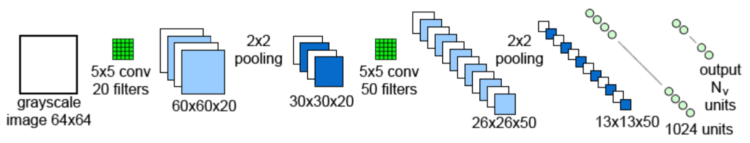


Figure 3.3: Architecture of a CNN $\mathrm{CNN}^{\text{cp-pred-j}}$ for predicting the projections of the control points.[2]

At run-time the predictions $\{\hat{\mathbf{v}}_{jkl}\}$ for the control points around each part j are estimated. The 2D uncertainty for these predictions is calculated by propagating the image noise through the CNN.

Each control point provides a 3D-2D correspondence, so estimating the object pose becomes a standard 3D pose estimation problem. Also a pose prior is used in form of a Mixture-of-Gaussians. Details on the pose estimation process can be looked up in [2].

3.2 Camera Model and Camera Calibration

Camera calibration is a task to get the geometrics of a camera system, it's important to get correct and accurate results for both, the training and also the testing steps. In this section we give a brief introduction into camera calibration.

The intrinsic calibration information consists usually of a calibration matrix K and distortion coefficients. To get these information some real word, reference points must be captured by the camera, usually a classical black-white chessboard, from different poses, like the fiducial markers described in Section 2.1. Also 3D reference patterns for more accuracy and sometimes 1D patterns are used. Then a homography between the real object and the image is estimated, delivering the calibration information. The checkerboard captured by the camera and already evaluated is visible in Figure 3.4.

The relation between the points on the real plane and the points on the captured images can be described by a matrix $P = K[R|t]$ (3x4 matrix) where R (3x3 matrix) and t (3x1 vector) are the rotation and the translation of the camera. The values of the matrix K stay the same over time for a specific camera-lense system and called intrinsic parameters. The composition of the desired matrix K is shown in equation 3.3. The

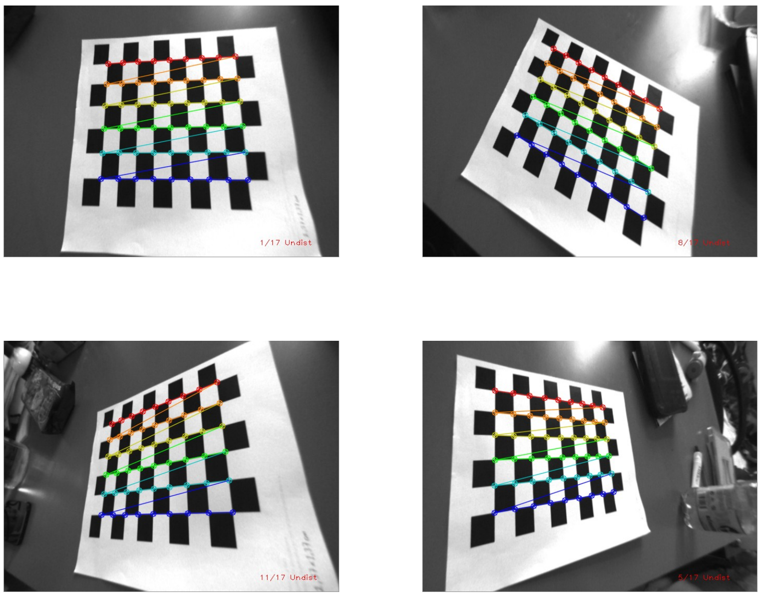


Figure 3.4: Checkerboard calibration pattern from different views.

values f_x and f_y are the focal lengths in x and y direction, c_x and c_y define the location of the principal point within an image.

The matrix P consists of 6 extrinsic (3 DOF for rotation R and 3 DOF for translation t) and 5 intrinsic parameters, 11 parameter in total, therefore at least 6 correspondences are needed for its calculation.

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.3}$$

The set of corresponding points on the reference checkerboard and on the captured images are used to compute P (e.g. by solving the linear system) where $P = [KR|KRt] = [M|Mt]$. Once the matrix P is estimated a RQ decomposition breaks M, the left 3x3 matrix of P, into two 3x3 matrices $M = AB$. A is an upper triangular and B is an orthogonal matrix (i.e. $B^T B = I$), where A corresponds to the matrix K and B to the rotation matrix R. The translation t can then be easily computed by $t = M_4^{-1} \cdot P_4$, P_4 refers to the the last column of the matrix P. Figure 3.5 shows the pinhole camera model and its parameters described in the previous paragraph.

Another elegant way to estimate K is by estimating the image of the absolute conic Ω. The absolute conic is invariant to rigid transformations and bypasses the estimation of the full homography P with its 11 DOF. Therefore it is possible to directly calculate the intrinsic parameters. For more details, the reader is referred to [12, 38].

If distortion due to imperfect lenses occur the Brown-Conrady model is used. Equa-

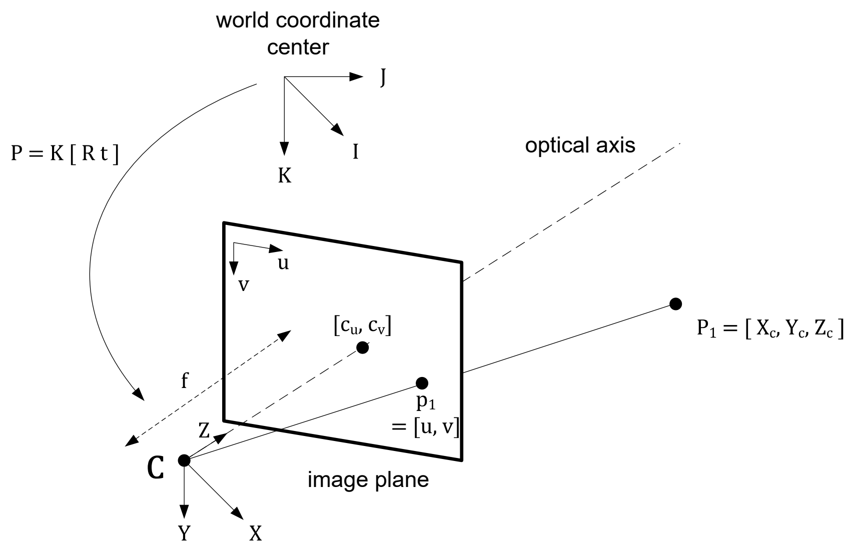


Figure 3.5: Pinhole camera model of geometric camera calibration: The matrix P describes the relation between the 3D world model and the 2D image coordinates. The camera center is donated as **C** and its distance to the image plane is called focal length f. The optical axis pierces through the image plane at the principal point $[c_u, c_v]$

tions 3.4 and 3.5 model the radial distortion using three coefficients k_1, k_2 and k_3. The effects of radial distortion, also referred to as barrel and pincushion distortion, are shown in Figure 3.6. Tangential distortion occurs when the lense and the sensor are not parallel, it can be resolved using equations 3.6 and 3.7. The number of parameters is limited to three for each distortion as in the model used by OpenCV.

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.4}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.5}$$

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.6}$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.7}$$

Sometimes it is not possible to use reference patterns, e.g. footage recorded in the past with no knowledge about the camera metrics, in that case self-calibration is used. Calibration is done by just moving the camera, resulting in more parameters, a much harder mathematical problem and often less accuracy.

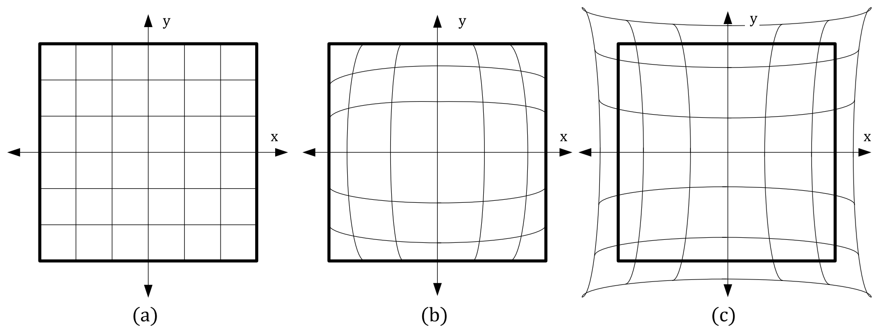


Figure 3.6: Radial distortions: (a) no distortion, (b) barrel and (c) Pincushion distortion

3.3 Large-Scale Direct Monocular SLAM

Large-Scale Direct Monocular SLAM (LSD-SLAM) is a SLAM System that uses direct image alignment coupled with filtering-based estimation of semi-dense depth maps. In this work LSD-SLAM replaces ARUCO as marker-less tracking system for the object parts.

The system is designed to cope with large scenes with large scale changes, drifting and loop closures. Figure 3.7 shows an example scene of a building including its generated depth maps and the resulting point cloud. The context in this section is based on the paper [5].

The global map is represented by keyframes connected by similarity transformations. The system incorporates scale changes of the environment and corrects accumulated drift. As mentioned in Section 2.1 we use this SLAM-System for tracking of selected object parts.

The algorithm consists of three components: tracking, depth map estimation and map optimization as shown in Figure 3.8.

The tracking component estimates the current rigid body pose $x_i \in se(3)$ with respect to the previously selected keyframe. The preceding frame is used for an initial guess of the pose. To refine or replace the current keyframe, the depth map estimation uses the set of tracked frames. The depth map is refined by many per-pixel small-baseline stereo comparisons. If the pose changes are too big, a new keyframe is introduced.

When the current frame is replaced by a new one, it is embedded into the global map by the map optimization part. This component also tries to find loop closures and estimates scale-drifts by calculating a similarity transformation to existing keyframes.

The global map, a pose graph of keyframes, stores all gathered information. Whereby a keyframe consists of a camera image I_i, an inverse depth map D_i and the variance of that depth map V_i. As mentioned before the semi-dense depth map and therefore also the variance map include only values on image points with large gradient magnitudes.

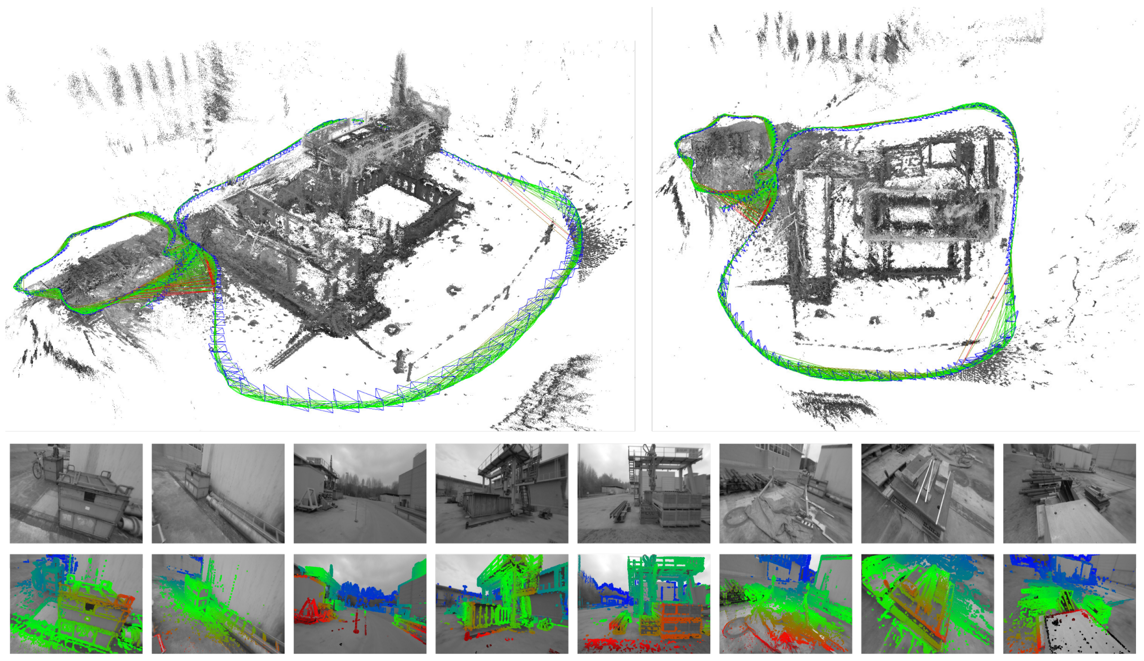


Figure 3.7: LSD-SLAM example. Top: Accumulated point cloud of all key frames. Bottom: Keyframes with color-coded semi-dense inverse depth maps.[5]

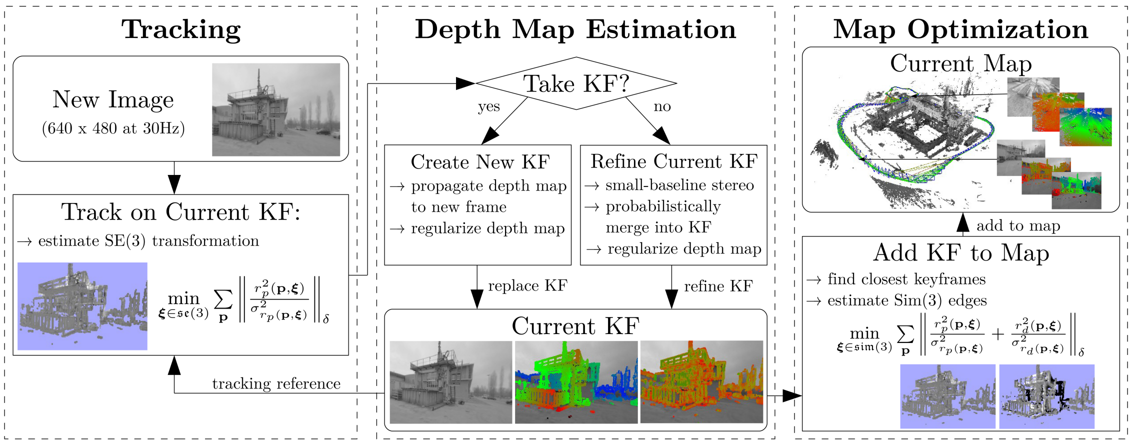

Tracking
New Image
(640 x 480 at 30Hz)
Track on Current KF:
→ estimate SE(3) transformation
$\min_{\xi \in \mathfrak{se}(3)} \sum_{\mathbf{p}} \left\| \frac{r_p^2(\mathbf{p},\xi)}{\sigma^2_{r_p(\mathbf{p},\xi)}} \right\|_\delta$
tracking reference
Depth Map Estimation
Take KF?
yes
no
Create New KF
→ propagate depth map to new frame
→ regularize depth map
Refine Current KF
→ small-baseline stereo
→ probabilistically merge into KF
→ regularize depth map
replace KF
refine KF
Current KF
Map Optimization
Current Map
add to map
Add KF to Map
→ find closest keyframes
→ estimate Sim(3) edges
$\min_{\xi \in \mathfrak{sim}(3)} \sum_{\mathbf{p}} \left\| \frac{r_p^2(\mathbf{p},\xi)}{\sigma^2_{r_p(\mathbf{p},\xi)}} + \frac{r_d^2(\mathbf{p},\xi)}{\sigma^2_{r_d(\mathbf{p},\xi)}} \right\|_\delta$

Figure 3.8: Overview over the LSD-SLAM algorithm.[5]

3.3.1 Tracking Frames

For the actual tracking of a new frame I_j, starting from a keyframe K_i, the photometric error is minimized to get the pose ξ_{ji} between these two frames. The calculation of the variance-normalized photometric error is illustrated in equation 3.8 (including 3.9 and 3.10) where $||.||_\delta$ designates the huber norm. The image intensity noise is assumed to be Gaussian σ_i^2. An image point p in 3D world coordinates and its inverse depth $D_i(p)$ are projected into the new image using the ω -function. The minimization of the energy E_p is accomplished using iteratively reweighted Gauss-Newton optimization.

$$E_p(\xi_{ij}) = \sum_{p \in \Omega_{D_i}} \left\| \frac{r_p^2(\mathbf{p}, \xi_{ij})}{\sigma^2_{r_p(\mathbf{p},\xi_{ij})}} \right\|_\delta \tag{3.8}$$

$$\text{with } r_p(\mathbf{p}, \xi_{ij}) = I_i(\mathbf{p}) - I_j(\omega(\mathbf{p}, D_i(\mathbf{p}), \xi_{ij})) \tag{3.9}$$

$$\sigma^2_{r_p(\mathbf{p},\xi_{ij})} = 2\sigma_I^2 + \left(\frac{\partial r_p(\mathbf{p}, \xi_{ij})}{\partial D_i(\mathbf{p})} \right)^2 V_i(\mathbf{p}) \tag{3.10}$$

3.3.2 Depth Map Estimation

A new keyframe is created when the camera leaves the area of the existing map. $dist(\xi_{ji})$, shown in equation 3.11, is calculated from the pose ξ_{ji} relative to the keyframe and thresholded, where W is a diagonal matrix containing the weights.

$$dist(\xi_{ij}) = \xi_{ij}^T W \xi_{ij} \tag{3.11}$$

Once the threshold is exceeded, the current frame becomes the new keyframe and the points are projected back from the previous one. After this initialization outliers are removed, and the depth map is scaled to a mean inverse depth of one. The new created keyframe is now used for further tracking of the frames.

All frames which don't become keyframes are used for refinement of the depth map. Many efficient small baseline stereo comparisons are performed leading to more accurate depth values and additional new pixels.

Monocular SLAM is inherently scale-ambivalent, which means the absolute scale of the world is not determinable. Unknown scale leads to scale-drift, one major source of errors. To avoid this issue, LSD-SLAM scales all keyframes to a mean inverse depth of one and a similarity transformation between the edges of the keyframes is estimated. For this problem Engel et al. proposed a method called direct, scale-drift aware image alignment on $sim(3)$. In addition to the photometric residual r_p, a depth residual r_d is incorporated, penalizing deviations in inverse depth between keyframes, resulting in a new error function to minimize (see equation 3.12).

$$E_p(\xi_{ij}) = \sum_{p \in \Omega_{D_i}} \left\| \frac{r_p^2(\mathbf{p}, \xi_{ij})}{\sigma^2_{r_p(\mathbf{p},\xi_{ij})}} + \frac{r_d^2(\mathbf{p}, \xi_{ij})}{\sigma^2_{r_d(\mathbf{p},\xi_{ij})}} \right\|_\delta \tag{3.12}$$

The system also searches for loop closures to improve the performance additionally. If a loop closure is detected, the accumulated drifting from previous keyframes will be corrected.

3.3.3 Map Optimization

The gathered depth information is continuously optimized using pose graph optimization. The error function to minimize is shown in equation 3.13. W defines the world frame.

$$E(\xi_{W_1} ... \xi_{W_n}) = \sum_{(\xi_{ij}, \Sigma_{ij}) \in \mathcal{E}} (\xi_{ij} \circ \xi_{W_i}^{-1} \circ \xi_{W_j})^T \Sigma_{ij}^{-1} (\xi_{ij} \circ \xi_{W_i}^{-1} \circ \xi_{W_j}) \tag{3.13}$$

3.4 Robot Operating System

The robot operating system [1, 28] is not an operating system in the traditional sense of process scheduling and management, it is rather a set of libraries, tools and conventions simplifying development of robot and robot related software on various platforms. It was developed to provide standard tools needed in robotics and simplifying the development processes by providing standard interfaces and by delivering drivers for hardware. ROS is licenced under the standard three-clause BSD license[1] and offers interfaces for C++ and Python. The supported platform is Ubuntu and there is experimental stage under OS X Android (NDK) and Gentoo Linux.

It is used by the LSD-SLAM system to gather camera images and transmit estimated tracking and mapping information to a client system. Therefore the whole LSD-SLAM core and client components run in a ROS environment providing a high level platform by managing its communication.

The fundamental concepts of the ROS implementation are nodes, messages, topics, and services [28]. A system implemented using ROS usually consists of several nodes, e.g. the LSD-SLAM core, these nodes communicate by exchanging messages. The message passing is based on a publisher-subscriber approach, where a publisher can feed several subscribers with its messages. The messages are defined by the message IDL (Interface Description Language) and sent out over a topic, e.g. `camera_image`, which defines the used channel. There can be several subscribers and also several publishers on the same topic, while publishers and subscribers in general are not aware of each others existence. The whole message passing system is anonymous and asynchronous, therefore it is easy to record data and afterwards play it back over different channels. This offers high flexibility

[1]http://opensource.org/licenses/BSD-3-Clause
(accessed: Saturday 9th January, 2016)

rosmake	This tool is used to build ROS packages in correct order of their dependencies.
rosrun	Rosrun allows to run an executable in an arbitrary package.
roslaunch	Starts ROS nodes locally and remotely via SSH.
roscore	This collection of nodes and programs are used as basis for other ROS nodes to communicate.
rosnode	Can be used to display debugging information about nodes like publications subscriptions and connections. • rosnode ping Tests the connectivity to a node. • rosnode list Shows a list of active nodes. • rosnode info Prints information about a node. • rosnode kill Kills a running node.
rxgraph	Plots a graph of ROS nodes that are currently running as well as ROS topics connecting them. Figure 5.1 shows a graph of our running system.

Table 3.1: ROS commands

during debugging and also during service. ROS services are used to realise synchronous transactions if needed. Other libraries provided by ROS and used in our system are mentioned in Section 5.1.1.

Table 3.1 presents some of the main commands in ROS needed to run and debug our system.

3.5 Viola-Jones and Local Binary Patterns

The Viola-Jones algorithm is a rapid object detection method motivated by the task of face detection.

In this thesis, the Viola-Jones framework is used for the training and detection of the object parts as an alternative to the proposed CNN in the paper [2].

This method allows to train a classifier for our tracked object parts and reduces the influence of background by using different training environments. It also reduces the influence of illumination and pose. Additionally, it can cope with many variations across individuals. The last aspect is not so important for us, because the parts learned are not changing their appearance.

In Figure 3.9 the structure of the Viola-Jones approach is visible. It shows the different levels the algorithm goes through and one can conclude that the inner parts are more often executed than the outer parts. This results in many feature comparisons on the lowest level, which have to be executed very fast.

Three main concepts are combined in this approach, the image representation called integral image, a classifier using AdaBoost and a method to combine complex classifiers in a cascade structure. The content in this section is based on the publications [35] and [18].

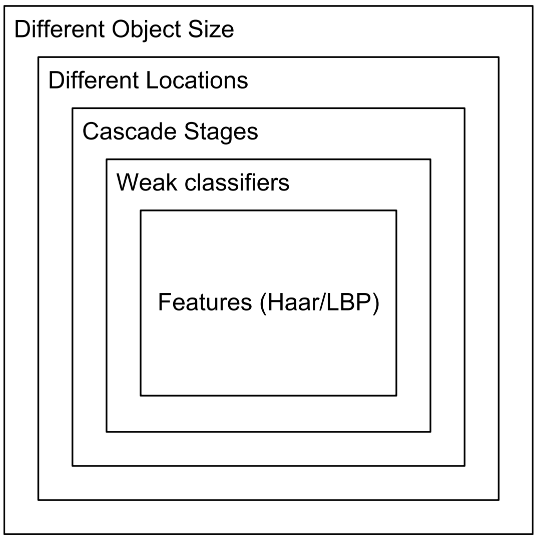


Figure 3.9: Overview over the layers of the Viola-Jones algorithm.[5]

3.5.1 Features

The features introduced by P. Viola and M. Jones are inspired by Haar basis functions. These Haar features are very efficiently calculated, the value of one feature is the difference between the sum over the pixels within the rectangular regions. In Figure 3.10 four features are visible, the difference between the sum over the dark regions and the sum over the white regions is estimated. The regions have the same size and shape and are small compared to the image size.

For efficient calculation of the sums within the rectangles a representation, called integral image, is introduced. Equation 3.14 shows the formula for calculating the integral image from an input image.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3.14}$$

The integral image can be computed in one pass over the image by additionally storing

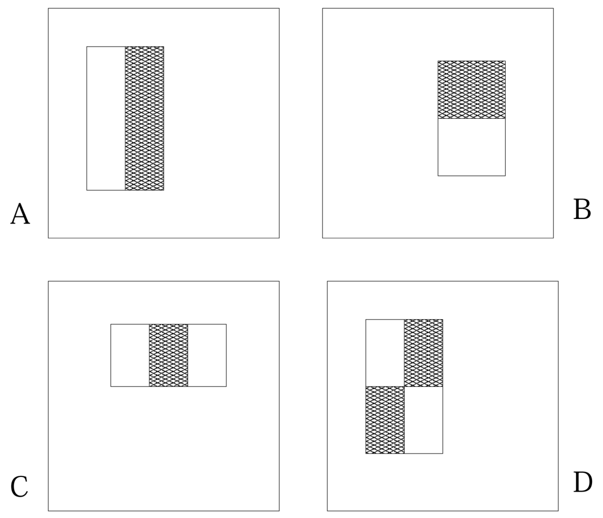


Figure 3.10: Rectangle HAAR features within the detection window. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.[35]

cumulative row sums $s(x, y)$ as shown in equations 3.15 and 3.16, with the initial values of $s(x, -1) = 0$ and $ii(-1, y) = 0$.

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{3.15}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{3.16}$$

The sum of the pixels within one rectangle can then be easily calculated by four reference points. For instance the sum for rectangle D visible in Figure 3.11 can be computed as $4 + 1 - (2 + 3)$. The value at location 2 for example includes the sum over the regions A and B.

3.5.2 Learning Classificator

Any machine learning approach can be used to learn a classification function using these features. The Viola-Jones method uses AdaBoost to choose a set of features and to train the actual classifiers. AdaBoost is used to boost the performance of a simple learning algorithm by combining several instances of it. Over 180000 features are associated with each image sub-window. This meta-algorithm is now used to select the most significant features of the huge set of rectangle features. These simple but significant features are combined to one classifier. The most significant features separate positive and negative samples the best.

The algorithm proposed by Viola-Jones, shown afterwards, discards a vast majority of

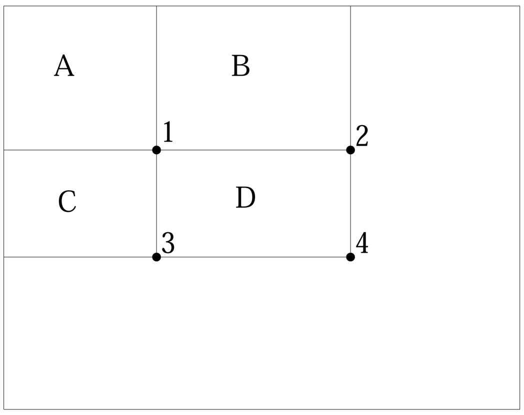


Figure 3.11: Concept of integral images.[35]

features to get an efficient classifier, by still delivering good detection performance.

- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- Given example images $(x_1, y_1), ..., (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- For $t = 1, ..., T$:
    1. Normalize the weights,
    $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
    2. For each feature, j, train a classifier h_j which is restriced to using a single feature. The error is evaluated with respect to $w_t, \epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
    3. Choose the classifier, h_t, with the lowest error ϵ_t.
    4. Update the weights:
    $$w_{t+1,i} = w_{t,i} \beta_t^{1-\epsilon_i}$$
    where $\epsilon_i = 0$ if example x_i is classified correctly, $\epsilon_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = log\frac{1}{\beta_t}$

The OpenCV implementation of the Viola-Jones algorithm offers four different variants of the AdaBoost algorithm: Discrete AdaBoost, Real AdaBoost, LogitBoost and Gentle AdaBoost. The variants differ in the weighting of the samples and the formation of the final classifier.

- **Discrete AdaBoost** is visible in the listing example above.
- In **Real AdaBoost** the error function is replaced by a probability value $p(x_i)$. $p(x_i) = P(y_i = 1|x_i)$ and is chosen by calculating the weighted least squares error.
- **LogitBoost** is an adaptation based on logistic regression.
- **Gentle AdaBoost** fits an regression function by weighted least-squared of y_i to x_i. It is chosen by minimizing $\sum_i w_{t,i}(y_i - h_t(x_i))^2$

The first selected features for example at face detection are obviously in the eye and eyebrow region, as visible in Figure 3.12. At this regions the shown features are very significant in every face sample image.

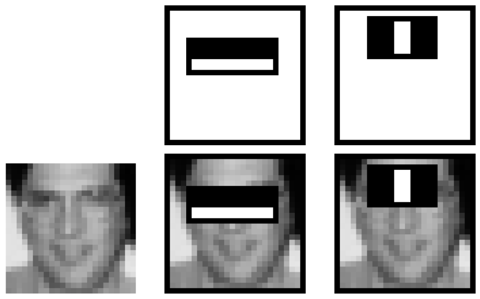


Figure 3.12: The first and second features selected by AdaBoost.[35]

3.5.3 The Cascade

To reduce the computation time at the testing phase, a cascade of different classifiers is built. This cascade starts with simple classifiers to reject big regions of the image followed by increasingly complex classifiers for the remaining finer regions. The resulting cascade (a degenerated decision tree) is visible in Figure 3.13. A negative outcome at any point within the cascade leads to a rejection of the evaluated sub-window. At training of the cascade there is a trade-off between the number of stages, the number of features within each stage and its threshold. For simplification each stage is trained by adding features until the required detection and false positive rate is reached. If the overall detection rate and false positive rate are reached no more stages are added.

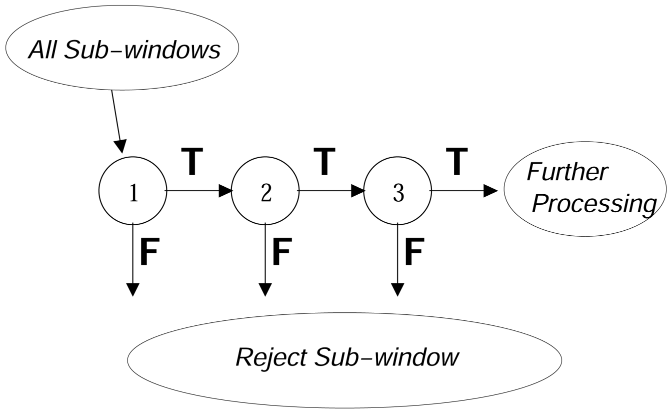


Figure 3.13: Viola-Jones detection cascade: At every decision node a classifier is executed, if the outcome is positive the branch 'T' is taken and the next classifier is executed. If the outcome is negative the checked sub-window is rejected and the branch 'F' is taken. [35]

3.5.4 LPB

Further improvement of the Viola-Jones method was done by Liao et al. Their work uses multi-scale block local binary patterns (MB-LBP) instead of the HAAR features. Training and detection with LBP are several times faster than with HAAR features. The standard Local Binary Patterns (LBP) operator, visible in Figure 3.14 (a), labels each pixel of an image by thresholding its 3 x 3 neighborhood with itself resulting in a binary string. Figure 3.14 (b) shows the extended MB-LBP. The comparison between single pixels is simply replaced with comparison between the average values of squared subregions resulting in filters of size s. For example, the original LBP have a size of $s = 3$, resulting in a filter size of 3 x 3.

Results of the LBP filter for different scales s in the example of faces are shown in Figure 3.15.

The performance and results of the Viola-Jones and LBP approach in our scenario are presented in Section 6.

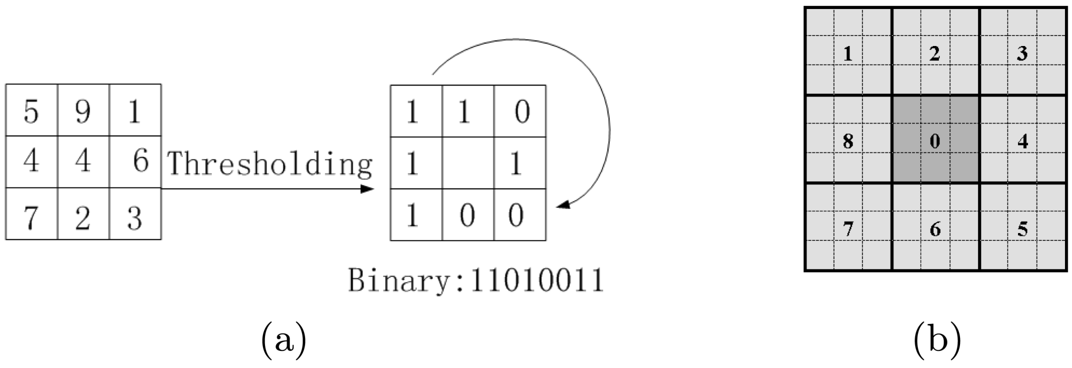


Figure 3.14: Local Binary Patterns concept: (a) Basic LBP operator (b) 9x9 MB-LBP operator. In each sub-region, average sum of image intensity is computed. These average sums are then thresholded by that of the center block. [18]

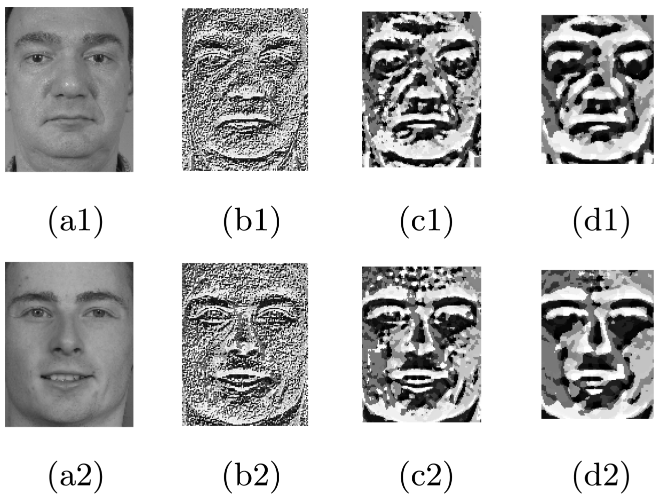


Figure 3.15: MB-LBP filtered images of two different faces. (a) original images; (b) filtered by 3 x 3 MB-LBP; (c) filtered by 9 x 9 MB-LBP; (d) filtered by 15 x 15 MB-LBP. [18]

3.6 Perspective-n-Point Problem

The Perspective-n-Point camera pose detection, or PnP-Problem, tries to find the rotation R and translation t of a camera by using 2D/3D point correspondences. For the PnP-Problem, the intrinsic parameters, contained in the matrix K, are assumed to be known.

The 3D rotation is defined by 3 DOF, the same as the translation with its 3 DOF. This leads to 6 DOF for the extrinsic camera pose and corresponding 6 parameters. Therefore, in general three point correspondences are enough for solving the system, leading to a P3P-Problem.

However using just three points for the computation leads to four possible results, where just two results are relevant. To remove this ambiguity a fourth correspondence has to be added. Furthermore, to increase the accuracy of the result more correspondences (n correspondences) can be added by using different algorithms for solving a PnP-problem.

3.6.1 P3P-Problem

In this section we present a method to solve the P3P-Problem using the distant-based definition [36] in closed form. Figure 3.16 shows the configuration of the P3P-Problem with its camera center C and the point correspondences between the 2D projections m_1, m_2 and m_3 of the 3D points M_1, M_2 and M_3.

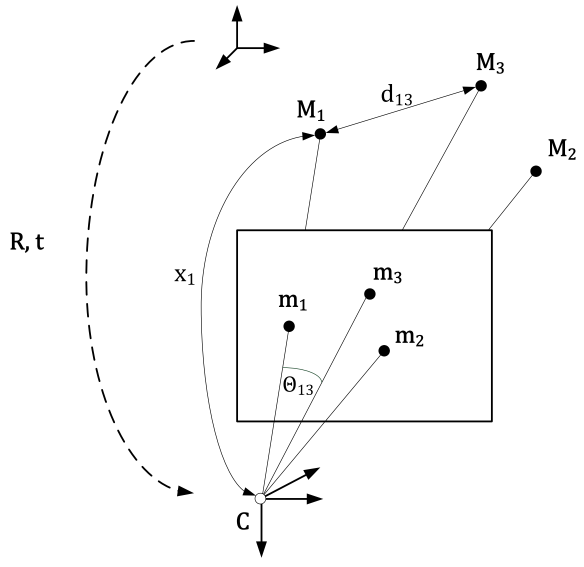


Figure 3.16: P3P-Problem between the 3D Points M_1, M_2 and M_3 and their 2D projections on the image plane m_1, m_2 and m_3. The resulting rotation matrix $\mathbf{R}$ and the translation $\mathbf{t}$ represent a transformation from 2D points to their corresponding 3D points.

From two known points M_i and M_j, e.g. M_1 and M_3, we can simply calculate the distance d_{ij}. Furthermore we can estimate Θ_{ij} from the known 2D points m_i and m_j. Afterwards using the law of cosines, one can set up the following equation with the unknowns x_i and x_j :

$$d_{ij}^2 = x_i^2 + x_j^2 - 2x_i x_j cos\Theta_{ij} \tag{3.17}$$

The equation for the distance between two 3D points can be transformed into a function f_{ij} depending on x_i and x_j:

$$f_{ij}(x_i, x_j) = x_i^2 + x_j^2 - 2x_i x_j cos\Theta_{ij} - d_{ij}^2 = 0 \tag{3.18}$$

For the P3P problem we can set up three equations between the three points and their enclosing angles:

$$f_{12}(x_1, x_2) = 0 \tag{3.19}$$

$$f_{13}(x_1, x_3) = 0 \tag{3.20}$$

$$f_{23}(x_2, x_3) = 0 \tag{3.21}$$

When solving this quadratic systems, we get the values for x_1,x_2 and x_3. The sylvester resultant simplifies this step. [3]

$$p_1(x) = a_m x^m + a_{m-1} x^{m-1} + ... + a_1 x + a_0 \tag{3.22}$$

$$p_2(x) = b_n x^n + b_{n-1} x^{n-1} + ... + b_1 x + b_0 \tag{3.23}$$

$$Syl(p_1, p_2) = \begin{bmatrix} a_m & \cdots & a_0 & 0 & \cdots & 0 \\ 0 & a_m & \cdots & a_0 & & \\ & & \vdots & & & \\ & & & a_m & \cdots & a_0 \\ b_m & \cdots & b_0 & 0 & \cdots & 0 \\ 0 & b_m & \cdots & b_0 & & \\ & & \vdots & & & \\ & & & b_m & \cdots & b_0 \end{bmatrix} \tag{3.24}$$

p_1 and p_2 have a common root if the $determinant(Syl(p_1, p_2)) = 0$. The sylvester resultant $determinant(Syl(p_1, p_2))$ delivers a polynomial only in one unknown. After solving this polynomial one can easily estimate the other unknown by using one of the two equations. Now we have estimated all x_i and can use them to compute the corresponding $\mathbf{M}_i^C$ which represent the position of the 3D points in the camera coordinate system:

$$\mathbf{M}_i^C = x_i \mathbf{A}^{-1} \mathbf{m}_i \tag{3.25}$$

Finally, one has to estimate the rotation R and the translation t from this set of 3D points $\mathbf{M}_i^C$ in camera coordinates. First the coordinates $\mathbf{M}_i^C$ and $\mathbf{M}_i^W$ are normalized by subtracting its mean value resulting in $\mathbf{N}_i^C$ and $\mathbf{N}_i^W$.

$$\overline{\mathbf{M}}^C = \frac{1}{n} \sum_{i=1}^{n} \mathbf{M}_i^C \rightarrow \mathbf{N}_i^C = \mathbf{M}_i^C - \overline{\mathbf{M}}^C \tag{3.26}$$

$$\overline{\mathbf{M}}^W = \frac{1}{n} \sum_{i=1}^{n} \mathbf{M}_i^W \rightarrow \mathbf{N}_i^W = \mathbf{M}_i^W - \overline{\mathbf{M}}^W \tag{3.27}$$

The rotation matrix $\mathbf{R}$ can be computed by maximizing following sum:

$$\max_{\mathbf{R}} \sum_i (\mathbf{N}_i^C)^T \cdot (\mathbf{R}\mathbf{N}_i^W) \tag{3.28}$$

This equation can be transformed using the theorem

$$\mathbf{v}^T \cdot (\mathbf{R}\mathbf{u}) = trace(\mathbf{R}^T \mathbf{u}\mathbf{v}^T) \tag{3.29}$$

Applying a Singular Value Decomposition on **L** we can calculate **R**.

$$\sum_i (\mathbf{N}_i^C)^T \cdot (\mathbf{R}\mathbf{N}_i^W) = trace(\mathbf{R}^T \sum_i (\mathbf{N}_i^C)^T \cdot \mathbf{N}_i^W) = trace(\mathbf{R}^T\mathbf{L}) \text{ where } \mathbf{L} = \sum_i (\mathbf{N}_i^C)^T \mathbf{N}_i^W \tag{3.30}$$

$$\mathbf{R} = \arg\max_{\mathbf{R}} trace(\mathbf{R}^T\mathbf{L}) \tag{3.31}$$

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T \text{ where } \mathbf{L} = \mathbf{U}\mathbf{S}\mathbf{V} \tag{3.32}$$

Finally the rotation matrix is used to estimate the translation **T**:

$$\mathbf{T} = \overline{\mathbf{M}}^C - \mathbf{R}\overline{\mathbf{M}}^W \tag{3.33}$$

Now we have a closed form solution for the P3P problem.

3.6.2 PnP in OpenCV

The OpenCV library provides a function to solve the PnP problem, it is called *solvePnP*. The function can be fed with an initial guess and offers different algorithms to solve the problem. One option is to use an iterative method based on Levenberg-Marquardt optimization. Another option is called P3P, this method is based on the paper "Complete Solution Classification for the Perspective-Three-Point Problem" by Gao et al. [8] and the function requires exactly four reference points. The last method available is called EPnP (Efficient Perspective-n-Point Camera Pose Estimation) it is based on the work of F. Moreno-Noguer, V. Lepetit and P. Fua [17]. It is a non iterative solution based on virtual control points. Usually four virtual control points are introduced instead of calculating the depth x_i of the reference points in the camera coordinate system the coordinates of these control points in the camera referential are estimated, which can be done in $O(n)$. This increases the speed for higher n compared to other approaches. Equation 3.34 shows the weighting of the four control points.

$$\mathbf{p}_i^\omega = \sum_{j=1}^{4} \alpha_{ij}\mathbf{c}_j^\omega, \text{ with } \sum_{j=1}^{4} \alpha_{ij} = 1 \tag{3.34}$$

OpenCV also provides a version of *solvePnP* that uses the RANSAC algorithm (random sample consensus), the function is called *solvePnPRansac*. RANSAC is an iterative non-deterministic algorithm that means it delivers a reasonable result only with a certain probability. It was proposed in [7]. The method is based on selecting random subsets of the input data and computing a model based on this subset. The full input data is afterwards back projected using the model. If the number of inliers is the highest so far, the model is stored. This steps are repeated certain times, delivering finally a reasonable good model. Following listing shows the pseudocode of the RANSAC algorithm.

Listing 3.1: Algorithm: RANSAC

```
loop 1..numIter
        S = randomSubset(C)
        T = getModel(S)
        E = reproject(T,C)
        N = numberOfInliers(E, confidenceLevel)
        if N > maxInliers
                maxInliers = N
                bestSet = S
end
```

In OpenCVs implementation it is possible to determine the number of iterations that are run through. One can also define the threshold for the reprojection error to count as inlier, and it is possible to select a minimum number of inliers that have to lay within the model to be accepted. Further aspects concerning RANSAC are discussed in Section 4.3.3.

4 Easily generating Training Data for Part-based 3D Detection

Contents

This section describes the concept and design of the implemented system. The whole system consists of three main parts. The first part is the tracking system which is used to extract patches of the object from training video frames.

All these patches are fed into a training algorithm called the training system. In our case we use a cascade trainer based on the Viola-Jones algorithm and LBPs. The last part detects the object parts and estimates the pose within test videos in different environments using the trained information from part two. Figure 4.1 shows the subsystems and their transferred information.

4.1 Tracking System

To collect training data we have to track parts of an object over time to get there appearance from different poses and under different lightning conditions. The camera has to be moved around the object by the user to capture patches of each part. Afterwards the captured patches can be stored in different formats to be used by different training systems.

To make the detection more robust, different video sequences from different environments can be used and combined to one training set for the same object. Therefore the rigid transformation between the sequences is calculated to align the interest points between these sequences. A training set, also called a project, contains all combined information needed to train the system for one object and afterwards for running the detection stage. The tracking system shown in Figure 4.2 consists of following parts:

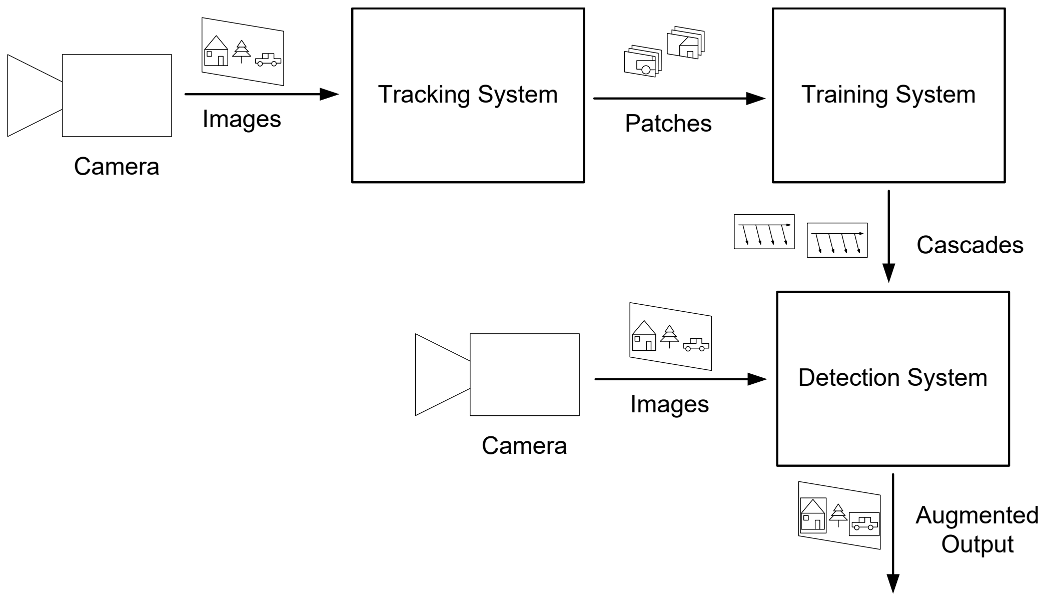


Figure 4.1: Overall Concept of the implemented System including tracking, training and detection subsystems.

- **SLAM System:** The SLAM System LSD-SLAM presented in Section [5] delivers the depth information for keyframes and the tracking information for every frame. It also delivers the gray values for each pixel and for every frame, while the depth values are only computed for salient points. The aspects of the SLAM system in our concept are discussed in Section 4.1.1
- **GUI:** The graphical user interface takes user input, more specifically, the initial position of the parts and control instructions for the capturing.
- **3D-part pose estimation:** This part is used to estimate the 3D position of each part in the world coordinate system and is responsible for proper back projection into the current frame.
- **Video registration:** If more than one video is used, the parts from previous video sequences are used to estimate the rigid transformation between these videos. Details are shown in Section 4.1.7. This transformation is used afterwards for further mapping.
- **Patch extraction:** After the back projection of the parts, the patches for the current frame can be cut-out and stored. Also if needed negative samples are recorded.
- **Training instructions:** Finally some training information has to be stored. Currently there are two training methods supported which need different instructions.

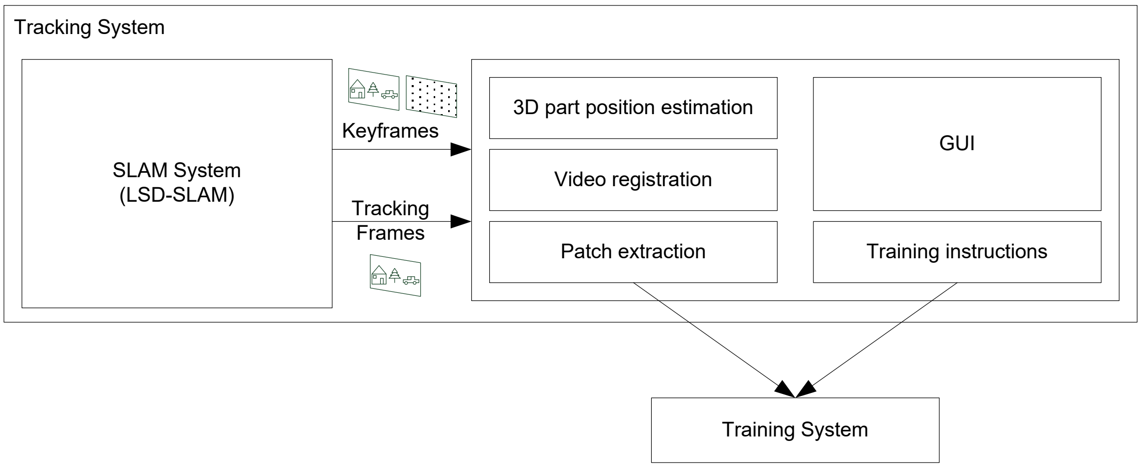


Figure 4.2: Concept of the Tracking System and its sub parts.

Each subpart of the tracking system is discussed in detail in following sections.

4.1.1 Using LSD-SLAM for 3D Localization

The SLAM system provides tracking and depth information. Properties and further information to the LSD-SLAM system are previously discussed in Section 4.1.1. In this section we focus on the usage of the SLAM system in our work.

The first frame captured by the system is also the first keyframe. It initializes the world coordinate system, all descendent keyframes are placed within this reference. If the camera moves further away from the initial view of the keyframe and hits a threshold, a new keyframe is established. At the same time the refinement of the current keyframe is finished and the deployment of its result is triggered. This leads to a delay of the mapping which is inherent for structure from motion approaches and has to be considered.

The frames processed by the LSD-SLAM core are published over two different topics. The keyframes are sent over `lsd_slam/keyframes` and all other tracked frames are sent over `lsd_slam/liveframes`. The keyframes are held in an incoming queue with a size of 20 elements, whereas live frames are stored in a single slot. Therefore older messages are dropped if they are not processed. This is due to the fact that keyframes hold important depth values whereas the live frames just hold tracking information which can be dropped if not used.

Each message provided by the SLAM System encapsulates one frame and contains following information shown in table 4.1:

`int32 id`	A ID number for the frame which is unique within the current SLAM process.
`float64 time`	This field stores a 64 bit timestamp for the frame.
`bool isKeyframe`	This field is set true when the message contains a keyframe.
`float32[7] camToWorld`	The position of the camera represented in world coordinate system (= coordinate system of first keyframe). The similarity transformation is represented with 4 values for rotation and 3 values for translation.The rotation is represented by a unit quaternion $\mathbf{q} = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$. The rotation matrix then be calculated from the quaternion coefficients: $Q = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{bmatrix}$ The Sophus library is used to transform the quaternion and the translation it into a 4x4 matrix.
`float32 fx, fy, cx, cy`	These values represent the focal length in x and y direction as well as the position of the principal point (cx,cy). This information is needed for projection of 2D image points into the 3D world coordinate system.
`uint32 height, width`	The resolution of the captured frames defined by the integer values for height and width. These values are also the resolution for the depth map.
`pointcloud`	Each element of the depth map consists of following parameters: • `float idepth`: The inverse depth for one pixel. • `float idepth_var`: The variance value for the inverse depth. • `uchar color[4]`: A color pixel as RGBA value. LSD-SLAM delivers the images as grayscale images, therefore the values for RGB are the same. The depth map has always width x height elements, but the idepth values are not set for live frames. Not all idepth values are estimated for a keyframe (semi-dense depth map), these values are set to -1.

Table 4.1: LSD-SLAM message content

4.1.2 Projection of Depth Values

LSD-SLAM delivers a depth map with every keyframe estimated. The depth maps are in direction of the view from which the keyframe was taken. To combine all depth maps they have to be mapped into the same world coordinate system resulting in a point cloud. This point cloud have to be stored and can be used to generate a new depth map from any required view V. The mapping of the point cloud is shown in Figure 4.3.

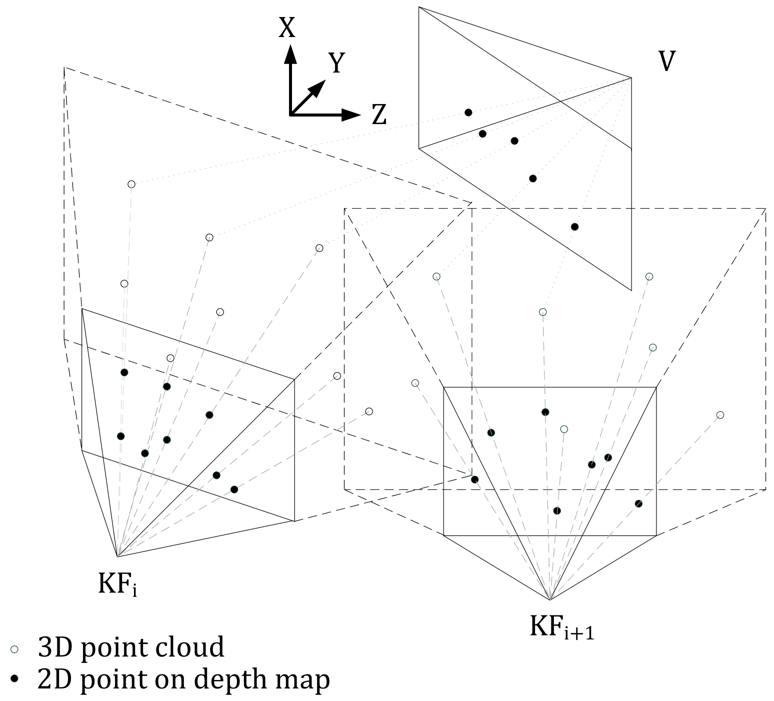
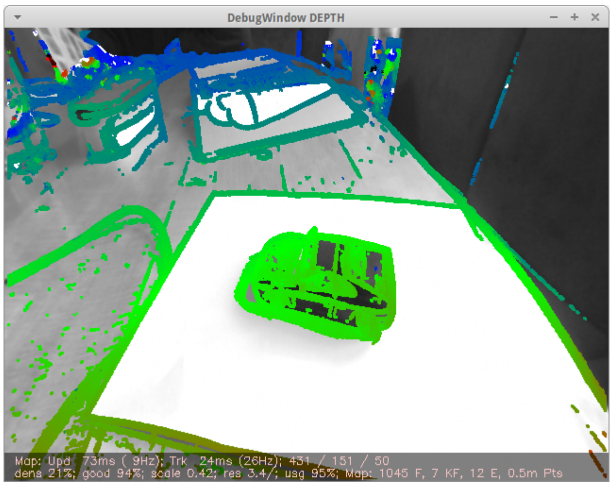


Figure 4.3: A figure of the point cloud created from different keyframes and a projection of points into a new view V.

For inserting the points from a keyframe KF_i into the point cloud, the first step is to project the 2D depth map coordinates into 3D coordinates of the corresponding keyframe. This is done by using the intrinsic parameters of the camera which are provided with every frame message but usually stay the same over time. Each point is represented by its coordinate in the depth map $(p_{x_{2D}}, p_{y_{2D}})$ and its depth value d.

$$p_{x_{3D}} = \frac{p_{x_{2D}} - c_x}{f_x} \cdot d \tag{4.1}$$

$$p_{y_{3D}} = \frac{p_{y_{2D}} - c_y}{f_y} \cdot d \tag{4.2}$$



```
Map: Upd  73ms ( 9Hz); Trk  24ms (26Hz); 431 / 151 / 50
dens 21%; good 94%; scale 0.42; res 3.4/; usg 95%; Map: 1045 F, 7 KF, 12 E, 0.5m Pts
```

Figure 4.4: Color coded depth values drawn on the grayscale image(from lsd_slam_core).

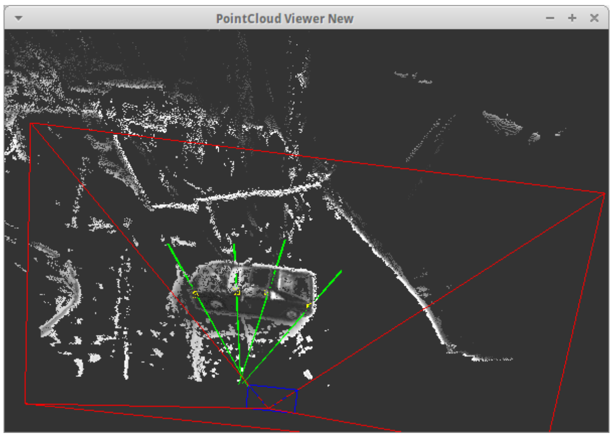


Figure 4.5: Screenshot of the point cloud (from lsd_slam_core). The red pyramid shows the current camera view. The green line depict the line of projection after selecting a point.

$$\mathbf{p}_{KF} = \begin{bmatrix} p_{x_{3D}} \\ p_{y_{3D}} \\ d \end{bmatrix} \tag{4.3}$$

After projecting the points into 3D coordinates, they are transformed into the world reference. This is done by using the transformation $T_{KF} \in sim(3)$ of its keyframe.

$$\mathbf{p} = \mathbf{T}_{KF} \cdot \mathbf{p}_{KF} \tag{4.4}$$

Further processing within the point cloud is discussed in Sections 4.1.3, 4.1.5 and 4.1.6.

4.1.3 Point Filtering

The points added to the point cloud are additionally filtered. The filtering is adapted from the original LSD-SLAM viewer, which is implemented in KeyFrameDisplay.cpp.

Three criterias lead to rejecting a point. First the variance of the inverse depth is tested. The multiplication of the variance by the depth value to the power of 4 must not exceed a certain threshold (e.g. 1). This leads to a filtering of high variance points scaled by the depth, where lower depth allows more variance. This value is additionally multiplied by the squared scale to include the scaling of the current keyframe. For this criteria a new threshold is defined.

Last, each of the eight surrounding neighbors is checked to see if the depth difference to its center point exceeds a certain value proportional to its depth. If the number of these neighbors exceeds a threshold (e.g. 5) the point is kept. This leads to a certain smoothing of the points.

4.1.4 Graphical User Interface

The graphical user interface is an important part of the tracking system. It is used to deliver monitoring information for the user and also to get user input for initializing the tracking positions. Part of the monitoring information is the live video stream showed in the main window. If a previous video sequence from the same object is already processed, the next part to select by the user is shown in the bottom left corner. Figure 4.6 shows the concept arrangement of elements for the GUI. One has to consider that selecting an object part is only possible after the first depth map is received. Therefore the camera must be moved around the object to initialize a new keyframe. Three modes are implemented to select parts.

- Selecting a fixed size patch based on the next keyframe: In this mode, the user presses a key to initialize the selection process. When the next keyframe is received, the video stream stops and the user can click to select the object part.
- Selecting a flexible size patch based on the next keyframe: The user again presses a key to start the selection. When the keyframe data arrives, the user clicks twice, first to select the centre of the object part and then to select the size of the patch. Some training methods don't accept patches with different parts, or different sizes.
- Selecting independent of key frames: Now the user can click at any time to select the object parts. Due to the delay of the mapping information the depth information might not be available. This has to be considered and can be checked in the point cloud viewer.

It is also possible to select the whole object before selecting its parts. The image patches of the entire object are needed if the pre-detection step, described in Section

4.1.4, is performed during the test phase. This is done by freely clicking any time in the object centre and dragging the mouse until the selection window fits the objects size.

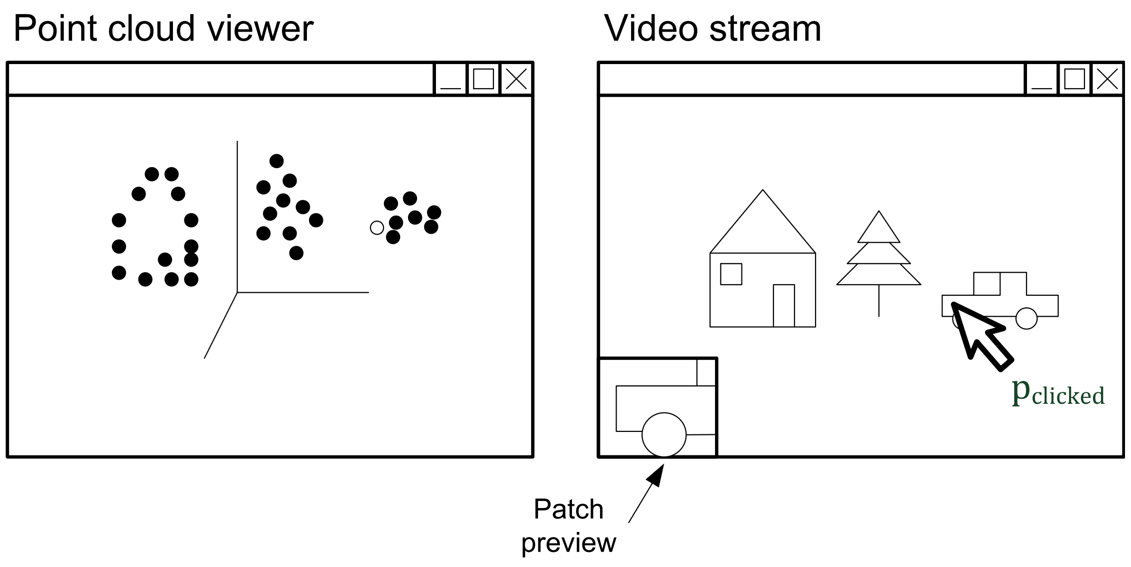


Figure 4.6: A concept of the GUI. The two windows are the point cloud viewer and the window for the live video stream. The selected position at $p_{clicked}$ marks the centre point of the part to track. The point is selected by clicking the left mouse button.

The GUI also includes a window for the point cloud. It is a 3D rendered space which is freely navigable by mouse and keys and shows the selected parts as well as the current camera view.

For multiple video sequences it is important that the system can match the same points in different sequences. So, if other video sequences are already processed and marked, the user gets a hint in form of a preview image of the next part to select. The user can skip parts if they are, for example not visible any more and can also insert new parts which were not visible in elapsed sequences.

4.1.5 Selection of Depth Points

As described in Section 4.1.4 the user manually clicks a point on the camera image to select an object part. If the selection is based on a single key frame as described in the first two modes, no projection of the point cloud is needed. In the case of free selection the points have to be projected into the current view V of the camera to generate a new depth map. Figure 4.3 shows also how the projection of the point cloud into a view V looks like. In all cases we have access to a correct depth map afterwards. The depth map is not filled or inpainted, it takes just the nearest depth values for the chosen location $p_{clicked}$ within the map. More specifically, n nearest depth values are taken (e.g. $n = 5$) and a weighted mean is calculated. The weights for the mean value computation are inversely proportional to the distance to $p_{clicked}$, as shown in Equation 4.5.

$$w_i = \frac{1}{(x_i - x_{clicked})^2 + (y_i - y_{clicked})^2} \tag{4.5}$$

The actual weighted mean is then calculated by

$$\overline{d} = \frac{\sum_{i=1}^{n} w_i \cdot d_i}{\sum_{i=1}^{n} w_i} \tag{4.6}$$

whereas some values are excluded. Values which have a high variance to the weighted mean are excluded. The weighted variance s^2 is defined as

$$s^2 = \frac{\sum_{i=1}^{n} w_i \cdot d_i^2}{\sum_{i=1}^{n} w_i} - \overline{d}^2 \tag{4.7}$$

The threshold for outliers was defined by a value of two times the standard deviation s.

4.1.6 3D Object Pose Estimation from Detected Parts

After the selection of the points, they must be tracked over the different video frames. The position must be estimated using the points' initial position within the first frame. E.g. for a point $\mathbf{p}_{KF}$ represented in coordinates of the key frame, the point $\mathbf{p}_V$ in the current view V can be calculated by

$$\mathbf{T} = \mathbf{T}_V^{-1} \cdot \mathbf{T}_{KF} \tag{4.8}$$

$$\mathbf{p}_V = \mathbf{T} \cdot \mathbf{p}_{KF} \tag{4.9}$$

The inverse transformation $\mathbf{T}_V^{-1}$ for the view V delivers correct results for the inverse case. Afterwards the point $\mathbf{p}_V$ has to be projected into the frames 2D coordinates. Transformed versions of the equations used for creating 3D coordinates in Section 4.1.2 are used.

$$p'_{x_{2D}} = \frac{p_{V_x} \cdot f_x}{p_{V_z}} + c_x \tag{4.10}$$

$$p'_{y_{2D}} = \frac{p_{V_y} \cdot f_y}{p_{V_z}} + c_y \tag{4.11}$$

As a result we get 2D coordinates $(p'_{x_{2D}}, p'_{y_{2D}})$ for the estimated point on the frame. This coordinates are used afterwards to extract the patches for tracking.

This equations project also 180°copies of the initial point p_{KF} into the image. Also the similar case of parts being viewed from their back sides must be avoided. They are also projected into the image but deliver wrong results in most cases. To avoid all these cases we set imaginary points around the point p_{KF}. After transforming them, using the equations above, we can check the orientation of these imaginary points. If these points are ordered clockwise, they are also viewed correctly from the front. Anti-clockwise cases are dismissed. To invert the transformation T_V of the view V we use following lemmas:

- The multiplication of two rotation matrices results in a new rotation matrix
- The multiplication of two transformation matrices results in a new translation matrix.
- A rotation matrix can be inversed by transposing it.
- There exists always an inverse of a transformation matrix.

We get following equation by using the lemmas and cancel out the scaling value:

$$\mathbf{T}^{-1}_{similarity} = \begin{bmatrix} 1/s^2 & 0 & 0 & 0 \\ 0 & 1/s^2 & 0 & 0 \\ 0 & 0 & 1/s^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & -\mathbf{R}^T \cdot t \\ & \\ \mathbf{0^T} & 1 \end{bmatrix} \tag{4.12}$$

4.1.7 Video Registration for Multiple-Sequence Registration

The system should be able to cope with different video sequences. For this reason the different videos must be registered among each other to interchange information correctly.

The user selects points at the beginning of the first sequence in a particular order. This order is followed in all subsequent videos as mentioned in Section 4.1.4. If at least four matching parts are selected in two video sequences a transformation between them can be computed. The resulting similarity transformation is used to select all virtual points in the same way. The issue of selecting virtual points correctly is discussed in Section 4.1.9.

Listing 4.1 shows the algorithm for estimating a similarity transformation from two point sets. The algorithm is based on the approach used in [22]. As input two point sets $A \in \mathbb{R}^{3xn}$ and $B \in \mathbb{R}^{3xn}$ are fed into the algorithm. First the two point sets are shifted into their mean values. The scaling value s is calculated from the vectors of eigenvalues $\lambda_\mathbf{A}$ and $\lambda_\mathbf{B}$ for the matrices C_A and C_B (see line 3). Under perfect conditions the relationship between two point sets $s^2 \cdot \lambda_A = \lambda_B$ holds.

To get the rotation matrix R a singular value decomposition from the matrix H is estimated (see line 5). Last, the translation vector $\mathbf{t}$ is calculated by estimating the distance between the two rotated and scaled mean values (see line 11).

Listing 4.1: Algorithm: Video registration

```
1  $A_{zero} \leftarrow A - mean(A), \quad B_{zero} \leftarrow B - mean(B)$
2  $C_A \leftarrow \frac{1}{n} A_{zero} A_{zero}^T, \quad C_B \leftarrow \frac{1}{n} B_{zero} B_{zero}^T$
```

```
3  $s \leftarrow \sqrt{\frac{\lambda_A \cdot \lambda_B}{\lambda_A \cdot \lambda_A}}$
4  $H \leftarrow \frac{1}{n} A_{zero} B_{zero}^T$
5  $U \Sigma V^T = H$
6  $R \leftarrow U V^T$
7  if det(R) = $-1$ then
8      $V \leftarrow [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$
9  end
10 $R \leftarrow U V^T$
11 $\mathbf{t} \leftarrow mean(B) - s \cdot R \cdot mean(A)$
```

4.1.8 Patch Extraction

For the patch extraction we differ between two modes. In the first mode the area around the main point $p_{clicked}$ is not transformed under different views. The size of each patch stays the same (e.g. 32x32px) over time. This mode is used for most training methods. In the second mode the 2D area including the part is transformed for different views, the content of the patch stays the same and is distorted.

Additionally a linear sparsification factor f is introduced where $1 \leq f < \infty$. It reduces the number of extracted samples depending on the training method used. The OpenCV cascade training approach needs less samples than available with the possible frame rate. A sparsification factor of $f = 10$ or $f = 7$ is chosen to get good results therefore every tenth or every seventh frame is used to extract patches. This allows the user to slow down the camera movement around the object which avoids blurring and a failing of the SLAM system by still not producing too much samples.

For some training methods the patch image must be normalized (e.g. for the CNN approach used by [2]). This is done by estimating mean value and standard deviation for linear image normalization:

$$mean_c = \frac{\sum_{i=1}^{n} I(i)}{n} \tag{4.13}$$

$$stddev_c = \frac{\sum_{i=1}^{n} (I(i) - mean_c)^2}{n} \tag{4.14}$$

$$c_{norm} = \frac{c - mean_c}{stddev_c} \tag{4.15}$$

To exploit all the image information, we extract negative samples from areas which are not including parts. We differ between random selection or pattern based extraction of negative patches. In the first case randomly selected patches are extracted, the number of negative patches is defined dependent on the training method. The pattern based approach selects patches on from an rectangular grid. In both cases it is also possible that patches from other parts are used for negative samples. The negative patches have usually the same size as the positive ones. To avoid overlapping of positive and negative patches, a minimum spacing value is defined.

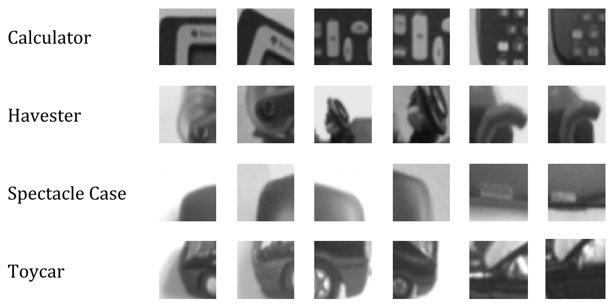


Figure 4.7: Examples of extracted patches for different objects.

4.1.9 Collecting the Virtual Points

As proposed in [2], virtual control points around the main parts are trained for 3D object detection (as discussed in Section 3.1). The implemented approach aligns the points along the coordinate axis of the current view when the point is selected. This view however can change in different video sequences. If the virtual control points are selected always through the current view, they are placed at various positions in different sequences. To avoid this issue and to get consistent training information the videos must be registered. After the video registration (see Section 4.1.7) the position of the virtual points can be set according to pre-recorded locations. To avoid bigger inaccuracy from the video registration, the position of the control points is only estimated relative to its main point.

4.1.10 Training Instructions

The training instructions combine the information extracted from the video sequences. Each training method needs different instructions. For the cascade training implemented in OpenCV we need following files:

- Positive image samples.
- Negative image samples.
- List of positive samples.
- List of negative samples.

All the instructions called for the training are written into script files. After finishing the tracking and saving all files, the training is started by using these script files. All details regarding the training are discussed in the following Section 4.2.

4.2 Training System

In this section we discuss the training system and focus on the cascade training implemented in our approach. The cascade training system in OpenCV is based on the Viola-Jones algorithm described previously in Section 3.5 and is extended by LBPs. Before starting the training, a set of samples has to be created. For this job, OpenCV provides the tool `opencv_createsamples`. After creating the samples, the tool `opencv_traincascade` is called, to start the actual training of the patches.

4.2.1 `opencv_createsamples`

The tool `opencv_createsamples` is used to create sets of training samples. The training sets consist of positive and negative samples. A positive sample patch shows the part we want to train, the negative sample shows an arbitrary patch which does not contain the part. The OpenCV tool can create a set by combining just the captured samples or it can create a much larger set by distorting the captured patches and create additional samples. In the case of creating additional distorted samples, one can define the degree of distortion by three angle values and a deviation value. The distortions then are randomly created within this defined borders.

The accuracy of the resulting classifier highly depends on the number of samples. For example, for the task of face recognition between 1000 and 5000 positive samples are used for training. Instance recognition, as it is the case in this work, can use less samples due to lower diversity of the patches.

4.2.2 `opencv_traincascade`

After creating the training samples, one can start the actual training. This is done by the tool `opencv_traincascade`,

Before starting training stage, one has to also choose a feature type, which can be selected between HAAR, LBP and HOG. The first two feature types are discussed in Section 3.5. The different results depending on the selected feature type are shown in Chapter 6.

The training is guided by three main parameters, the number of stages, the minimum hit rate and the maximum false alarm rate.

The number of stages defines how many stages are trained for the cascade, this value has a direct influence on the qualitative and quantitative result of the training.

The minimum hit rate defines how many samples of the set of positive samples must be tagged "positive" from the system within each stage. If the hit rate is reached, the training stops.

The last parameter, the maximum false alarm rate, defines how many false positives are allowed. On the one hand, a lower rate of false positives is better, but on the other hand rejects more true positives.

Further details on how to use these parameters are discussed in Section 5.2.2.

One tricky part is the number of positive samples. Some positive samples are excluded from the set due to negative outcomes. So each stage of training might exclude in the worst case $(1 - minHitRate) \cdot numPos$ positives from the set. To not run out of positive samples a number of $numPos = 0.9 \cdot numPosOrig$ is suggested [1].

To further improve the result and reducing the false alarm rate, it can be wise to add more negative samples captured in different environments. One can find many databases of images online. Moreover the same set of negative images can be used for all objects and patches.

While the entire training step might take a long time, its duration is mainly dependent on the number of samples and the numer of stages. However the training is stopped when the required hit rate is reached.

4.3 Detection System

The job of the detection system is to find all trained parts within an image and calculate the pose of the object by solving the PnP problem. After the pose estimation the image is augmented to validate the result. For 3D augmentation a mesh cube containing the object is shown. As 2D augmentation the position of the object within the image is marked and an interesting point is highlighted.

The input for the detection system are the trained cascade for each part and if available the trained cascade for the whole object. As image source one can use a stored image or video file and streams from webcams (IDS uEye cameras as well).

It has to be considered, that the images are distorted. The implemented system processes the video stream from an IDS uEye camera with a fish eye lens. The camera is first calibrated and during runtime every image is distorted with the estimated camera parameters.

4.3.1 Two Stage Detection

The detection of the parts is carried out in two steps, as shown in Figure 4.8.

- First, the entire object is detected using the corresponding trained cascade. The found 2D area within the image is extended by a certain factor and handed over to the second step.

- The second step is to search for the trained object parts within the region estimated in the first step. If the entire object was not detected in the first step, the entire image is used for the part detection.

[1]http://answers.opencv.org/question/7141/about-traincascade-paremeters-samples-and-other/ (accessed: Saturday 9th January, 2016)

This concept increases the run-time just by the run-time of one cascade detection but decreases the runtime of all other cascades by the factor $s = \frac{size(Image)}{size(ROI)}$. Additionally the number of false positive detected parts can be reduced, because big areas of the background are skipped in the part detection step. The performance is especially increased for objects with many parts.

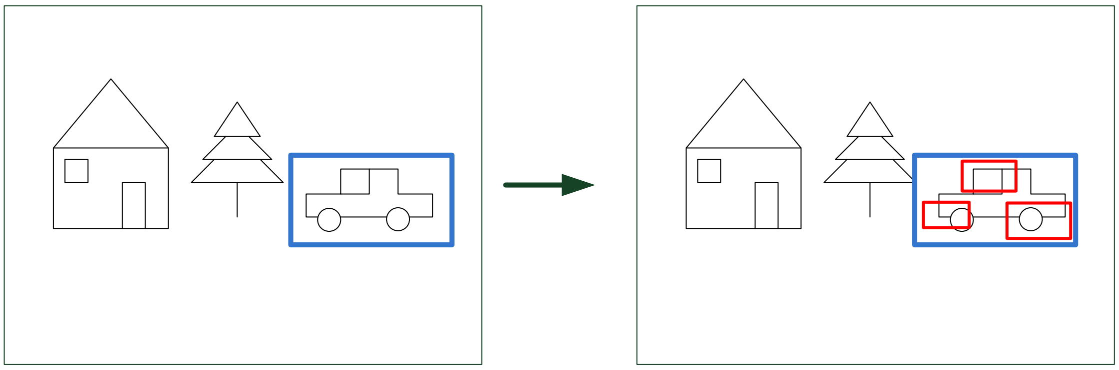


Figure 4.8: Two Stage Detection:

In Figure 4.9 the 2D region of the detected entire object and the patches are visible.

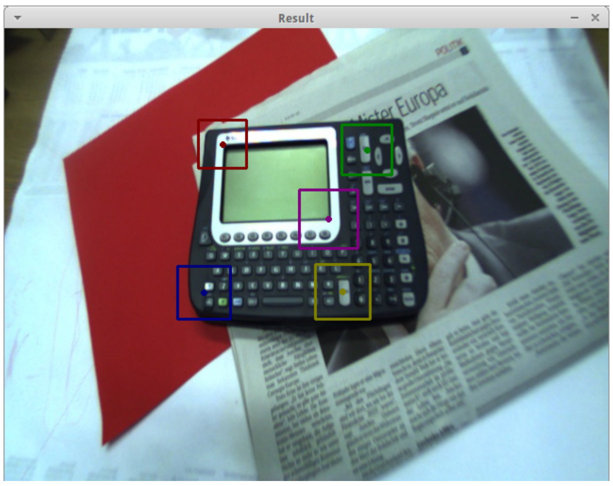


Figure 4.9: 2D regions of the detected entire object and patches on calculator example. The part centres are represented as dots.

4.3.2 Additional Performance improvements

Beside the reduction of the region of interest for the part detection by detecting the entire object, we implement a second method to reduce the computation time based on the assumption of only small changes between two consecutive image frames.

If a part is in the set of positive candidates, it is very likely that this part is in the area around its previous location. The set of positive candidates is estimated by the PnP-Ransac algorithm, described in the next Section 4.3.3.

The location of the region of interest for the part detection is based on the last positive detected instance which lies no longer than n frames behind. For slower movements, n can be smaller and the region of interest can be smaller as well. For faster movements it is the opposite.

In Figure 4.10 the regions of interest around an object part are shown. For the depth of the part history, $n = 1$ is chosen and the ROI size equals to $1.5 \cdot partSize$.

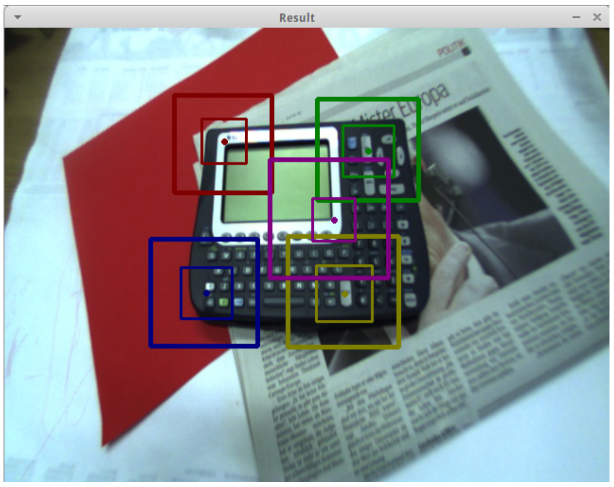


Figure 4.10: Detected patches and corresponding ROIs on calculator example.

4.3.3 Adaptations to the PnP-Ransac Algorithm

To estimate the position of the object from the 2D-3D correspondences we have to solve the PnP problem (see Section 3.6). The results from the Viola Jones framework are not perfect, therefore we use a Ransac approach, where the best detected candidates are estimated. The best part-candidates are the basis for the pose calculation.

We altered the Ransac based PnP solver implemented in OpenCV, to increase the performance of the pose estimation. In the classic Ransac approach, each sample is chosen with equal probability (see Figure 4.11a)). If we take in to account that some samples have lower confidence levels than others, the probability of selecting them randomly should decrease for the ones with lower confidence.

Each 3D point is selected with equal probability and each of them has an associated pool of 2D point candidates. The 2D candidates from the Viola-Jones detection are selected from these pools. If a pool has many 2D candidates, the probability of selecting a certain candidate is lower. This results in the fact that 2D-3D correspondences which belong to sets of many false positives get chosen less likely. The concept is visible in Figure

4.11b). The original Ransac algorithm from Listing 4.2 must be replaced by the algorithm shown in Listing 4.3.

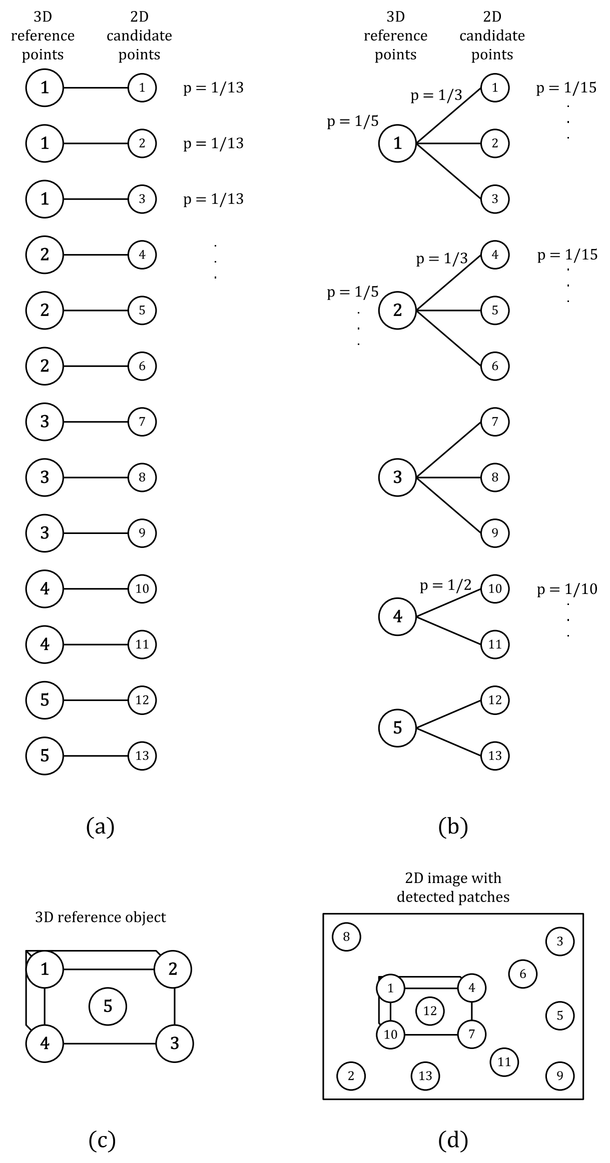


Figure 4.11: Correspondences between 3D points and 2D points. (Incl. probabilities p for selecting correspondence in RANSAC). (a) Probabilities for selecting a 2D-3D correspondence in the classic Ransac approach implemented in OpenCV. (b) Adapted probabilities: The 2D point 10 will be selected more likely than point 4 for example. (c) Sketch of the positions of 3D reference points on an object. (d) Image with the detected 2D patches.

Listing 4.2: Algorithm: RANSAC implemented in OpenCV

```
loop 1..numIter
```

```
        S = randomSubsetOfCorrespondences(C)
        T = solvePnP(S)
        E = reproject(T,C)
        N = numberOfInliers(E, confidenceLevel)
        if N > maxInliers
                maxInliers = N
                bestSet = S
end
```

Listing 4.3: Algorithm: Optimized RANSAC

```
loop 1..numIter
        S = randomSubsetUnique3DPoints(C) //This function is adapted
        T = solvePnP(S)
        E = reproject(T,C)
        N = numberOfInliers(E, confidenceLevel)
        if N > maxInliers
                maxInliers = N
                bestSet = S
end
```

5 Implementation

Contents

This chapter explains the details of the implementation of the different subsystems. All systems are implemented in C++ and use the OpenCV library.

5.1 Tracking System

The tracking system, including the LSD-SLAM system (details in Section 4.1), is a separate ROS node and is called `lsd_slam_trainer`. One project, containing one object to train, has a name, a project xml file and a working directory where the extracted information is stored. This information consists of the extracted patches and training information.

The extracted patches are held in the main memory and stored to the hard drive when demanded by the user (by pressing key 'x'). The tracking can be paused if the user wants to stop the patch extraction by pressing the key 'p'.

Each part is addressed by number which is unique for one object over the whole project, the part number. Also every video sequence has a unique video number. The captured patches are named by a string containing the project name, the part number and a consecutive number. Therefore each extracted patch can be uniquely identified.

`[working directory]/positves/[project name]-[part number]-[patch-number].png`

The XML project file contains following information:

- **videos:** Each project consists of several video sequences.

- **number:** A unique number for one video sequence.
- **frameCount:** The number of frames in this video sequence.
- **parts:** A list of parts in this video.
  - **camToWorld:** The coordinate system of the part centre.
    - **x:** The x - location of the part centre under the view camToWorld.
    - **y:** The y - location of the part centre under the view camToWorld.
    - **depth:** The depth of the part centre under the view camToWorld.
  - **numSamples:** The number of collected samples of this part within this video sequence.
  - **numNegativeSamples:** The number of negative samples collected for this part.

- **referenceImages:** A list of locations for reference images of each part. The reference image patch is shown in the user interface to illustrate the next part to select.

The project file must be accessible to the `lsd_slam_trainer` for each processed video sequence of the same project/object. Each execution of the trainer adds one entry in the video list. After the training step, the detector has also access to this xml-file. Therefore the detector has access to the same information. The location of the part centres, for example, are the ground truth for the PnP solver.

The communication between the `lsd_slam_core`, the camera system and the implemented `lsd_slam_trainer` is based on ROS and uses a publisher subscriber logic. The communication structure for the implemented system is shown in Figure 5.1.

The video stream from the camera is called `/image_raw` and is published by `ueye_manager`. The training video stream can be captured for testing and debugging reasons easily using the `rosbag` tool (see Section 3.4).

After the processing of the video stream, LSD-SLAM publishes the keyframes and the tracked frames. The corresponding publisher are called `/lsd_slam/keyframes` and `/lsd_slam/liveframes`.

The `PointCloudViewer`, as visible in Figure 4.5, is a Qt window based on `QGLViewer`. The `QGLViewer` class provides a OpenGL environment where the camera is freely navigable with use of the mouse in every direction.

For the `ImageViewer` we use the same basis, although it just presents 2D images. Using for example a OpenCV window beside a Qt-window would lead to problems.

5.1.1 Libraries

The tracking and patch extraction system uses several software libraries besides the LSD-SLAM system and the Robot Operating System:

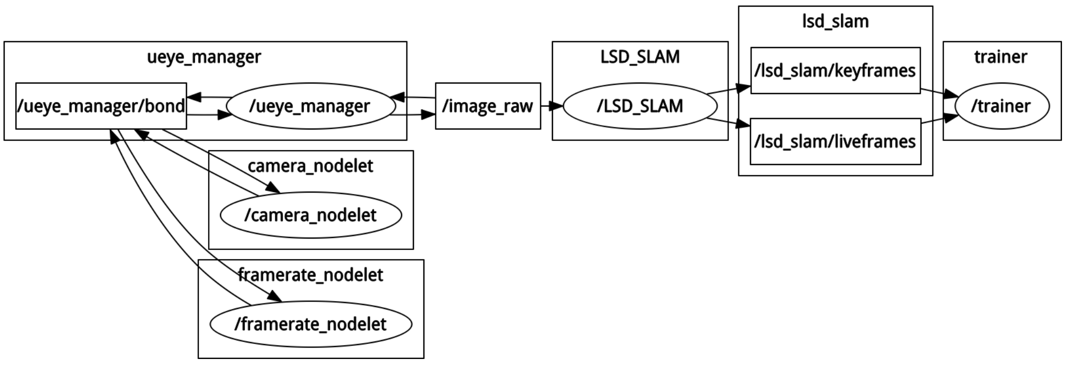


Figure 5.1: ROS topics: Publisher and subscriber. The implemented training system is designated as `trainer`.

- **Boost**[1] is used for the project handler, for threading and for string handling. Boost provides a library for creating xml files from tree structures. The project information is stored in those files (see above in this section). The threading is needed to parallelise the tracking and patch storage.
- **Qt**[2] provides several tools for efficient development of graphical user interfaces.
- **Sophus**[3] is used by LSD-SLAM to handle vectorial data structures. Therefore the tracking system must use Sophus to utilize the passed data from the SLAM system.
- **OpenCV**[4] provides many implementations of computer vision algorithms. We use this library for image preprocessing, for its data structures and mainly for its implementation of the Viola-Jones algorithm.

5.2 Training System

The training system utilizes the tools `opencv_createsamples` and `opencv_traincascade`, provided by the OpenCV library. These tools are the core of the Viola-Jones training steps.

5.2.1 Using `opencv_createsamples`

In Listing 5.1 the parameters for `opencv_createsamples` are shown.

Listing 5.1: Usage of `opencv_createsamples`

```
createsamples
[-info <description_file_name >]
```

[1]http://www.boost.org/
[2]http://www.qt.io/
[3]https://github.com/strasdat/Sophus
[4]http://opencv.org/

```
[-img <image_file_name>]
[-vec <vec_file_name>]
[-bg <background_file_name>]
[-num <number_of_samples = 1000>]
[-bgcolor <background_color = 0>]
[-inv] [-randinv] [-bgthresh <background_color_threshold = 80>]
[-maxidev <max_intensity_deviation = 40>]
[-maxxangle <max_x_rotation_angle = 1.100000>]
[-maxyangle <max_y_rotation_angle = 1.100000>]
[-maxzangle <max_z_rotation_angle = 0.500000>]
[-show [<scale = 4.000000>]]
[-w <sample_width = 24>]
[-h <sample_height = 24>]
```

Our approach uses following configuration for the training set of a patch for the calculator example:

`opencv_createsamples -info calculator-positives-0.dat -bg calculator-negatives-0.dat -vec samples.vec -num 500 -bgcolor 0 -bgthresh 0 -maxxangle 1.1 -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 30 -h 30`

In this case, the description file passes a list of patch images to `opencv_createsamples`, where the location of the files and the positions of the positive samples within the image are defined. If just one image is passed over, one can use the `-img` option where the positive sample fills the whole image. It is also possible to define a background color to define transparent areas using `-bgcolor`. Anyhow it should be considered, that the ratio of the stated values for height to weight equals the one of the sample image.

If one uses more than one initial image and wants to create more distorted versions of them, `opencv_createsamples` must be called multiple times. Naotoshi Seo wrote a useful script[5] for that issue. After using the script, the resulting multiple output *.vec files must be merged using the OpenCV tool `mergevec`.

5.2.2 Using `opencv_traincascade`

The previously created sample sets are stored in the *.vec- file and are now passed over to the training with additional negative samples (`-bg`). The result of the training is stored in the given cascade directory (`-data`), where each stage is stored separately. So the training can be stopped and started again without loosing the already trained data. All stages and the final classifier are stored in xml- files which are used by the testing/detection part.

`opencv_traincascade` accepts following parameters:

Listing 5.2: Usage of `opencv_traincascade`

```
opencv_traincascade
```

[5] http://note.sonots.com/SciSoftware/haartraining.html

```
-data <cascade_dir_name>
-vec <vec_file_name>
-bg <background_file_name>
[-numPos <number_of_positive_samples = 2000>]
[-numNeg <number_of_negative_samples = 1000>]
[-numStages <number_of_stages = 20>]
[-precalcValBufSize <precalculated_vals_bufSz_in_Mb = 256>]
[-precalcIdxBufSize <precalculated_idxs_bufSz_in_Mb = 256>]
[-baseFormatSave]
--cascadeParams--
[-stageType <BOOST(default)>]
[-featureType <{HAAR(default), LBP, HOG}>]
[-w <sampleWidth = 24>]
[-h <sampleHeight = 24>]
--boostParams--
[-bt <{DAB, RAB, LB, GAB(default)}>]
[-minHitRate <min_hit_rate> = 0.995>]
[-maxFalseAlarmRate <max_false_alarm_rate = 0.5>]
[-weightTrimRate <weight_trim_rate = 0.95>]
[-maxDepth <max_depth_of_weak_tree = 1>]
[-maxWeakCount <max_weak_tree_count = 100>]
--haarFeatureParams--
[-mode <BASIC(default) | CORE | ALL
```

Different variants of the boosting algrotihm (`-bt`) can also be selected, as discussed in Section 3.5.2. The tool distinguishes between Discrete AdaBoost (`DAB`), Real AdaBoost (`RAB`), LogitBoost (`LB`) and Gentle AdaBoost (`GAB`).

The number of stages is defined by the parameter `-numStages`. It defines how many stages are trained for the cascade (visible in Figure 3.13 in Chapter 3). This value directly influences the resulting hit rate and false alarm rate, which further affects the overall result.

The parameter `-minHitRate` defines how many of the positive samples must be detected in each stage. A higher hit rate results in more positive detections, but leads to more false detections in return. The overall hit rate is defined by multiplying the hit rates of the single stages and is defined as $hitRate = minHitRate^{numStages}$.

The false alarm rate, passed by the parameter `-maxFalseAlarmRate`, defines the maximal number of false detections in each stage. Just as for the minimal hit rate the overall false positive rate is defined as $falseAlarmRate = maxFalseAlarmRate^{numStages}$.

5.3 Detection System

The job of the detection system is to find the parts within the image, to estimate the pose of the object and to augment the scene. As image source one can use stored images, stored video sequences and webcams. Our training environment gets images from an IDS uEye camera, which can not be loaded with the standard interfaces provided by OpenCV.

5.3.1 Camera Settings

The IDS uEye camera can be accessed over an API with its specification available in [10].

Beside the different access modes, one can define gain, saturation, brightness, shutter, exposure time and white balance. Our observations showed that the white balance and shutter time affected the result most. To cope with many different environment we used the auto settings for white balance and shutter time from the camera. The auto gain function don't effect the detection result, it just improves the visibility for the user in dark environments.

5.3.2 Augmentation

The augmentation of the scene, based on the estimated object pose, requires several known parameters.

The parameters of the perspective projection in OpenGL can be defined with the function `gluPerspective`. These parameters are the view angle in degrees, the aspect ratio between width and height, the near and the far clipping plane.

The camera view model for perspective projection in OpenGL and its parameters are shown in Figure 5.2.

The video stream of the camera is shown in the background of the augmentation. This is done by an orthogonal projection, one can use the OpenGL function `setOrthogonalProjection`.

The resulting augmentation is shown in Figure 5.3.

Additional to the 3D augmentation it is also possible to augment in 2D which must be added within an orthogonal projection. Figure 5.4 shows additional 2D augmentation.

5.4 Utility Tools

This last section shows software and tools which were used to support the implementation process:

- **Qt Creator**[6] (Version 3.4) is a open source C++ IDE for development of Qt applications.

[6]http://www.qt.io/ide/

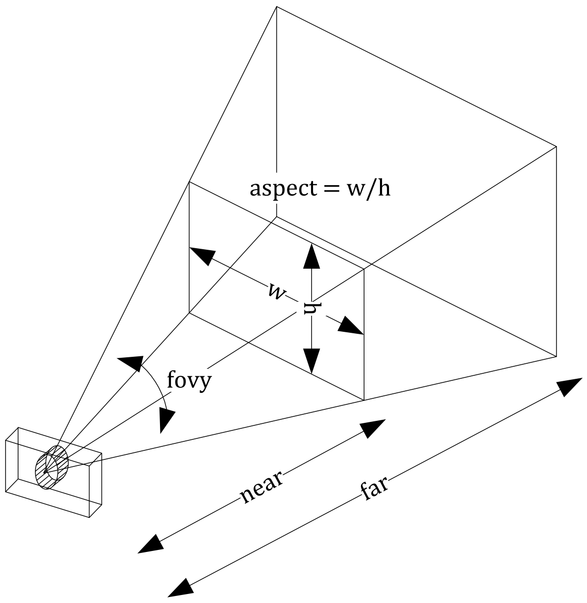


Figure 5.2: OpenGL camera view model for perspective projection.

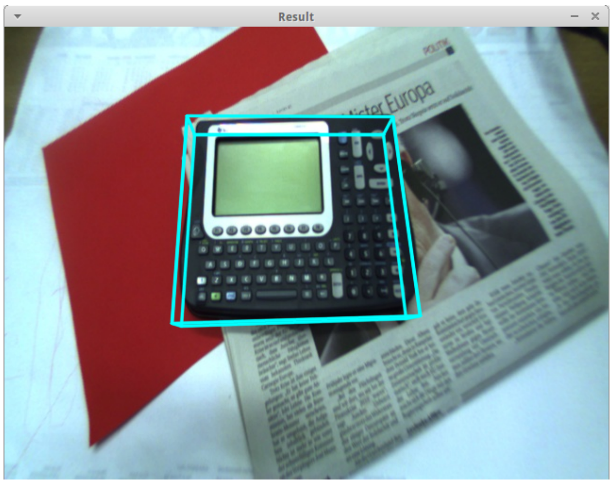


Figure 5.3: 3D Augmentation on calculator example.

- **Git**[7] (Version 2.6.2) is used for backup and version controlling of the source code.
- **Ubuntu**[8] (Version 14.04 LTS) was chosen as development platform. LSD-SLAM was also implemented on Ubuntu and therefore it was relatively easy to deploy.

[7]http://git-scm.com/
[8]http://www.ubuntu.com/

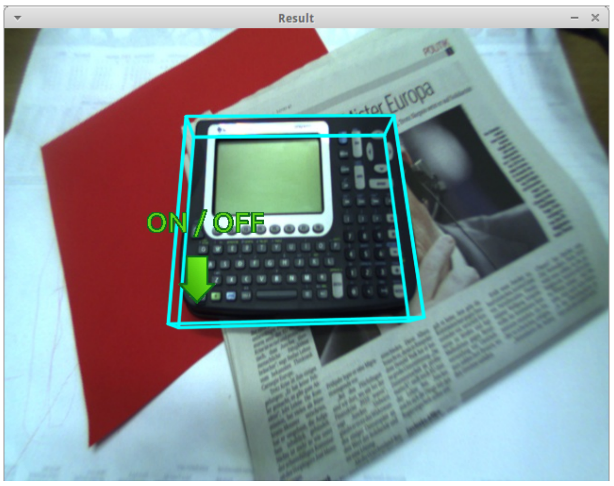


Figure 5.4: 3D and 2D Augmentation on calculator example.

- **MiKTeX**[9] (Version 2.9) The documentation was created in LaTeX. TeXstudio [10] was used as editor.

[9]http://www.miktex.org/
[10]http://www.texstudio.org/

6 Evaluation

Contents

This chapter shows the accuracy and performance of the implemented system.

The localization and tracking of the parts is based on LSD-SLAM and therefore the accuracy and performance of the system is entirely dependent on the LSD-SLAM system. The quantitative evaluation of LSD-SLAM is available in [5].

6.1 Measured Results

The accuracy of the implemented part detection system is compared with the proposed approach by Crivellaro et.al. in [2] (referred to as CNN). Besides of LBP features, also the accuracy of HAAR features is evaluated.

Several different objects where tested. Following results are based on the calculator training and test set visibile in Figure 6.1.

The training as well as the testing data was captured with an IDS UI-1221LE-C-HQ camera. The camera has an 1/3" Aptina CMOS chip, with max. 87.2 frames per second, 752px x 480px resolution and a global-shutter.

The run-time tests where executed on a PC with Intel®Core^{TM} i5 CPU M430 2.27GHz with 4GB RAM.

The accuracy of the different approaches is estimated by the 2D distance in pixel from a manually labeled ground truth. The ground truth labels are set at the center of the object parts. Additionally to the geometric error, the false-positive and false-negative rates are estimated.

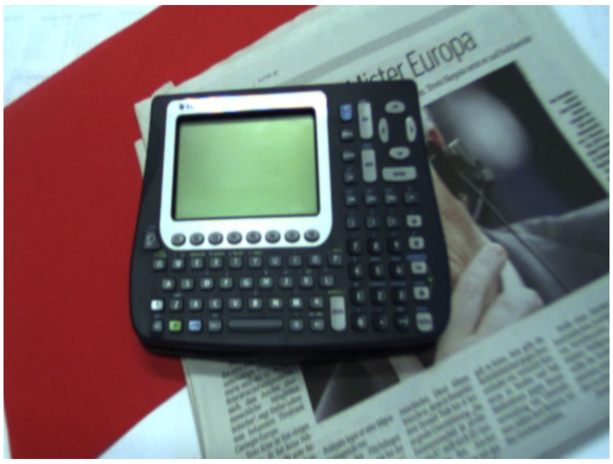


Figure 6.1: Extract from dataset calculator.

	CNN	LBP	HAAR
Average error (in pixel)	6.38	4.53	4.01
Standard deviation of error (in pixel)	3.28	3.01	2.60

Table 6.1: Results: Average accuracy of part detection in pixel (n = 1500).

	CNN	LBP	HAAR
number of false-negatives	45	217	430
false-negative rate	0.03	0.14	0.29

Table 6.2: Results: False negative rate (n = 1500).

Table 6.1 shows the results of accuracy measurements. Column CNN shows the results for the method proposed by Crivellaro et.al.. LBP shows the results for LBP-features and HAAR the results for HAAR-features in our implemented detection method. The measurements are based on a labeled dataset with 1500 entries.

In Table 6.2 one can see the false-negative rate for the different approaches. A false-negative detected part has more influence than a false-positive detection. Because false-positives are dismissed by the RANSAC step at the PnP solver. The false-positive rate is shown in Table 6.3.

The run-time of the detection algorithm depends on many factors. In Table 6.4 we present the results for our approach. We compare the case where the object is visible with the case of an invisible object, e.g. the camera is pointed in a different direction or the

	CNN	LBP	HAAR
number of false-positives	30	198	1210
false-positive rate	0.02	0.13	0.81

Table 6.3: Results: False positive rate (n = 1500).

	object visible	object not visible
Average run-time (fps)	9.62	3.45
Standard deviation of run-time (fps)	1.94	0.15

Table 6.4: Results: Average run-time for objects with five parts (n = 300).

lightning conditions are very bad. In cases of a visible object, the run-time decreases due to the improvements discussed in Section 4.3.2.

The tested object has five parts. A higher number of parts increases the run-time linearly.

6.2 Discussion of the Results

In terms of the accuracy, our approach based on the Viola Jones algorithm performs better compared to the approach based on convolutional neural networks. The HAAR features deliver slightly more accurate results than LBP, this confirms results from many previous works comparing features.

The false-negative rate indicates how many parts are not detected. The high false-negative rate is the major drawback of the Viola Jones method. This leads to flickering when not many parts are visible, due to the fact that at least four parts must be detected to estimate the pose.

The false-positive rate can be compensated by the PnP RANSAC algorithm. False positive detections are assessed as outliers and skipped. Whereby the very high false-positive rate with HAAR features strongly decreases the run-time. This is reasoned by the increased number of test models in the RANSAC approach.

The performance improvements by reducing the ROI for the part detection clearly decrease the run-time, as visible in Table 6.4. According to run-time, the differences between HAAR and LBP features can be neglected in our use-case. Although previous works showed that LBP features can be estimated faster. The CNN approach tested on different hardware and the results can not be compared reliable.

7 Conclusion

Contents

We present a method for easy collecting training samples for part based 3D detection methods. The location and tracking is based on LSD-SLAM, a recently proposed SLAM method. This allows accurate tracking with loop closures, without intense drifting.

The method we present is also capable of multiple video sequences which are registered to each other. This allows the user to acquire object patches from different environments without the issue of inconsistent coordinates.

As an alternative to the convolutional neural networks approach proposed in [2], we implemented a part detection system based on the Viola Jones algorithm. To cope with the false positives detected by the algorithm we adapted the RANSAC approach for the PnP problem solving. Additionally, we implemented performance improvements to reduce the run-time of the method.

The results show that the LBP features should be preferred over HAAR features due to the better false-negative rate and the neglectable accuracy differences. They also show that the convolutional neural networks approach delivers significantly better detection rates. This makes CNN detection better suited for the following control point detection.

7.1 Future Work

There are some possibilities to improve the detection rates and also the run-time of the presented approach by further improving the parameterization of the training.

Beside of adapting the parameters for the training of the Viola-Jones algorithm one can include additional information: The approach of exploiting locality at the 2D part detection stage can be extended to the 3D poses of the object. One can use a 3D pose model which penalizes strong pose changes within consecutive frames and therefore excluding wrong poses. This approach is already proposed in previous works.

At the part detection step the constellation of the detected parts is not considered. If the additional information, which is already available in the project file, is taken into account, wrong constellations can be excluded before they are handed over to the PnP Ransac solver.

The accuracy of the part detection can be increased if the part reference patches are included in the detection pipeline. The position of these patches is known, and therefore they can be used to increase the accuracy if their gradient images are matched on the input frame with simple template matching. The accuracy increases especially for long narrow objects, where the error can strongly enlarge.

To get better and automated augmentation, the object can be segmented in 3D space from the point cloud. The segmentation can use the position of the parts to define roughly the object borders. The resulting 3D hull can then be augmented over the object in the detection system or can be used for various other applications.

A Quick Start Guide

Contents

This chapter gives a quick introdcution on how to install and run the implemented systems.

A.1 Installation

This section gives introductions on how to install the implemented software.

All systems where tested on Ubuntu 14.04 LTS.

The compilation of the ROS nodes (`lsd_slam_core` and `lsd_slam_trainer`) works with the GNU Compiler Collection (GCC) v4.6. The compilation with GCC v4.8.4 (actual version in Ubuntu 14.04) led to problems at run time.

A.1.1 Installation of ROS

1. Installation of rosinstall:
   ```
   sudo apt-get install python-rosinstall
   ```
   If `python-rosinstall` is not available one can use:
   ```
   sudo apt-get install python-pip
   sudo pip install -U rosinstall
   ```

2. Prepare the ros working space:
   ```
   mkdir ~/rosbuild_ws
   cd ~/rosbuild_ws
   ```

```
rosws init . /opt/ros/indigo
mkdir package_dir
rosws set ~/rosbuild_ws/package_dir -t .
echo "source ~/rosbuild_ws/setup.bash" >> ~/.bashrc
bash
cd package_dir
```

3. Installation of ROS components:
```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net --recv-key
0xB01FA116
sudo apt-get update
sudo apt-get install ros-indigo-desktop-full
sudo apt-get install liblapack-dev libblas-dev freeglut3-dev
libqglviewer-dev libsuitesparse-dev libx11-dev
```

A.1.2 Compile the Binaries

Copy the source files into the `package_dir` directory and run:
```
rosmake lsd_slam
```

A.1.3 Install the uEye Camera Driver

1. Download the latest driver from http://www.ueyesetup.com
2. Extract the ZIP archive
3. Run the driver installation script:
```
sudo sh ./ueyesdk-setup*.run
```
4. Connect the camera
5. Install the ROS uEye camera driver:
```
sudo apt-get install ros-indigo-ueye
```

A.2 Camera Calibration

The camera calibartion uses the calibration tool from PTAM. First, the calibration tool must be downloaded and installed:

1.
   ```
   mkdir -p ~/catkin_ws/src
   cd ~/catkin_ws/src
   git clone https://github.com/ethz-asl/ethzasl_ptam
   cd ../
   catkin_make
   ```

2. Start capturing frames from the uEye camera:
   ```
   roslaunch ueye nodelets.launch pixel_clock:=20 frame_rate:=40
   color:=1
   ```

3. Run the camera calibrator:
   ```
   ~/catkin_ws/devel/lib/ptam/cameracalibrator /image:=/image_raw
   ```
   Following tutorial gives an introduction on how to calibrate a camera in PTAM: http://wiki.ros.org/ethzasl_ptam/Tutorials/camera_calibration

4. Copy the five parameters of calibration (d1, d2, ..., d5), which is printed out to the console with the following format:
   ```
   Camera calib is <d1> <d2> <d3> <d4> <d5>
   ```

5. Create a new calibration file:
   ```
   gedit <path_to_lsd_slam>/lsd_slam/lsd_slam_core/calib/ueyeCalib.cfg
   ```

6. Paste the following in this file and save the file:
   ```
   <d1> <d2> <d3> <d4> <d5>
   752 480
   crop
   640 480
   ```

A.3 Running the Trainer

Start the subsystems in following order:

1. `roscore`

2. `roslaunch ueye nodelets.launch pixel_clock:=20 frame_rate:=40`

3.
   ```
   rosrun lsd_slam_trainer trainer _project:=calculator
   _dir:=/var/LSD_TRAINER/calculator/ _config:=Configuration.xml
   ```
   The content and parameters of the configuration xml-file are discribed in the sample file: Configuration.xml.

4. `rosrun lsd_slam_core live_slam /image:=image_raw _calib:=./calib/ueyeCalib.cfg`

A.4 Starting the Detection

The detector accepts following parameter:

```
Usage:  ./Detector -c calib_file -p project_name -d project_directory [-u
[camera_number] | -v [camera_number] | -i filename | -f src_directory | -el
src_directory | -e ]
```

It can be choosen between different image sources:
`-u` uEye camera.
`-v` ordanary webcam.
`-i` stored image from hdd.
`-f` directory of stored images from hdd.
`-el` images from hdd, with additional ground truth for error evaluation.

Compile the detector and run it:

```
cmake .
make
./Detector -c ./calib/camera_data_ueye.xml -p calculator -d
/var/LSD_TRAINER/calculator/ -u
```

Bibliography

[1] About ROS. `http://www.ros.org/about-ros/`. Accessed: 2015-09-15. (page 21)

[2] Alberto Crivellaro, Mahdi Rad, Yannick Verdie, Kwang Moo Yi, Pascal Fua, and Vincent Lepetit. A Novel Representation of Parts for Accurate 3D Object Detection and Tracking in Monocular Images. In *Proceedings of the International Conference on Computer Vision*, 2015. (page iii, v, 1, 2, 6, 13, 14, 15, 22, 43, 44, 59, 63)

[3] Vikas Dhiman, Julian Ryde, and Jason J Corso. Mutual localization: Two camera relative 6-dof pose estimation from reciprocal fiducial observation. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1347–1354. IEEE, 2013. (page 30)

[4] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 998–1005. IEEE, 2010. (page 9)

[5] Jakob Engel, Thomas Schoeps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014*, volume 8690 of *Lecture Notes in Computer Science*, pages 834–849. Springer International Publishing, 2014. ISBN 978-3-319-10604-5. doi: 10.1007/978-3-319-10605-2_54. URL `http://dx.doi.org/10.1007/978-3-319-10605-2_54`. (page 8, 9, 18, 19, 23, 34, 59)

[6] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010. (page 9, 10)

[7] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. (page 32)

[8] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):930–943, 2003. (page 31)

[9] S Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. (page 6, 7)

[10] IDS Imaging Development Systems GmbH. *Benutzerhandbuch: uEye Software Development Kit (SDK).* (page 56)

[11] Chris Harris and Carl Stennett. Rapid-a video rate object tracker. In *BMVC*, pages 1–6, 1990. (page 9)

[12] Richard Hartley and Andrew Zisserman. *Multiple View Geometry.* Cambridge University Press, 2003. (page 16)

[13] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. (page 8)

[14] Georg Klein and David W Murray. Full-3d edge tracking with a particle filter. In *BMVC*, pages 1119–1128, 2006. (page 9)

[15] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A scalable tree-based approach for joint object and pose recognition. In *AAAI*, 2011. (page 9)

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (page 14)

[17] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009. (page 31)

[18] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z Li. Learning multi-scale block local binary patterns for face recognition. In *Advances in Biometrics*, pages 828–837. Springer, 2007. (page 23, 28)

[19] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Pattern Recognition*, pages 297–304. Springer, 2003. (page)

[20] Joseph J Lim, Aditya Khosla, and Antonio Torralba. Fpm: Fine pose parts-based model with 3d cad models. In *Computer Vision–ECCV 2014*, pages 478–493. Springer, 2014. (page 10)

[21] David G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):441–450, 1991. (page 9)

[22] Daniel J Mirota, Hanzi Wang, Russell H Taylor, Masaru Ishii, Gary L Gallia, and Gregory D Hager. A system for video-based navigation for endoscopic endonasal skull base surgery. *Medical Imaging, IEEE Transactions on*, 31(4):963–976, 2012. (page 42)

[23] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011. (page 8)

[24] Nadia Payet and Sinisa Todorovic. From contours to 3d object detection and pose estimation. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 983–990. IEEE, 2011. (page 9, 10)

[25] Bojan Pepik, Michael Stark, Peter Gehler, and Bernt Schiele. Teaching 3d geometry to deformable part models. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3362–3369. IEEE, 2012. (page 10)

[26] Victor A Prisacariu and Ian D Reid. Pwp3d: Real-time segmentation and tracking of 3d objects. *International journal of computer vision*, 98(3):335–354, 2012. (page 9)

[27] Victor Adrian Prisacariu, Aleksandr V Segal, and Ian Reid. Simultaneous monocular 2d segmentation, 3d pose recovery and 3d reconstruction. In *Computer Vision–ACCV 2012*, pages 593–606. Springer, 2013. (page 9)

[28] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009. (page 21)

[29] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994. (page 5)

[30] Ashish Shrivastava and Arpan Gupta. Building part-based object detectors via 3d geometry. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1745–1752. IEEE, 2013. (page 9, 10)

[31] Iryna Skrypnyk and David G Lowe. Scene modelling, recognition and tracking with invariant image features. In *Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on*, pages 110–119. IEEE, 2004. (page 9)

[32] Veronica Teichrieb, Joao Paulo Silva do Monte Lima, Eduardo Lourenço Apolinário, Thiago Souto Maior Cordeiro de Farias, Márcio Augusto Silva Bueno, Judith Kelner, and Ismael HF Santos. A survey of online monocular markerless augmented reality. *International Journal of Modeling and Simulation for the Petroleum Industry*, 1(1), 2007. (page 5, 6)

[33] Carlo Tomasi and Takeo Kanade. *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991. (page 5)

[34] Luca Vacchetti, Vincent Lepetit, and Pascal Fua. Stable real-time 3d tracking using online and offline information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1385–1391, 2004. (page 9)

[35] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001. doi: 10.1109/CVPR.2001.990517. (page 2, 23, 24, 25, 26, 27)

[36] Yihong Wu and Zhanyi Hu. Pnp problem revisited. *Journal of Mathematical Imaging and Vision*, 24(1):131–141, 2006. (page 29)

[37] Yu Xiang, Changkyu Song, Roozbeh Mottaghi, and Silvio Savarese. Monocular multiview object tracking with 3d aspect parts. In *Computer Vision–ECCV 2014*, pages 220–235. Springer, 2014. (page 9, 11)

[38] Zhengyou Zhang. Camera calibration. In G. Medioni and S.B. Kang, editors, *Emerging Topics in Computer Vision*, chapter 2, pages 4–43. Prentice Hall Professional Technical Reference, 2004. (page 16)