UNIVERSITÄT SALZBURG

SUPPORT FRAMEWORK FOR OBSTACLE DETECTION ON AUTONOMOUS TRAINS

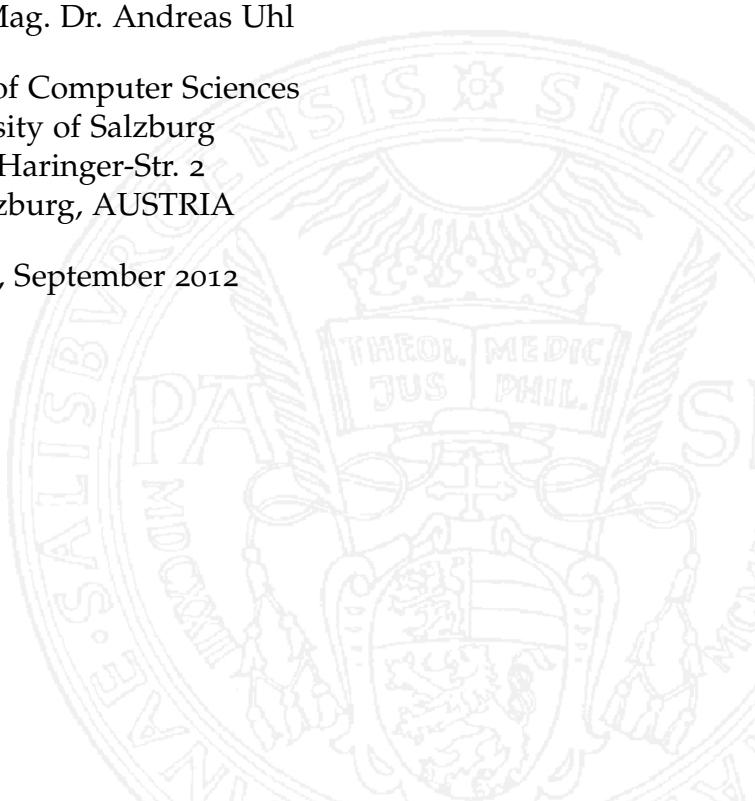
by
Michael Gschwandtner

Dissertation submitted to the
Faculty of Natural Sciences, University of Salzburg
in partial fulfillment of the requirements
for the Doctoral Degree

Thesis Supervisor
Univ.-Prof. Mag. Dr. Andreas Uhl

Department of Computer Sciences
University of Salzburg
Jakob-Haringer-Str. 2
5020 Salzburg, AUSTRIA

Salzburg, September 2012



Dedicated to the loving memory of Friedrich Gschwandtner.

You were always proud of me. I hope you would have been proud of this work too. 1924 – 2011

ABSTRACT

Autonomous driving vehicles are a rapidly emerging technlogy that will radically transform the face of public and personal transportation in the near future. This work is part of project *autoBAHN*, which has the goal to develop an autonomous driving train and in turn prevent small railroad branch lines from beeing shut down due to cost saving measures. The focus of research in the field of sensors used for autonomous vehicles is on the detection of obstacles. However, detecting obstacles is only a part of an autonomous driving vehicle. This work aims at providing the basis for making a complete autonomous driving train possible. This basis is a combination of sensor calibration techniques, track detection for railroads to classify obstacles and non-obstacles and simulation of sensor data for the verification of the individual underlying algorithms.

ZUSAMMENFASSUNG

Öffentlicher Verkehr und Individualverkehr werden sich in Zukunft durch autonom fahrende Fahrzeuge stark verändern. Neben autonom fahrenden Autos sind im Speziellen auch autonome Züge ein wichtiger Faktor. Diese Arbeit ist Teil eines Projektes (*autoBAHN*), das sich mit der Frage beschäftigt, ob autonom fahrende Züge die Schließung unrentabler Nebenbahnen verhindern können. Für ein autonom fahrendes Fahrzeug ist die Erkennung von Hindernissen eine zentrale Aufgabe. Forschung im Bereich autonomer Fahrzeuge beschäftigt sich intensiv mit dem Problem der Hinderniserkennung. Um aber einen vollständig autonom fahrenden Zug zu entwickeln, reicht es nicht, sich nur mit reiner Hinderniserkennung zu beschäftigen. Grundlagen für einen autonom fahrenden Zug sind unter anderem die Kalibrierung der verschiedenen Sensor-Technologien, die Erkennung der Schienen unmittelbar vor dem Zug, um Gefahrenzonen zu erkennen und die Simulation der Sensordaten, um die zugrunde liegenden Algorithmen verifizieren zu können.

PUBLICATIONS

Some ideas and figures of Chapter 2 and Chapter 3 have appeared previously in ([17])

M. Gschwandtner, R. Kwitt, W. Pree, and A. Uhl. *Infrared camera calibration for dense depth map construction.* In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV '11), ©IEEE*

Some ideas and figures of Chapter 5 have appeared previously in ([19])

M. Gschwandtner, W. Pree, and A. Uhl. *Track detection for autonomous trains.* In *Advances in Visual Computing, volume 6455* of *Lecture Notes in Computer Science, (ISVC 2010), ©Springer-Verlag Berlin, Heidelberg*

Some ideas and figures of Chapter 6 have appeared previously in ([18, 34])

M. Gschwandtner, R. Kwitt, W. Pree, and A. Uhl. *BlenSor: Blender Sensor Simulation Toolbox*. In *Proceedings of the 7th international conference on Advances in visual computing - Volume Part II*, volume 6939 of *ISVC'11, ©Springer-Verlag Berlin, Heidelberg*

P. Marion, R. Kwitt, B. Davis, and M. Gschwandtner. *Pcl and paraview - connecting the dots.* In *IEEE CVPR Workshop on Point Cloud Processing (PCP '12), ©IEEE*

Other publications by the author which are unrelated to this thesis ([16, 20]):

M. Gschwandtner, M. Liedlgruber, A. Uhl, and A. Vécsei. *Experimental study on the impact of endoscope distortion correction on computer-assisted celiac disease diagnosis.* In *Proceedings of the 10th International Conference on Information Technology and Applications in Biomedicine (ITAB'10), ©IEEE*

M. Gschwandtner, J. Hämmerle-Uhl, Y. Höller, M. Liedlgruber, A. Uhl, and A. Vécsei. *Improved endoscope distortion correction does not necessarily enhance mucosa-classification based medical decision support systems.* In *Proceedings of the IEEE International Workshop on Multimedia Signal Processing (MMSP'12), ©IEEE*

Doing linear scans over an associative array
is like trying to club someone to death with a loaded Uzi.

— Larry Wall

ACKNOWLEDGMENTS

I want to thank everybody who has provided valuable input during my work on this thesis, Roland Kwitt and Stefan Huber for their opinion on many of the topics in this work. I want to thank Andreas Uhl for beeing a great supervisor and an endless resource of fresh ideas.

Special thanks goes to my wife Anne-Marie for just beeing there when I needed her and beeing patient enough to let me finish my work but not too patient to help me meet my deadlines.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

BlenSor Blender Sensor Simulation

DARPA Defense Advanced Research Projects Agency

DoG Difference of Gaussian

FOV Field of View

GPS Global Positioning System

GPU Graphics Processing Unit

FPS Frames per Second

IPM Inverse Perspective Mapping

IR Infrared

LCM Lightweight Communications and Marshalling

LED Light-emitting Diode

LIDAR Light Detection and Ranging

PCL Pointclouds Library

PTP Precision Time Protocol

RANSAC Random Sample Consensus

RGB Red Green Blue

ROI Region of Interest

ROS Robot Operating System

SAD Sum of Absolute Differences

SVM Support Vector Machine

ToF Time of Flight

UIC International Union of Railways

Part I

INTRODUCTION

1 INTRODUCTION

This work is a part of project *autoBAHN [13]*. Project *autoBAHN* evaluates whether fully autonomous trains can be operated on smaller branch lines in Austria. In recent years numerous tracks have been shut down due to cost saving measures. While there is still demand for those lines it is not enough to sustain a cost-efficient operation. The premise of *autoBAHN* states that increased frequency on those lines would make them more attractive and thus increase passenger numbers. However such an increase in frequency would require more train conductors nullifying the gain in passenger numbers. Project *autoBahn* further states that the cost for additional train conductors could be prevented if the trains were operated autonomously.

While autonomous trains are already operated around the world, their tracks are usually unaccessible by non-authorized personnel. Examples of such autonomous trains are the Las Vegas Monorail [1] and the Copenhagen Metro [2]. In case of the Las Vegas Monorail the tracks are completely unaccessible for a passenger since it drives on an elevated track and all stations are guarded with automatic doors, which only open when a train is inside the station. In contrast to such already available autonomous trains the aim of *autoBAHN* is an operation on unmodified branch lines in Austria. Such branch lines usually have no protection against unauthorized access to the tracks at all. Some "train stations" are even not more than a park-bench and a traffic sign. Many railroad crossings are secured solely by a traffic light. Physically preventing unauthorized access to such branch lines through structural changes is virtually impossible without rebuilding the whole track.

1.1 VEHICLE

The test vehicle is a rail-car that includes the passenger section. It contains 40 seats for passengers and was modified by Siemens and Elin to be operated automatically while preserving the capability of manual operation. Communication between the autoBAHN system and the control module is handled via a *CANopen* bus. Control commands allowed from the autoBAHN system to the control modul are setting the target velocity and signaling with the train horn. Ensuring safe acceleration/deceleration is handled by the control module. Even if the autoBAHN system would issue potentially dangerous speed

1 http://www.lvmonorail.com/
2 http://intl.m.dk/

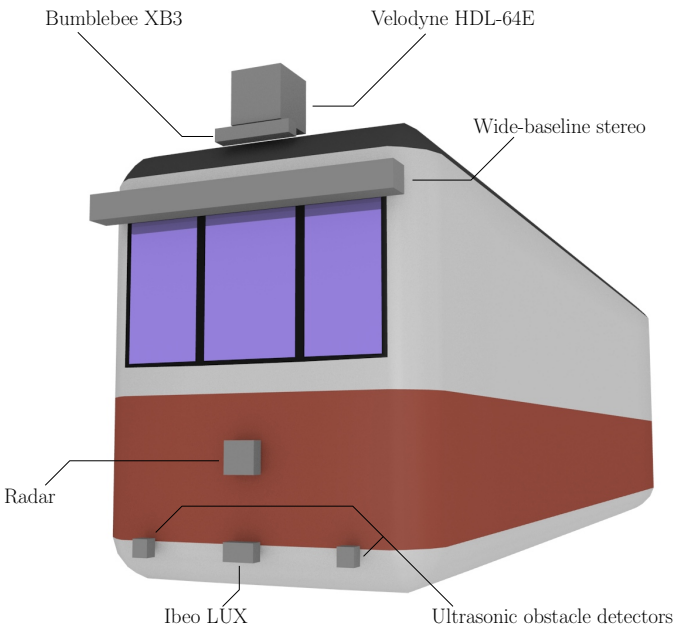


Figure 1: Overview of all tested sensors

settings for acceleration/deceleration or ignore traffic lights, the control module would not execute them. Such an additional safety layer is necessary because the autoBAHN system in its current form has no connection to the *train traffic control system* and does therefor not know whether it is allowed to enter certain track sections. Another train inside a section may also forbid such a maneuver.

1.2 SENSORS

Our autonomous train is equipped with numerous sensors (Figure 1) that in some way or another are used to detect objects in the surrounding environment. Each sensor provides a different view on the environment – some of them by employing completely different sensing technologies. Several different sensor technologies were tested during this project – some of them active sensors and some of them passive.

Active sensors emit a signal and measure the part of the signal that is reflected back to the sensor

- Laser range finder, also known as Light Detection and Ranging (LIDAR) devices.
  This type of sensors can be further divided into LIDARs that scan only very few lines and those which use many layers to do a partial or full scan of the environment.
- RADAR
- Ultrasound

(a) Train with wide-baseline stereo-camera, Radar, Ibeo LUX and Ultrasonic obstacle detectors

(b) Train with Velodyne sensor on top

Figure 2: Different sensor setups

Passive sensors may be augmented by active components i.e. additional light sources. However, their basic scanning principle does not rely on such augmentation

- *Monoscopic cameras*
  Cameras seem a natural choice as a sensor for autonomous vehicles since they closely resemble how a human driver perceives the environment. Monoscopic cameras can be further classified by the frequency spectrum they are able to detect
  - *Visible light cameras* - conventional camera systems designed for the parts of the light spectrum also visible to the human observer
  - *Near infrared cameras* - often referred to as Nightvision cameras
  - *Far infrared cameras* - also known as thermal imaging cameras
- *Stereoscopic Cameras* - even if stereoscopic cameras are merely a combination of monoscopic cameras, the data processing and extraction of useful information from those sensors works for the better part in a completely different way. This justifies the decision to put them in a sensor class of their own.

Two possible sensor setups are shown in Figure . The setup shown in Figure 2a contains all sensors except the Velodyne and the Bumblebee XB3. The sensor setup shown in Figure 2b contains only a Velodyne and a Radar sensor.

1.2.1 *Velodyne*

LIDAR devices are the key sensor technology in today's autonomous vehicle systems. Their output is used for obstacle detection, tracking, surface reconstruction and object segmentation, just to mention a few. Many algorithms exist which process and analyze the output of such devices. The Velodyne HDL-64E S2 is a rotation scanner with 64 lasers. The version used in this project can provide up to 15 scans (full rotations) per seconds. This scanner is used for example in systems like *Junior* ([37]), by the Tartan Racing Team ([54]) and the autonomous car of the Karlsruhe Institute of Technology ([39])

1.2.2 *IBEO*

The IBEO Lux is a rotating LIDAR scanner with up to four scan layers. The vertical Field of View (FOV) is 3.2° and the horizontal FOV is either 110°, if only two scanning layers are used, or 85°, if all four scanning layers are used. The scanner is designed for automotive use. The four layers with the vertical FOV of 3.2° are meant to compensate the vehicle pitch during acceleration, deceleration and driving over bumps in the road.

The output of the IBEO scanner is either a point-cloud or a list of detected objects. Detected objects are tracked throughout the scans. The scanner provides and estimated size, position and speed of tracked objects. The IBEO Lux was used for example in Junior ([37]) the Stanford entry for the Defense Advanced Research Projects Agency (DARPA) Urban Challenge and by several other DARPA candidates. Another example is an experimental semi-automatic valet parking service [49].

1.2.3 *Infrared*

The FLIR PathFindIR camera is a thermal imaging sensor with a resolution of 320×240 pixel. Although the sensor is able to output data at 25 Hz the version we are using can only provide 8.3 Hz due to international export regulations. With a horizontal FOV of 36° and a vertical FOV of 29° the system is equipped with a wide angle lens.

The sensor is mounted below the middle window of the driver's cabin, forward-facing with the principal axis of the camera perpendicular to the front surface of the train.

1.2.4 *Monocular cameras*

Monocular cameras are a very attractive sensor in autonomous systems because they are relatively cheap but still very flexible. There are usually no moving parts and their properties can be tailored to the

specific scenario. One can for example choose different lenses, different color/wavelength filters, sensor resolution, rate of acquisition and exposure time. This flexibility makes them very attractive for a wide area of use cases.

In autonomous vehicle systems they are used for example for pedestrian detection ([22]) by applying an AdaBoost algorithm to do a rough estimation of interest regions followed by an Support Vector Machine (SVM) classifier that decides whether or not the region is a pedestrian. In visual odometry applications like [58] corresponding vertical lines are used to estimate the ego-motion of a mobile robot platform.

But even if monocular cameras are not used for detecting objects or motion estimation they are also used for visualization and situation recording. It is very important to see what the system has seen when one tries to find problems in the algorithms.

We are using several *Basler scout scA1300-32gm* gray-scale cameras capable of acquiring 32 frames per second at a resolution of 1296x966 pixels.

1.2.5 *Stereo cameras*

Even though much work on monocular obstacle detection has already been done, the distance of an object is inferred only from prior knowledge like their size or known cues in the scene. Using two or more cameras in a stereo setup, the distance to objects can be triangulated. To achieve this, corresponding points in both views have to be detected.

Stereo cameras have all the advantages of monocular cameras while at the same time enabling them to reliably calculate the distance of objects in the image. Since cameras are passive sensors, they will not interfere with each other, no matter how many of them are viewing the same scene. Even though the fact that this is a passive sensing technology may be an advantage in some situations (no interference), it is also the greatest weakness of monocular and stereo cameras. No matching or detection is possible when objects do not reflect enough light or when the texture is too weak to calculate reliable matches. In addition to *Basler* cameras the system is equipped with a *Bumblebee XB3 stereo* camera with a resolution of 1280x960 at 15 fps. The XB3 is attached to the base of the Velodyne HDL-64E2 and is used mainly for visualization purposes.

1.3 PROCESSING PLATFORM

Having many sensors also has a disadvantage – processing costs. Using a single system to process all sensors inputs may be feasible on a high-performance machine, but even on such systems there is a

point where sensor input can no longer be processed. To allow such a system to scale beyond the capacities of a single machine, the work has to be distributed to several systems and only the pre-processed data should be sent to a central processing system. For example, it does not make sense to send all the data from a Velodyne scanner to the sensor fusion unit if the only data that is processed are the bounding volumes of objects. In such a case it is much more efficient to process the data on the system where it is acquired and to only send the bounding volumes to the sensor fusion.

While it would be possible to implement such a system on our own we are going to use one of the state-of-the-art messaging libraries that are designed especially for such environments.

We are going to define the main criteria by which the system is selected

- *Fast communications*
  If the processing units are located on different computers, there are network transmissions involved which are a limiting factor. However, it is still imperative that messages are transmitted as fast as possible. A plus would also be if the messaging system could exploit the fact that some processing units are located on the same computers and do not need to communicate over sockets but could rather communicate over shared memory.

- *Recording capabilities*
  The system should be able to record the messages and play them back. An additional advantage would be if the time could be recorded too. Processing units could then work on the data as if they were running on the actual test vehicle.

- *Inspection*
  Monitoring message bandwidth and verifying what data gets transmitted is an important task during development and debugging. The messaging system should include tools to assist in verifying the correct behavior of the system as a whole.

Using these criteria we selected two popular messaging systems. The Robot Operating System (ROS) by Willow Garage and Lightweight Communications and Marshalling (LCM)

1.3.1 *ROS*

One of the most popular messaging systems in robotics is the ROS ([41]). However, the name is misleading, as ROS is not an actual operating system, but rather a very big collection of programs and libraries that provide means of communication and standard utilities for data processing of point clouds, images, etc. Each process in a ROS system is called “node”. Nodes can communicate by publishing messages,

subscribing to messages, providing services and calling services. A ROS system consists of a master node (*roscore*) and several nodes that process or provide data inside the system. Communication between ROS nodes is done via TCP or UDP and in special cases via shared memory.

Messages are addressed via so called topics. A topic is a named message-bus to which the node that created the topic publishes messages and to which none, one or many nodes can be subscribed. This decouples the sender from the receiver. A sender does not need to know how many nodes are subscribed to the topic. Since all communication is set up via the *roscore* node, there is a central point that has a complete view of all nodes that communicate with each other.

Retrieving the local time via ROS functions returns the current time, if the system is running live. However the huge benefit of using the ROS functions instead of the native operating system functions comes during playback. While playing back a recorded scenario, all nodes receive the same timestamps as during the recording. This simplifies the debugging process because the sensor data has the correct timestamps appearing to the processing nodes exactly like during recording.

ROS is used in our system for all camera related nodes and for the Visualization described in Section 1.4.

1.3.2 *LCM*

The LCM framework ([38, 26]) is a library for message passing on tightly coupled networks. Just like ROS it allows nodes in the system to communicate in a publisher/subscriber manner and handles the marshalling and unmarshalling of transmitted data (objects). Compared to ROS, it has the advantage that the whole framework is very small and has very few external dependencies. The LCM is used in our system for publishing the object positions and publishing the track information.

1.4 VISUALIZATION

Using many different sensors and algorithms creates a certain degree of complexity and makes tracking down bugs in the whole system very time consuming. In addition the sensor output may not be very intuitive, if the output is manually evaluated without the proper context. For example, the output of a single scan from the IBEO Lux is a series of (x, y, z) coordinates of points along the 4 scan layers. Even if they are rendered as a point-cloud, it is hard to distinguish features. In order to analyze the output of the sensors we developed a ROS node for overlaying sensor output over the camera image. This

helps debugging the obstacle detection, track detection and position estimation systems.

The general idea is to calibrate the camera used for visualization to the common coordinate system of the train, which will be explained in Section 3. Once this calibration is known, all objects that are detected can be projected into the visualization along with information that is useful for debugging, which is

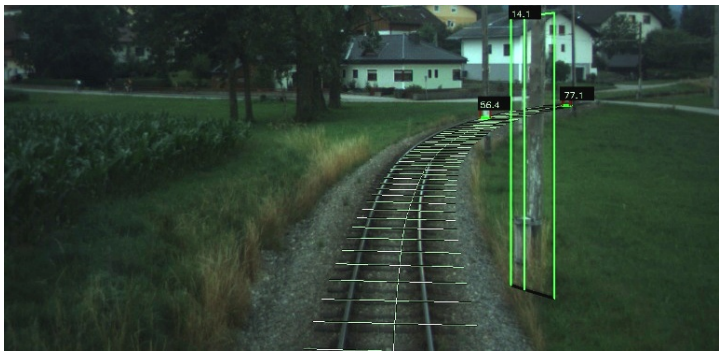
- A fixed size bounding box around the center of detected objects from the Ibeo sensor, illustrated in Figures 3a-3d. Since no three dimensional shape information is provided by the internal object detector, the objects are assumed to have an upright standing bounding box of 1×1 meter with the frontal face orthogonal to the ray from the sensor to the detected object.

- A variable sized bounding box from the Stereo sensor. It has the proper width and height of the objects that were detected.

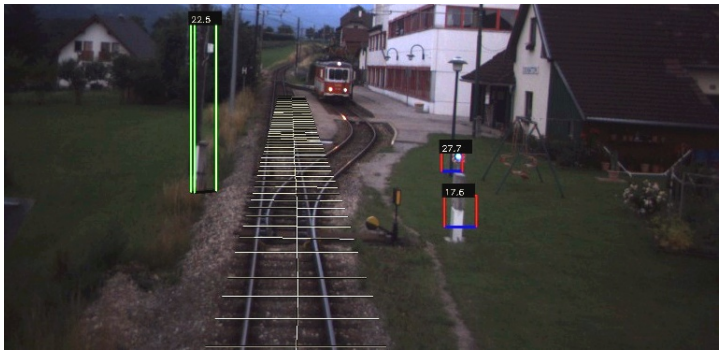
- A two plane approximation of the part of the convex hull that is visible from the Velodyne, illustrated in Figure 3a. In case of a cubic obstacle like a building this approximation evolves into an L-shape (when viewed from the top) in which the planes are perpendicular. For non-cubic objects it degenerates into a V-shape. For the purpose of visualization these shapes do not matter as they are projected into an image plane. However, the visualizer has to process the same data as the obstacle detection subsystem.

- The expected track used to verify the correctness of the self-localization. The projection is show in Figures 3a-3d. The track information is published either by a subsystem that extracts it from a known map based on the current train position, or by a subsystem that detects the track in the camera images in real-time which will be explained in Chapter 5. This track information is also used to make the distinction between obstacles and objects which will be explained in Chapter 4.

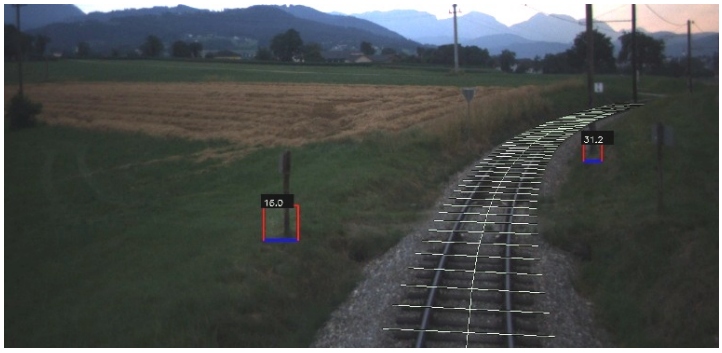
The visualizer is subscribed to the camera node through ROS , and to all nodes that provide object information and to the subsystem that publishes the track information through LCM. The visualizer itself broadcasts the augmented images on a dedicated network interface to prevent interference with the autonomous system. The reason for this is that it should make virtually no difference whether or not the visualizer is connected to the autonomous system. Otherwise we could not argue about the validity of the results, if the substantial amount of traffic generated by the visualizer delays packets between the sensors and the obstacle detection system.



(a) Ibeo obstacles (red boxes with blue base) and a mast (green) detected by the Velodyne sensor



(b) Ibeo obstacles (red boxes with blue base) and a mast (green) detected by the Velodyne sensor



(c) Obstacles detected by the Ibeo sensor (red boxes with blue base)

(d) Obstacle detected by the Velodyne and the Ibeo scanner

Figure 3: Visualization examples

Part II

CALIBRATION

2 CAMERA CALIBRATION

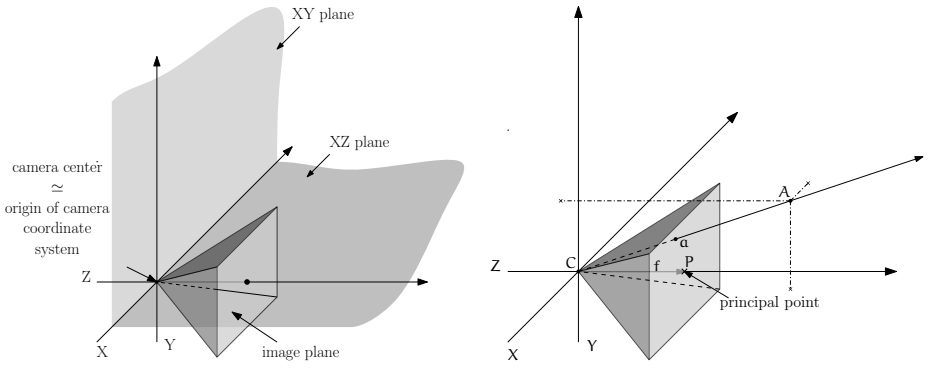
2.1 PINHOLE CAMERA MODEL

The relationship between points in the world coordinate system and measurements of those points by the camera (image points) is established by modeling the camera with the pinhole camera model. This widely known concept is often used to model a real world camera with commonly used optics (lenses). Using the pinhole camera model means applying a central projection to the points in the camera coordinate system. A point in the camera coordinate system, given as $A = (X, Y, Z)^T$ in Figure 4b, is projected to the point $a = (u, v)^T$ onto the image plane in Figure 4b. The projection is calculated as

$$(u, v) = \left(\frac{f \cdot X}{Z}, \frac{f \cdot Y}{Z} \right) \tag{2.1}$$

where f is called the *focal length* of the camera, $A \in \mathbb{R}^3$ and $a \in \mathbb{R}^2$. The line normal to the image plane passing through the origin of the camera coordinate system (also known as the camera center) is called the *principal axis* shown as the Z-axis in Figure 4a. The intersection between the *principal axis* and the *image plane* is called the *principal point* P.

This projection can be formulated as a matrix multiplication by embedding the points in homogeneous coordinates



(a) Camera coordinate frame (b) Perspective projection

Figure 4: Camera coordinate frame

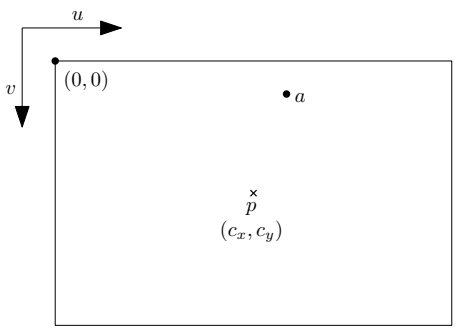


Figure 5: Image coordinate system

$$s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} s \cdot u \\ s \cdot v \\ s \end{pmatrix} = P \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.2}$$

where c_x, c_y are the image coordinates of the projection of the *principal point*. In real cameras, the origin of the image coordinate system and the image of the principal point do not coincide because u, v are indices into the two-dimensional array of pixels and thus can not be negative. Therefore, the image of the principal point is at$\sim$ $(\frac{width}{2}, \frac{height}{2})$ shown in Figure 5. As a convention, we will use an image coordinate system in which the horizontal axis u advances from left to right and the vertical axis v advances from top to bottom. This means that the origin of the image coordinate system is in the upper-left corner (Figure 5).

The 3×4 matrix P in Equation 2.2 is called a *projection matrix*. In this simple case, where the camera center is at the origin of the world coordinate system and pointing along the Z axis, the 3×3 sub-matrix from column one to column three is called *intrinsic camera matrix* K. If the camera is in a different place or if the camera is rotated relative to the world coordinate system the matrix K can not be extracted directly from P. In this case the projection can be formulated as

$$s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K\,[R|t] \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = P \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.3}$$

where $K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$, R is a 3×3 *rotation matrix* describing the rotation of the *camera coordinate* system to the *world coordinate system*

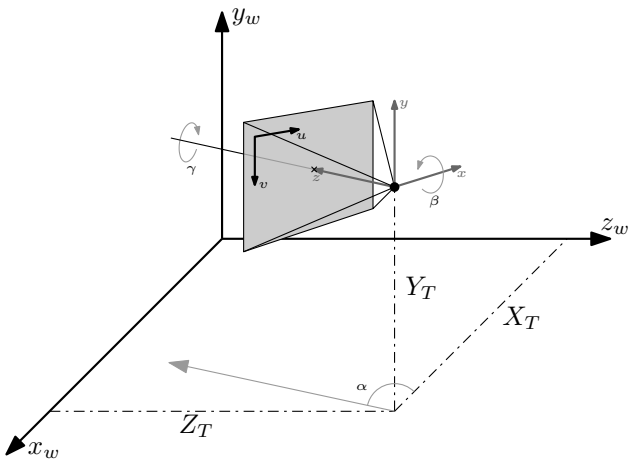


Figure 6: Extrinsic camera parameters. (X_T, Y_T, Z_T) is the translation vector from the *world coordinate system* to the *camera coordinate system*. α, β, γ are the angles for *yaw, pitch* and *roll.*

and $t = -RT$ where T is the translational offset of the camera center from the origin of the *world coordinate system*.

As we can see there are three coordinate systems that are of interest

1. *World coordinate system*
   Coordinates are relative to a world origin and along the world axis. This may be a common reference point for all sensors, the origin of one of the sensor coordinate systems or any other arbitrarily chosen point that is well known.

2. *Camera coordinate system*
   Coordinates are relative to the camera center. The z-axis or the y-axis are usually normal to the image plane. The *camera coordinate system* relates to the *world coordinate system* by a rotation and a translation of the coordinates (Figure 6).

3. *Image coordinate system*
   Coordinates are relative to the origin of the image coordinate system. This origin is usually the *upper-left* or *lower-left* corner of the image sensor. The *image coordinate system* relates to the *camera coordinate system* by a *perspective projection* of the point in the *camera coordinate system* to the points in the *image coordinate system*.

The *intrinsic camera parameters* K and the *extrinsic camera parameters* R, T are all the information needed to project a given point from the *world coordinate system* to the *image coordinate system*.

2.1.1 *Re-projection error*

During the calibration of a camera sensor one needs to know the quality of the calibration results. Given the estimated calibration pa-

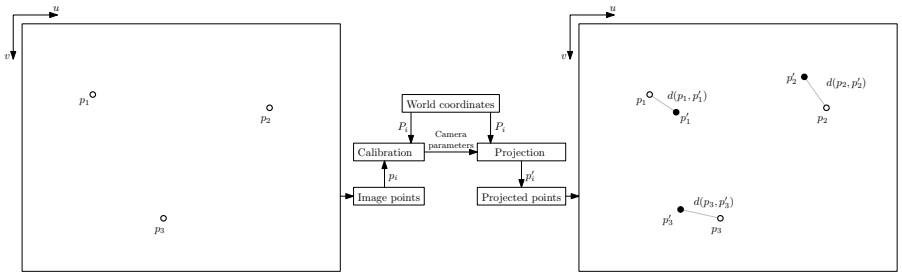


Figure 7: Calculation of the re-projection error

rameters and the input data, a metric should measure the accuracy of the parameters. The calibration process uses measured points in the image and their corresponding *world coordinates* to calculate the camera parameters. In most cases, where the measurements of the points are subject to errors, the calibration does not yield an exact result, but only an estimate. These estimated parameters are then used to *(re-) project* the *world coordinates* into the *image coordinate frame* (Figure 7). Due to the measurement errors the projected points won't overlap with the measured points exactly. Using the euclidean distance to measure the error between *projected points* and *real points* leads to the *re-projection error*. It is the sum of squared euclidean distances between the *real points* and the *projected points* (Equation 2.4).

$$\sum_i d^2(p_i, p'_i) \tag{2.4}$$

This *re-projection error* depends on the number of points used, which would cause problems when different calibration scenarios are compared. For this reason we will use the *average re-projection error* in this work to compare the results. The *average re-projection error* is the arithmetic mean (Equation 2.5) of all Euclidean distances of p_i and p'_i

$$\frac{1}{N} \sum_i d^2(p_i, p'_i) \tag{2.5}$$

where N denotes the number of points.

2.2 CALIBRATION

To calibrate the camera parameters, we are using the methods described in [24]. The basic idea is to find enough points in the image for which the world coordinates are known to set up a system of linear equations which is then solved to determine the projection matrix P (Equation 2.3). The projection matrix we are using has 11 degrees of freedom ([24]) which requires 6 point correspondences since every point creates two equations. However, this would only be true if all point correspondences were without error. This is not the case in real world calibration scenarios. To create a robust calibration, the num-

ber of point correspondences should be much higher. The resulting system of linear equations is then solved in a *least squares* manner and provides results that have a certain degree of robustness against measurement errors.

The general approach towards intrinsic and extrinsic camera parameter estimation of any visible-light camera is to use some sort of calibration pattern with known dimensions to establish object to pixel correspondences [24]. Usually, this is accomplished by using a black/white checkerboard pattern mounted on a planar calibration board. The pattern is recorded by the camera at varying positions. Detecting the corners of each of the squares provides the required world to image correspondeces that are used to estimate the *intrinsic parameters* as well as the geometric distortion introduced by the lens. Given the *intrinsic parameters* are already available, it is possible to determine the extrinsic parameters, that describe the position and orientation of the camera reference frame with respect to a known world coordinate frame.

Detecting a calibration board in front of a fixed camera is technically the same as moving a camera in front of a fixed calibration board. To establish a useful relationship between the different positions and the calibration board, the algorithm must be able to detect the same points on the calibration board for which the *world-coordinates* are known.

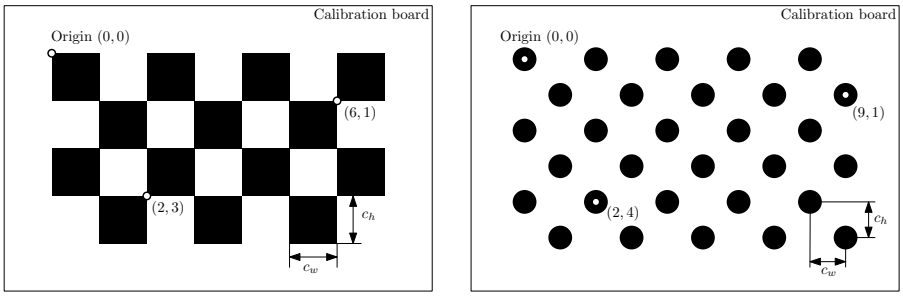
2.2.1 *Checkerboard pattern*

The whole calibration process relies on a robust detection of the calibration board ([45, 42]). The most common calibration pattern is a checkerboard (Figure 8a). In this pattern the corners between white and black fields are detected. Every corner is assigned a *world coordinate* corresponding to the position within the pattern. This is very simple since one only needs to count the horizontal and vertical corners and assign each corner those two numbers as x and y coordinates. The z coordinate is the same for all points because the calibration board is a planar object. Without loss of generality, the z coordinate is set to zero.

An example for a *world coordinate* in the "checkerboard coordinate system" is $(X_c, Y_c, 0)$. These coordinates are further scaled according to the real size of the checkerboard fields, c_w (width) and c_h (height)

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} c_w \cdot X_c \\ c_h \cdot Y_c \\ 0 \end{pmatrix} \tag{2.6}$$

which yields the final *world coordinates* of the checkerboard corners (X, Y, Z). The algorithm now has a correspondence between *image coordinates* and *world coordinates*, which is required to calculate the



(a) Checkerboard (b) Circle-pattern

Figure 8: Calibration patterns

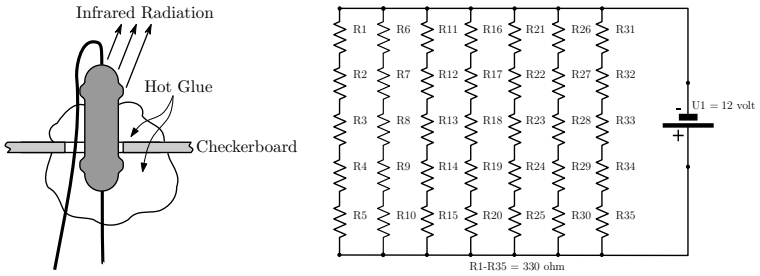
projection matrices ([45]) and from that the *intrinsic* and *extrinsic camera parameters*.

Another widely used pattern for camera calibration is a grid of circles as shown in Figure 8b. The calibration algorithm works just like when a checkerboard pattern is used. The centers of the circles are placed on a well known grid which can be converted to *world coordinates*. Once the circles are detected, their centers are converted to preliminary *world coordinates*, which are then converted to real world coordinates by scaling them according to the spacing between the circle centers (Equation. 2.6).

2.2.2 *Far-infrared camera calibration*

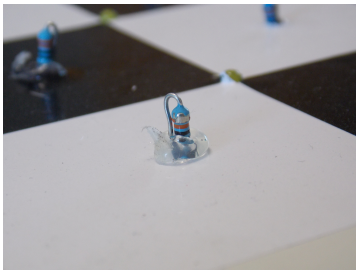
Calibrating a *far-infrared camera* is a bit more involving than calibrating a *visible light camera* for which we chose a black/white checkerboard pattern mounted on a planar (metal) surface as a basis for calibration. This enables an easy estimation of the intrinsic parameters of the visible-light camera by using the method of Zhang [59]. However, the *far-infrared camera* is unable to detect the black/white checkerboard. In order to use the same approach to determine the *intrinsic* Infrared (IR) camera parameters, the calibration board has to be augmented with a pattern visible to the IR camera.

Our approach to tackle this problem is to install a pattern of electrical elements emitting IR radiation. Using Light-emitting Diode (LED)s to create a detectable pattern in the visible and IR spectrum does not work because the thermal difference to the calibration board is too low, even though LEDs have an efficiency level of ≈ 20 percent, leaving enough energy for thermal radiation. As an alternative strategy, we decided to use a set of resistors mounted in the centroid of each square (see Figure 9a, 9c and 10). This has the advantage that the geometric corners of the black/white squares are not physically distorted by the resistors. Consequently, the corner points in the image of the visible-light camera can still be automatically detected (using the algorithm

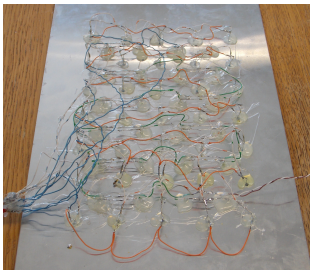


(a) Resistor mounting

(b) Circuit diagram



(c) Resistor mounting on the real board



(d) Wiring of the board

Figure 9: Electrical and physical properties of the calibration board

of Vezhnevets, implemented in the OpenCV library [3]). Calibration is not negatively affected.

Regarding the physical configuration of the calibration board, we install an evenly spaced array of 35 (7×5) resistors connected to a 12 volt DC power supply (Figure 11). Each resistor is a 330 Ohm metal film resistor. The circuit diagram (see Figure 9b and the corresponding wiring in Figure 9d) shows that there are seven rows attached in parallel to the power supply. Each row consists of 5 resistors connected serially. After adding all wires, the whole circuit draws ≈ 1.12 ampere and thus emits ≈ 13.44 watts of thermal radiation.

By looking at the two exemplary IR images showing the calibration board in Figure 12a, one may notice that simple gray-value thresholding to detect the resistors will not work reliably for the following reason: the person holding the calibration board emits much more IR radiation than the resistors which eventually leads to a broader range of intensity values. This makes it impossible to detect the resistor spots by means of gray-value thresholding alone. One could argue that this problem could be solved by using a mounting bracket. However, such a solution is usually constrained to a laboratory environment.

The basic idea of our approach is to exploit the regularity in the spacing of the resistors (in both directions) to reliably detect the pattern. The algorithm consists of two steps which are executed iteratively and one finalization step in which the most suitable result is picked and the remaining resistors are identified. In one iteration step, a gray-value thresholding is performed to identify resistor candidates

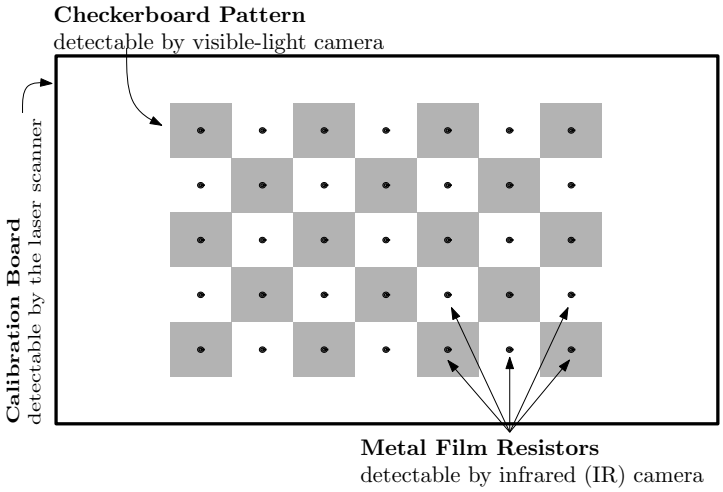


Figure 10: Schematic of the calibration board with one resistor mounted in the center of each checkerboard square.

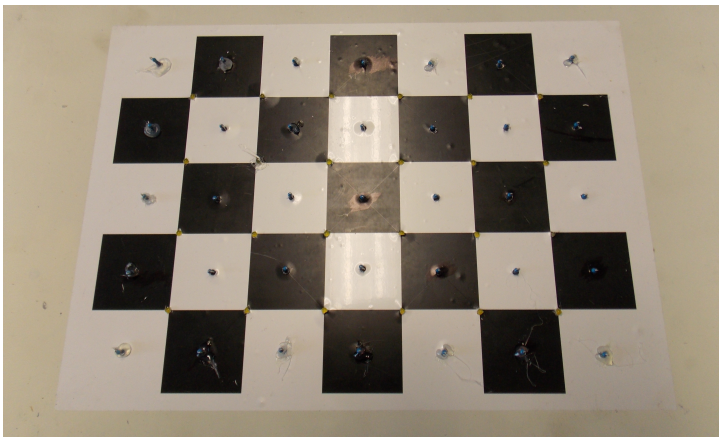
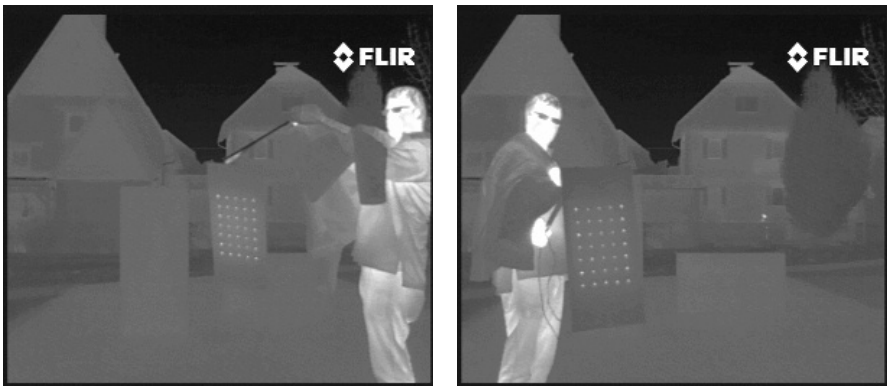
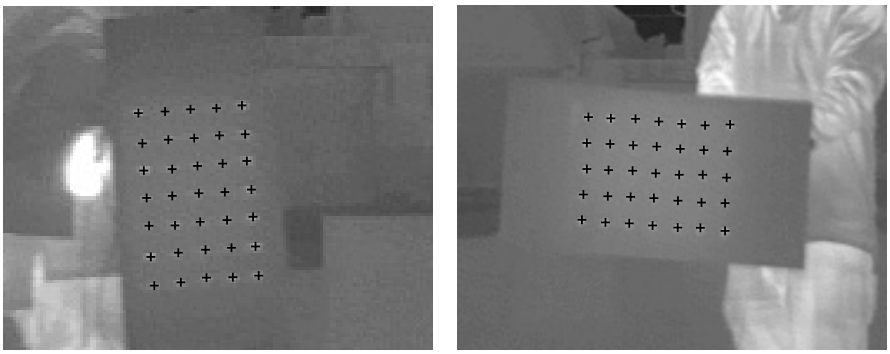


Figure 11: Real calibration board.



(a) IR images with calibration board



(b) Detected resistors (black crosses)

Figure 12: Exemplary IR images showing calibration board and the 7×5 resistor pattern.

and the regularity of the candidate resistor pattern, which most likely corresponds to the true resistors, is measured.

Candidate search

In one iteration step, gray-value thresholding with a threshold chosen from a predefined range, is performed. Since the intensity in IR images can differ significantly depending on the environment, the idea is to increase the gray-value threshold $t \in [0, 1]$ in each iteration in order to find a setting where the resistors can at least be segmented from the metal calibration board. In that case, a high regularity in the identified candidate resistors is expected, which will be measured in the second step. The range of thresholds $R := [a, b], a := t_O - \epsilon, b := t_O + \epsilon$ is determined by Otsu's [40] intensity threshold t_O and a heuristically determined value ϵ [1]. After the IR image is binarized, all connected regions with an area larger than 50 pixels are removed and the region centroids are calculated. Since the calibration board has to be positioned at least six feet away from the sensor, any region larger than 50 pixel is most likely not a resistor. The centroids, denoted by the vertices $V_i \in \mathbb{R}^2$ represent the resistor candidates and are the input to the next step.

1 a value of $\epsilon = 0.25$ has shown to work well in practice

Measuring Regularity

In the second step of one iteration, the Delaunay triangulation T of the set of vertices V_i is computed. The Delaunay triangulation has the property that the minimum interior angle among all triangles is the greatest possible among all triangulations. Figure 13a shows the part of the Delaunay triangulation which contains the $x \times y$ resistor pattern (illustrated as black dots). Let us consider the $2(x-1)(y-1)$ triangles which triangulate this pattern. A triangle is a *neighbor* of another triangle if they share a coincident edge. It can be observed that the $L = 2(x-3) \cdot (x-3)$ inner triangles (shaded gray) share a common property: all three immediate neighbors of each of the L triangles as well as their immediate neighbors and the center triangle itself have approximately the same perimeter (Figure 13b). We explicitly focus on the L inner triangles, since considering all $2(x-1)(y-1)$ triangles would not allow to establish a rigorous neighborhood criterion. In case of the border triangles of the resistor pattern for instance, the criterion does not hold. Obviously, the inner triangles share other properties as well, but the perimeter led to the most robust detection results in the experiments.

The computational steps to identify the L inner triangles can be summarized as follows: For each of the triangles Δ_j in the triangulation T, compute the maximum perimeter difference to all of its nine neighbors. Formally, given that $p(\Delta_j)$ denotes the perimeter of triangle Δ_j and $\Delta_{jn}, n = 1, \ldots, 9$ denote the nine neighbors, calculate

$$m_j = \max_{n=1,\ldots,9} \{|p(\Delta_j) - p(\Delta_{jn})|\}. \tag{2.7}$$

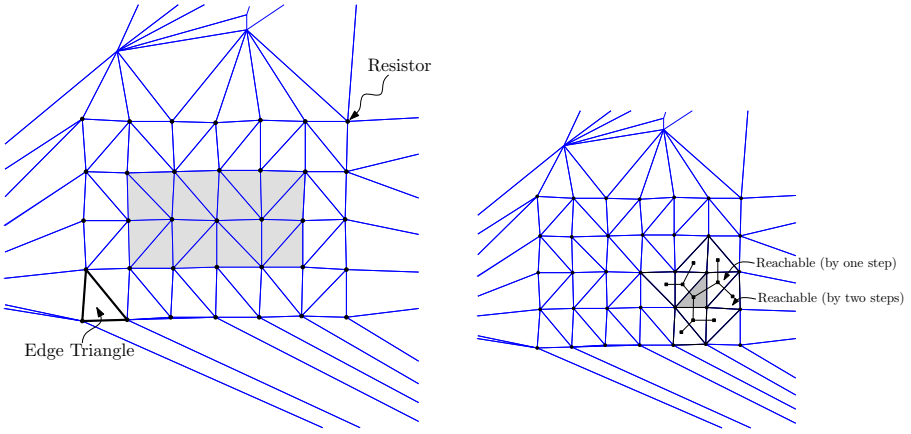
In case a triangle has less than nine neighbors the maximum perimeter difference is set to infinity. Next, the variance σ^2 of the L smallest values of $m_1, \ldots, m_M$, is determined, where M denotes the total number of triangles in the triangulation T. This variance is a measure of regularity among the triangles with the smallest maximum perimeter difference to its nine neighbors. Let this set of L triangles be denoted as $\{\Delta_1, \ldots, \Delta_L\}$. Now, given that $v(\Delta_j)$ returns the set of vertices that span Δ_j, we can require that the cardinality of the set

$$S = \bigcup_{j=1}^{L} v(\Delta_j) \tag{2.8}$$

is $(x-2)(y-2)$. Otherwise, the detection result cannot be valid, since ther are only $(x-2)(y-2)$ resistors in the pattern.

Resistor Identification

Depending on the threshold step-size s, $\lceil (b-a)/s \rceil$ iterations of the *candidate search* and *regularity measurement* step are performed. Letting the subscript i denote the iteration index, the configuration (t_i, S_i) where the variance σ_i^2 is minimal will eventually be picked. The final steps to identify the remaining $K := xy - (x-2)(y-2)$ resistors are



(a) Illustration of the *inner L* triangles (light gray)

(b) Neighborhood and reachability from a center triangle (dark gray)

Figure 13: Illustration of an exemplary Delaunay triangulation as well as the neighborhood relations.

straightforward. The boundary vertices of the $(x-2)(y-2)$ identified vertices are picked and all triangles attached to these vertices are determined. The union of the triangle vertices then gives at least $K-4$ of the K remaining resistors. Depending on the triangulation T, we might however miss the four corner vertices. This can happen because the triangles containing a corner vertex might not be attached to any of the $(x-2)(y-2)$ vertices found by means of the regularity criterion. One of these triangles is illustrated in Figure 13a. The final four resistors are found by performing a nearest neighbor search around the intersection points of the edges determined by the outermost vertices. Figure 12b shows two exemplary detection results.

Considering the fact that there is a boundary of metal around the resistors, the next resistor candidate outside the $x \times y$ pattern does not affect the robustness of this approach. However potential disturbances can occur in case one threshold setting cannot identify all resistor centroids. This is obviously possible due to thermal differences of the resistors which can show up as considerable intensity variations.

3

SENSOR TO VEHICLE CALIBRATION

Our autonomous vehicle is equipped with several sensors of different categories (camera, laser, ...). When they are mounted to the train, the relation between their sensor coordinate system becomes fixed, disregarding minimal changes due to vibrations. As explained in Chapter 2, the orientation and translation of the camera coordinate system is described by the *extrinsic camera parameters* (Figure 6). The same is true for all other types of sensors, although we will call them just *extrinsic parameters* for non-camera sensors.

Knowing the *extrinsic parameters* between two sensor coordinate systems enables us to relate their sensor measurements to each other, also called *sensor fusion*. This is necessary to

- *accommodate sensor noise*
  All measurements are subject to noise. Combining measurements from different sensors allows us to improve the accuracy of the measurements.

- *complement measurements between sensors*
  The measurements of a single sensor are confined to a certain region. For example, a front-facing radar can only detect objects of a certain material in a certain field of view in front of the vehicle. If the field of view or the types of objects that are detectable in front of the vehicle need to be expanded additional sensors have to be added.

- *manage false positives*
  There is always a chance that a sensor provides a measurement for an object that is not really there. One example is electromagnetic interference that can lead to spurious measurements from the *Velodyne* scanner. To filter such false positives, it is necessary to have several sensors with overlapping regions, as it is very unlikely that several sensors are affected in exactly the same way by the effect that created the spurious measurement.

However, pairwise calibration of each sensor pair that needs to share information is however not necessary. Instead, every sensor coordinate system will be calibrated to a common vehicle coordinate system. This should be a point that can be easily accessed when measuring the distances to the origins of the sensor coordinate systems. It is also useful if the origin of the vehicle coordinate system has properties that can later be used. For example, a useful origin of the train coordinate system is directly above the pivot point of the frontal axle P_{zero} (Figure 14). This point has the following properties

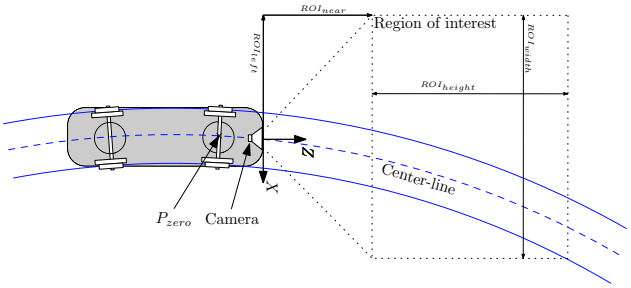


Figure 14: Train coordinate system with origin P_{zero}

- It moves along the center-line of the track. This means that P_{zero} has the same normal-distance to both rails.
- Even if the swinging of the train can not be neglected, P_{zero} is the point on the train that is least affected by it.
- In our case the frontal axle can be reached through an access hatch from within the train. This hatch is directly above the pivot point of the axle and thus the position of P_{zero} can easily be determined from within the train.

Another example for the origin of the train coordinate system would be the location of the Global Positioning System (GPS) antenna. However, if the GPS antenna is not mounted permanently on the vehicle, or if the antenna is kept in place only by magnets, the choice of the origin should be a different point on the vehicle.

Once a good origin is selected, the extrinsic calibration of the sensors can be performed. This is done in one of three ways

- Some sensors may be calibrated manually, for example by measuring the offset of the GPS antenna to the origin of the *train coordinate system*.
- Most sensors are calibrated coarsely by gauging the *extrinsic parameters*, then sensing a well known object and manually modifying the extrinsic parameters to fit that measurement. This is the easiest way to do extrinsic calibration for sensors where no automatic or semi-automatic method is available. However experience shows that these results are almost always inferior calibration parameters than would be possible with *automatic* or *semi-automatic* calibration. This is simply due to the fact that *automatic calibration* may run hundreds or thousands of update steps until the calibration is sufficiently accurate.
- The camera sensors are calibrated in a *semi-automatic* way. We will use two different methods for extrinsic camera calibration. One method requires the placement of markers in the scene, which are measured relative to the origin of the train coordinate system (subsequently called *train origin*), and another method is to rely on special properties of the railroad scene.

3.1 CAMERA TO VEHICLE CALIBRATION

The whole calibration process should be as simple and streamlined as possible, especially when we are working on a test system that requires the sensors to be mounted on the vehicle every time a test-drive is scheduled. The whole calibration process can be split into two independent stages, just like the camera calibration parameters consists of two parts (extrinsic and intrinsic parameters). The intrinsic calibration (Section 2.2) is generally the more tedious task since one needs to get close to the camera or use impractically big calibration boards. Fortunately this step needs to be done only when something directly related to the camera is changed (e.g., different camera sensor, focal length). Normally this is not the case. Thus the *intrinsic parameters* can be calibrated while the camera is not mounted on the vehicle. For example, the calibration can be performed in the office and remains valid even if the extrinsic parameters change.

3.1.1 *Extrinsic calibration through point correspondences*

The most generic method to calibrate extrinsic camera parameters is to find correspondences between known world points and image points. Normally, this is done by acquiring images of a three-dimensional calibration object (cube). In a static scene, this can be simplified by measuring the X, Y, Z distances of several points in the scene relative to the *train coordinate* system. It does not matter if those measurements can not be made at the same time, but over the course of several frames, since neither the points nor the camera are moving.

This process requires several steps to acquire the *3D train coordinates* of the calibration points and also requires manual extraction of the *2D image coordinates* of those points. A few things can be simplified in our train scenario, if the calibration is done in the train station. The two rails are perfectly parallel and also parallel to the Z axis of the train coordinate system, with a rail width of exactly *one meter*. Furthermore we assign the top of the rails a Y coordinate of *zero.* The origin of the train coordinate system is the pivot point of the frontal axle and located perfectly between the two rails. The calibration process is as follows

1. Measure the normal distance of the center of the train coordinate system to the front end of the train

2. Select an arbitrary point on one rail and measure the distance (along the rail) from this point to the front end of the train. This can be easily achieved with a laser range meter. In addition with the offset of the front of the train from Step 1 this distance is the Z coordinate of the point in *train coordinates*

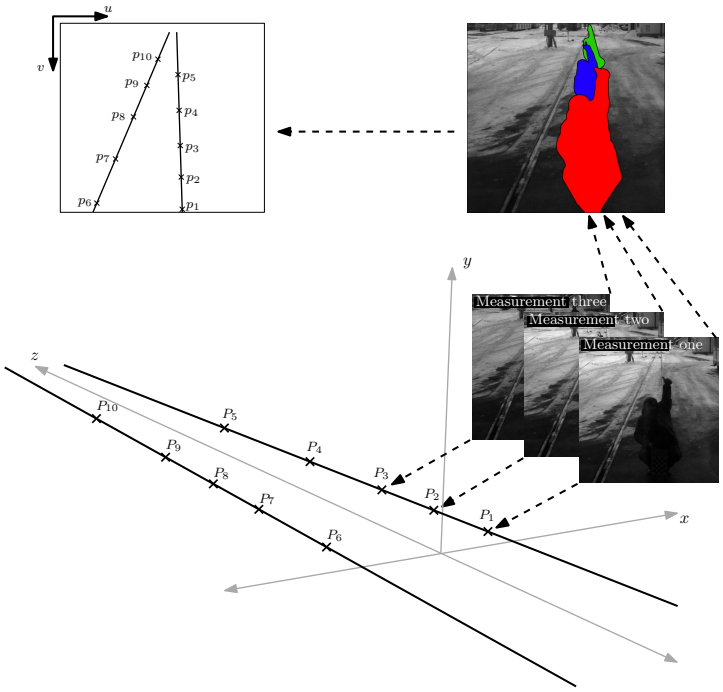


Figure 15: Extrinsic calibration by taking several measurements of points in the scene

3. The X coordinate of all points on the left rail is -0.5 and the X coordinate of all points on the right rail is 0.5, because the origin lies exactly between the two rails

4. The Y coordinate of all points on the rails is, as by our convention, 0.0

5. Continue with Step 2, until enough points ($\sim 10 - 20$) on both rails are known

6. Select the image coordinates of the measured points from the recorded frames. This is greatly simplified by placing some kind of marker on the rails for the currently measured points

The result is a list of pairs of *image coordinates* and *train coordinates.* Those correspondences are the input to the calibration algorithm in [3].

3.1.2 *Extrinsic calibration through parallel lines*

The calibration method described in Section 3.1.1 is a generic solution and can easily be modified to fit other scenarios. The disadvantage is the complexity of the calibration process. Since our tests are done during normal operation on the railroad, keeping the time it takes to setup the system at a minimum is a crucial requirement. The sensor systems must be up and running in a short period of time and

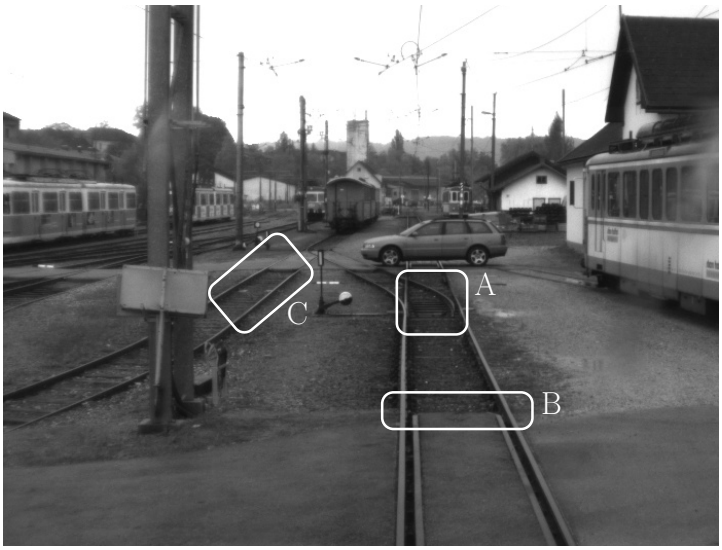


Figure 16: Possible features which we can not rely on

the calibration has to be done on the vehicle. Numerous calibration mechanisms exist that exploit simple constraints in the projections. For example, calibration based on vanishing points [21, 1], several coplanar lines [48] or coplanar circles [5]. However those calibration methods do not apply to our scenario, because the scene does not contain coplanar circles. Although we have a very prominent vanishing point (the intersection of the two rails), we are missing a second vanishing point that can be reliably extracted from the image. For a robust calibration, the algorithm would need two vanishing points that are extracted from structures in the scene that are orthogonal to each other. If we use the rails as the first pair of parallel lines, we would need to extract two other pairs of lines from region A in Figure 16. It is clearly visible that this can not be done in a reliable way, because the resolution of the camera is not sufficient. But even if it was possible to extract those points the vanishing point would be a bad candidate for calibration. The lines between the rails are almost parallel to the image plane. This results in a vanishing point (intersection of the lines) that is numerically very unstable (approaching infinity). We can see from region B in Figure 16 that there may not even be lines at all, if the vehicle is in a region where the track is embedded in concrete.

A remaining candidate for our scenario is the calibration based on four coplanar lines. Region C in Figure 16 shows that such lines would in fact be available. However, this depends heavily on the position of the train in the station and would not work in other stations along the track, because there is often no second track. Furthermore, if the camera calibration is done from recorded data, the distance of those tracks is not known without going back to the scene and taking the measurements. And those tracks may not be visible at all, because another train is blocking the line of sight. Since an important

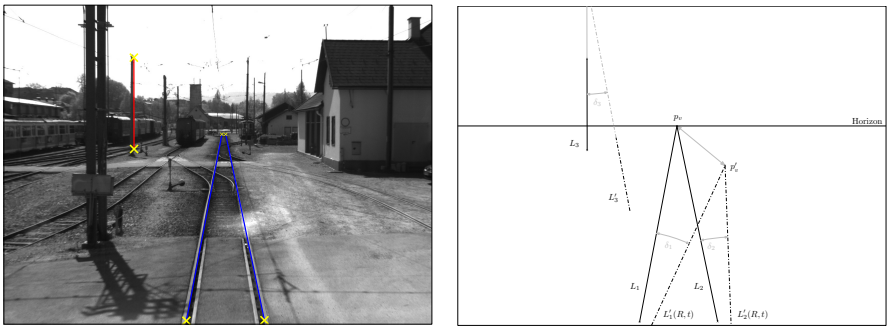


Figure 17: Calibration using three lines

requirement is not to interfere with normal traffic, it is not possible to freely move any train to our needs.

Therefor we need to design our own calibration technique that is specifically tailored to this scenario. Some things about the scene in the train station are known and will not change even over time.

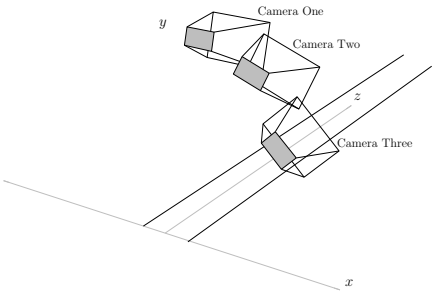
- The distance of the rails is *one meter*.
- The rails are perfectly parallel and extending along the Z axis, if the train is standing in the train station
- The electricity-poles are standing orthogonal to the plane formed by the two rails. This is only true if the ground plane has zero ascending slope.
- The top of the rails has per definition a Y coordinate of zero

The operator calibrating the cameras has to select six points in the image, namely two points on each, the left and right rail, and two points on an electricity-pole or other vertical structure to the left of the tracks.

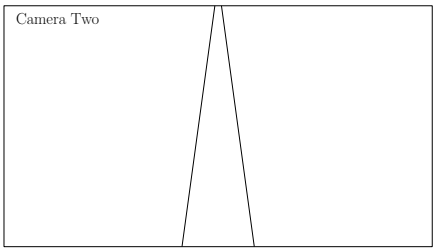
Using this information, we are able to define a simpler calibration method based on an error metric that compares the vanishing points p_1, p_1' and the angles of the two rails L_1, L_2 with two projected rails L_1', L_2'. The projected rails have a distance of one meter and are parallel to the Z axis with an X coordinate of $0.5m$ and $-0.5m$ respectively.

$$\begin{aligned} \underset{R,t}{\operatorname{argmax}} \quad & 10^2 \cdot ((L_1 \angle L_1'(R,t))^2 + (L_2 \angle L_2'(R,t))^2) + \\ & d^2(In(L_1, L_2), In(L_1'(R,t), L_2'(R,t)) \end{aligned} \tag{3.1}$$

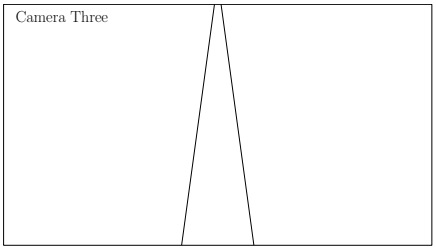
where $L_1'(R,t)$, $L_2'(R,t)$ are the projections of the left and right rail into the *image coordinate system* using Equation 2.2, R is the estimated 3×3 *rotation matrix* and t is the estimated three dimensional *translation vector*. $In(L_1 L_2) \in \mathbb{R}^2$ is the intersection of the two lines L_1 and L_2 , which are the image of the left and right rail and $In(L_1'(R,t), L_2'(R,t)$ is the intersection of the projection of the left and right rail, using the estimated calibration parameters (R,t).



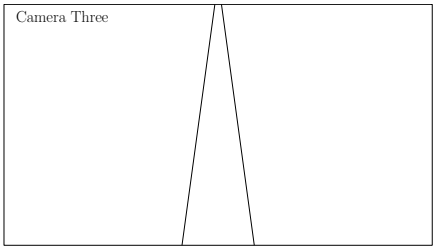
(a) Different camera parameters which produce the same image of the rails



(b) Image of the first camera



(c) Image of the second camera



(d) Image of the third camera

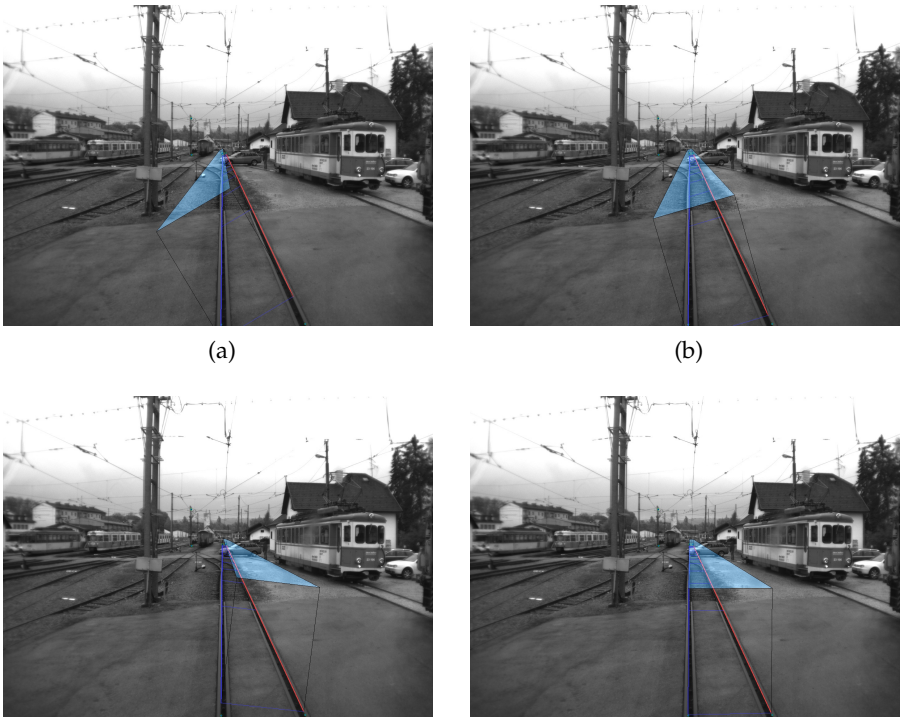
Figure 18: Different camera calibrations may result in the same projection of the two rails

This does not necessarily lead to usable calibration parameters because there is an infinite number of calibrations that minimize this error function. Three different camera parameters that produce exactly the same projection (Figures 18b-18d) of the rails are shown in Figure 18a.

In a second step the algorithm tries to determine the correct *roll* parameter, which in our case is a rotation around the Z axis of the *camera coordinate system*. Based on the previous parameter estimation, a range of *roll* parameters is evaluated by minimizing an error function w.r.t the fixed *roll* parameter.

$$\begin{aligned} \underset{R_{roll},t}{\operatorname{argmax}} \quad & 10^2 \cdot ((L_1 \angle L_1'(R_{roll},t))^2 + \\ & (L_2 \angle L_2'(R_{roll},t))^2 + (L_3 \angle L_3'(R_{roll},t))^2) + \\ & d^2(In(L_1,L_2), In(L_1'(R_{roll},t), L_2'(R_{roll},t)) \end{aligned} \tag{3.2}$$

The line L_3 is a vertical pole or mast in the image that must be located to the left of the tracks. R_{roll} is a rotation matrix with a fixed *roll* parameter. The algorithm first evaluates 40 coarse steps around the *roll* parameter calculated in step one in a range $[-1, 1]$ radians which is shown in a few selected images in Figure 19. The best *roll* parameter is selected and 40 finer steps in the range $[-0.05, 0.05]$ around this *roll* parameter are evaluated. In the last refinement the best *roll* parameter



(a) (b)

(c) (d)

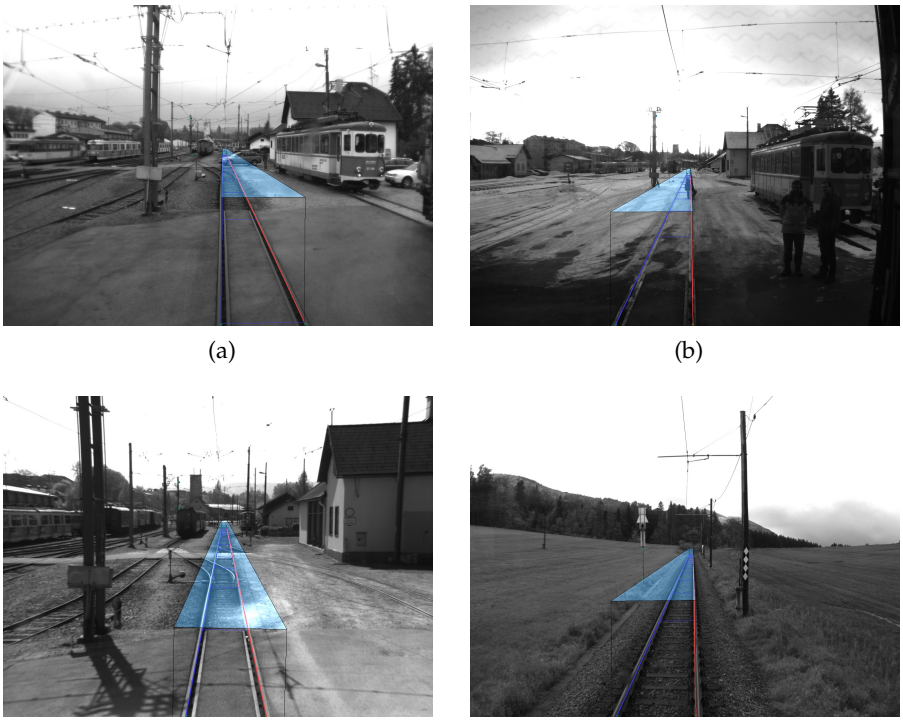
Figure 19: Projection of a small tunnel based on four intermediate roll parameters of the second calibration step

in the range $[-0.0025, 0.0025]$ around the previous one is selected as the final result.

The results of the calibration process is shown in Figure 20, which is a projection of a tunnel into the image, that is supposed to be as wide as the tracks and standing vertical to the ground plane. Figure 20a shows the calibration of a *Basler scout scA1300-32gm* mounted in the left part of driver's cab. Figures 20b and 20d are also from a *Basler scout* but mounted on the right side of the driver's cab, and Figure 20d also shows that the calibration works on a sufficiently straight part of the regular track. For Figure 20d, the railroad sign to the left of the track was used as the vertical structure. Finally, Figure 20c shows the calibration of a *Bumblebee XB3* mounted on top of the train.

3.1.3 *Joint LIDAR and camera calibration*

Calibrating multiple sensors to each other, requires that we use a calibration pattern visible to all sensors. In our setup, we thus need a pattern visible to a 3D laser range scanner, a forward-looking IR (FLIR) camera and a visible-light camera. Due to the extensive existing work [59, 52, 46, 14] on the calibration of visible-light cameras, our strategy is to keep the black/white checkerboard pattern mounted on a planar



(a) (b)

(c) (d)

Figure 20: Visualization of the calibration result in four different scenes with different cameras and different lenses

surface and to augment it by a set of low-cost electrical components to facilitate IR camera calibration.

The goal is to provide an easy to use calibration mechanism that can be used in the field by minimizing the constraints on the scenery. It is required that on-site changes to the test vehicle can be dealt with immediately rather than calibrating the system in the office.

3.1.3.1 *Lens Distortion*

To calculate accurate and robust radial (and tangential) distortion parameters, we would have to have calibration points throughout the whole image (especially points close to the image borders). This could either be achieved by moving the board close to the camera or by using a large calibration board. Unfortunately, moving the board close to the camera leads to unusable laser returns since a 3D range scanner often only provides reliable range measurements for objects at a distance of at least six feet (see Section 3.1.4). Consequently, it would not be possible to use the same set of calibration images for *intrinsic* and *extrinsic* calibration. We would inevitably need two separate sets; one which is solely used for the calibration of *intrinsic camera parameters* and one which is used for the calibration of *extrinsic camera parameters*.

Using a large calibration board as a potential solution to this problem is quite impractical, though, especially in outdoor environments and in situations where the sensor equipment is not permanently mounted on the vehicle and requires frequent recalibration.

In our setup, visual assessment of the camera distortion has shown that there is hardly any distortion in the area of interest, which is at least six feet in front of each camera and is relatively narrow as well. For that reason, we decided to omit estimation of the lens distortion, however, we emphasize that this step might have to be incorporated depending on the cameras. In that case, we favor the solution of using two separate image sets due to the lower practical impairment.

3.1.3.2 *The 3-D Laser Range Scanner*

To establish a correspondence between the measurements of the 3-D laser range scanner and both camera systems, we need to automatically detect the 3-D coordinates of the calibration board corners. This is a non-trivial task due to the inherent noise in the range measurements [15]. The problem can be split into two parts: Firstly, we have to identify the point cluster corresponding to the range measurements returned from the calibration board. Secondly, we have to fit a rectangle with the dimensions of the calibration board to the identified point cluster.

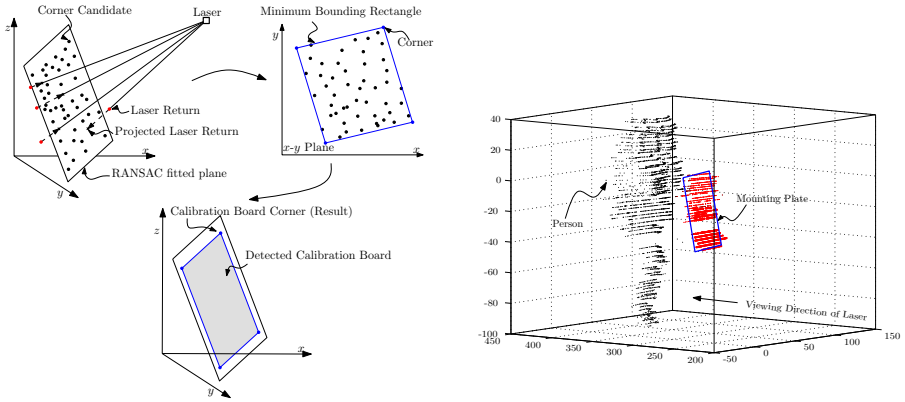
Point Cluster Identification

In order to reliably identify the 3-D point cluster corresponding to the range measurements from the calibration board, we physically simplify the task by using a metal stick to hold the calibration board (fixed by means of a vacuum cup). This allows simple distance thresholding to separate the range measurements off the board from range measurements off the person holding the board. Otherwise, it would be significantly harder to automatically separate the point cluster of interest. We further note that the position where we can present the board to the laser scanner is constrained by the smallest field of view angle of one of the cameras, in both vertical and horizontal direction. Given that we know the length of the stick, these constraints allow for the automatic identification of the point cluster. An exemplary segmentation result is shown in Figure 22b, where the point cluster corresponding to the calibration board is shown in red.

Detecting the Corners

Regarding the second part of our problem, the identification of the corner coordinates, we have to cope with the aforementioned sensor noise and the fact that the board might be tilted as well. Since we want to allow outdoor calibration, we need a strategy that can handle possibly occurring outliers caused by reflections as well. In order to cope with these impairments, we rely on the well-known Random Sample Consensus (RANSAC) algorithm [11] to fit a plane to the point cluster. The points are then projected onto the fitted plane along the viewing direction of the laser scanner. Next, we rotate

Figure 21: Boundary of the calibration board projected into the IR images based on the automatically calibrated intrinsic and extrinsic IR camera parameters.



(a) The three steps (b) Detection/fitting result

Figure 22: Illustration of the (three) steps to find the calibration board in potentially noisy laser range measurements.

the projected points to the x-y plane and determine the minimum bounding rectangle (i.e. the bounding box of the convex hull of the points). Rotating the corner points of the bounding rectangle back to our original coordinate system yields the 3-D coordinates of the calibration board's corners. All three steps are illustrated in Figure 22a.

We emphasize that we can exploit our knowledge about the dimensions of the calibration board in the corner detection step of our algorithm to validate the quality of the plane fit. Allowing only a certain deviation from the true rectangle area[1] immediately excludes cases where the RANSAC algorithm followed by the projection and rotation step led to a wrong minimum bounding rectangle. In Figure 22b, we show an exemplary result of fitting the calibration board to the laser range measurements. The data shown in this figure corresponds to the rightmost IR image in Figure 21. Due to a missing ground truth, we have to rely on visual validation of the result.

3.1.3.3 *Extrinsic Calibration*

For each calibration image, we estimate the extrinsic parameters of the cameras relative to the calibration pattern (Section 3.1.3). The

1 in our experiments, we allow a deviation of 3.9 square inches

distance from the calibration points to the corners of the calibration board is known. This allows calculating the image coordinates of the corners of the calibration board by projecting the four points into the image, illustrated in Figure 21, using the previously calculated extrinsic parameters. Consequently, we have knowledge about the 2-D image coordinates and the 3-D coordinates of the corners of the calibration board in all images. The extrinsic camera parameters relative to the laser are then estimated by minimizing the *re-projection error* (using the OpenCV library). The position and rotation of the cameras relative to the laser coordinates is fixed. This means that we can use the 2D-to-3D point correspondences of every calibration image together, as if we would be using a much bigger and much more complex calibration object. This helps making the calibration process robust against sensor noise (mainly introduced by the laser scanner).

3.1.4 *Experiments*

The objective of the first part of the experiments (i.e. Section 3.1.4.1) is to evaluate the quality of the IR calibration method against state-of-the-art calibration methods for visible-light cameras. Those experiments are conducted in daylight, as otherwise the visible-light cameras would fail to provide any useful data. The second part of the experiments (i.e. Section 7.2), provides a visual comparison of depth map estimation at conditions similar to driving-by-night scenarios.

Our multi-sensor platform consists of a (visible-light) Bumblebee XB3 camera (15 FPS at a resolution of 1280×960 pixel), a PathfindIR IR camera (8 Frames per Second (FPS) at a resolution of 360×288 pixel) and Velodyne HDL-64E S2 3-D laser range scanner (operating at 10Hz). The dimensions of our calibration board are $20'' \times 10''$. Image and laser scan acquisition are performed using three equivalent standard PCs, synchronized by a software implementation[2] of the IEEE 1584 Precision Time Protocol (PTP). Since calibration is done using static scenes, accurate time synchronization guarantees that the sensors observe the same pose of the calibration board. Obviously, this is an essential requirement for sensor to sensor calibration. For the Bumblebee calibration we rely on the automatic checkerboard corner detection and calibration algorithm implemented in the OpenCV library [3], but any other calibration tool could be used as well.

3.1.4.1 *Camera Calibration*

First, we evaluate the robustness of the IR resistor detection approach on a number of IR images captured under varying environmental conditions and different views. The first test set consists of 15 images

2 http://ptpd.sourceforge.net/

Environment	IR (our algorithm)		Visible Light Full (OpenCV)		Visible Light Small (OpenCV)	
	Detection	Error [%]	Detection	Error [%]	Detection	Error [%]
Outdoor	15/20	0.2677	20/20	0.0893	12/20	0.0822
Office	15/15	0.3007	23/23	0.2084	15/23	0.1534

Table 1: Detection rate and re-projection error of resistors in IR images with respect to different capture environments.

captured indoors (i.e. office) with an environmental temperature of $\approx 72°$F. The second test set consists of 20 images captured outdoors with an environmental temperature of $\approx 50°$F. Table 1 lists the fraction of all images where the resistors are successfully detected. The table further lists the *re-projection error* using the automatically calibrated intrinsic and extrinsic parameters. We compare the *re-projection error* to the automatic checkerboard corner detection and calibration algorithm implemented in the OpenCV library. The Bumblebee XB3 images are selected to show the same outdoor scene and calibration board pose we used to estimate the IR camera parameters. In order to obtain a fair comparison, we have to crop 40 pixels of the left and right part of each Bumblebee XB3 image (to obtain the same aspect ratio as the IR camera images) and scale the resulting image down to 360×288 pixel. For comparison, the checkerboard is detected two times in every picture: The first time, we detect the checkerboard in the full resolution image (i.e. 1280×960 pixel) and downscale the points afterwards. The second time, we first downscale the images and then detect the checkerboard in the low resolution image. The detection in the full resolution image represents our ground truth and the detection in the downscaled version of the images represents the scenario which is comparable to the resistor detection in the IR images.

With the OpenCV checkerboard detector, the calibration points are found in every image of the *Outdoor* and the *Office* scene using the full resolution Bumblebee XB3 images (see Table 1). Detection in the downscaled images, however, is only possible in $\approx$ 60 percent of all cases. The proposed IR resistor detection achieves considerably better detection rates, ranging between 75 and 100 percent. We further observe that the re-projection error of the IR calibration approach is slightly higher than the re-projection error of the OpenCV calibration result on the Bumblebee XB3 images ($\approx$ 0.3 pixel), but well below one pixel. The re-projection error obtained in the office environment is higher for all three tests because the camera distortion has more impact on objects closer to the camera (and thus close to the image borders). These results clearly demonstrate that the detection of the resistors and the calibration of the IR camera provides comparable

results to the calibration of a visible-light camera using a checkerboard pattern.

Part III

TRACK DETECTION

4 OBSTACLES

Objects are detected by dedicated sensor systems like a Velodyne HDL-64E S2 or a Stereovision system, but objects can be anything (trees, buildings, cars, ...). Obstacles are objects that interfere with the path of the moving train. If the train was moving on a track that is a simple straight line, a rectangular region of interest in *train coordinates* would suffice to differentiate between objects and obstacles. However a real track is obviously not just a straight line.

We are going to define a region along the track in front of the vehicle that must not contain any object. All objects detected within this region are automatically classified as obstacles and cause the obstacle detection to react accordingly. This region will be called *track clearance.* An exception to this are well known objects recorded in a *track atlas*. The *track atlas* is a list of points along the track that contain objects which are known to be *non-obstacles* like for example masts, guard railings and traffic signs.

4.1 LOADING GAUGE

The *loading gauge* defines the maximal envelope of a railroad vehicle to safely travel along the railway. This envelope in combination with the minimal curve radius can be used to calculate how close structures may come to the track to still allow all railroad vehicles to pass by. These envelopes are used during track planning and construction to allow all vehicles that conform to that standard to be operated on the tracks. For example, if a track is constructed according to the International Union of Railways (UIC) - C (Figure 23) envelope, all vehicles that are UIC - C conform can drive on this track, but also vehicles that conform to the UIC - Standard (Pass-Everywhere) gauge. The UIC - Standard gauge is the smallest common denominator of the UIC gauges.

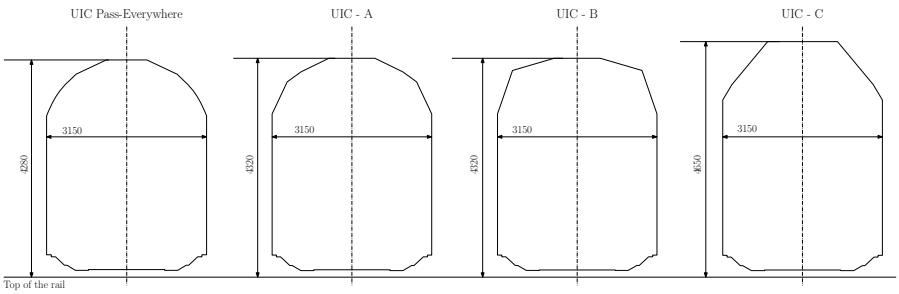


Figure 23: UIC (International Union of Railways) Loading Gauges

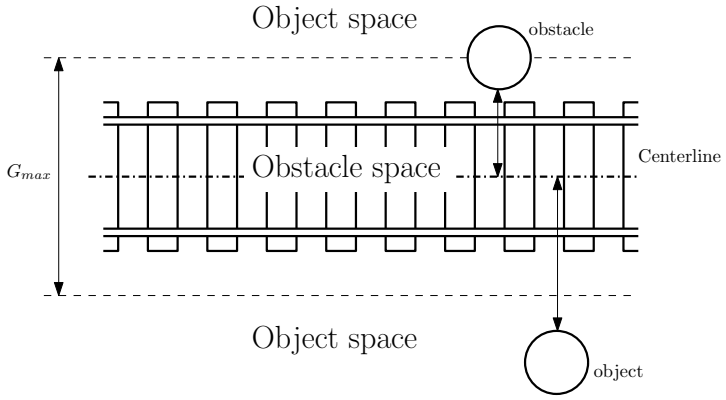


Figure 24: Distinction between obstacle and object space

4.2 TRACK CLEARANCE

We will use this *loading gauge* to define two new terms – the *obstacle space* and the *object space*. The object space is the area of our scenery that does not interfere with our train. For example, a house should be an object that has enough distance to the track to not interfere with the train passing by. Such an object belongs to the *object space*. However, an object positioned at the center of the track is clearly an obstacle and thus belongs to the *obstacle space*. For the train to sucessfully pass the whole track, the distinction between *obstacle* and *object space* has to be made very carefully. If the threshold is set too wide, too many objects will be classified as obstacles, resulting in a high *false positive* rate which will prevent the train from traveling along the path. Setting the threshold too narrow is, however, much more dangerous, because obstacles which will collide with the train may not be classified as such, resulting in a high *false negative* rate, which has to be prevented under all circumstances. A *false positive* means that the train may stop even if there is no reason to. A *false negative*, however, means that an obstacle will collide with the train, even though the obstacle has been seen by some of the sensors, but was classified as a *non-obstacle* i.e. *object*.

We know the *loading gauge* that was used to design the track. Thus, we can use this information to define our threshold between *obstacle* and *object space*. In reality, the loading gauge is a two-dimensional curve which would create some kind of tunnel when it is extended along the track. This means that the *obstacle space* and the *object space* are not simply regions on a map but volumes in 3D space. For the sake of simplicity we will project all objects into the *birdseye-view* (top-down view) and thus only use the most extreme points of the *loading gauge* for the width of the train.

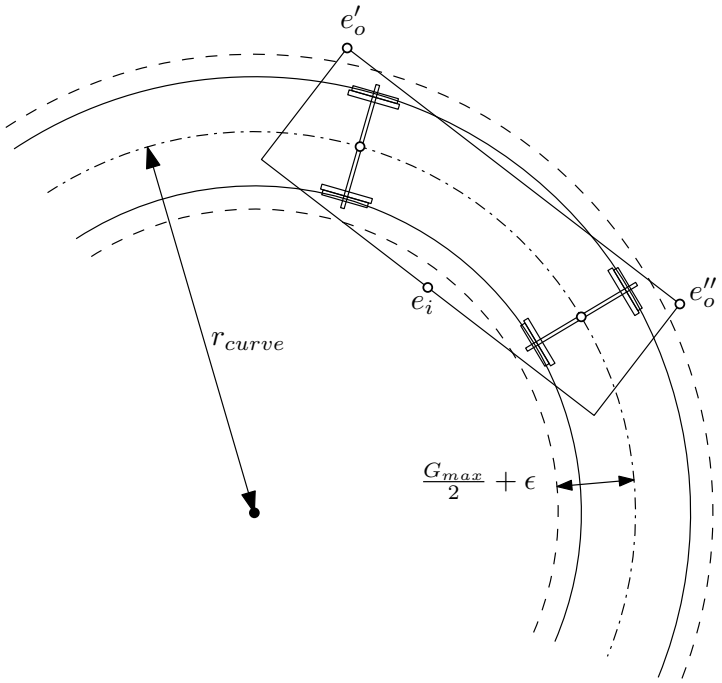


Figure 25: Extreme points of the train outline in curves

4.2.1 *Naive approach: static outline*

Once we know the *loading gauge* the threshold is set to the maximum width of the loading gauge plus an ϵ to make the *obstacle space* slightly larger than the actual outline of the (maximum possible) train. This threshold is shown in Figure 24 as G_{max}. An object where any point on its outline is farther away from the track centerline than $\frac{G_{max}}{2} + \epsilon$ belongs to the *object space (Figure 24)* and is classified as *not an obstacle*, which is a dangerous assumption because the *loading gauge* describes only the most extreme outline of the vehicle on a straight track. It does not take into account that the vehicle could extend even further outside the track while passing through a curve (Figure 25). Depending on the curve radius r_{curve} and the dimensions of the train, the most extreme points of the outline e'_o, e''_o, e_i may violate the threshold severely. Therefore, measuring the normal distance from an object to the track center and comparing this distance to the outline of the vehicle leads to a potential mis-classification of *obstacles* as *non-obstacles*.

4.2.2 *Dynamic outline*

The most extrene points of the train outline e_i, e'_o, e''_o (Figure 26) depend on the curve radius r_{curve} , the distance between the two axles d_{axle}, the width of the train, or in our case the width of the *loading gauge* G_{max}, and the length of the train L as shown in Figure 27a. We need

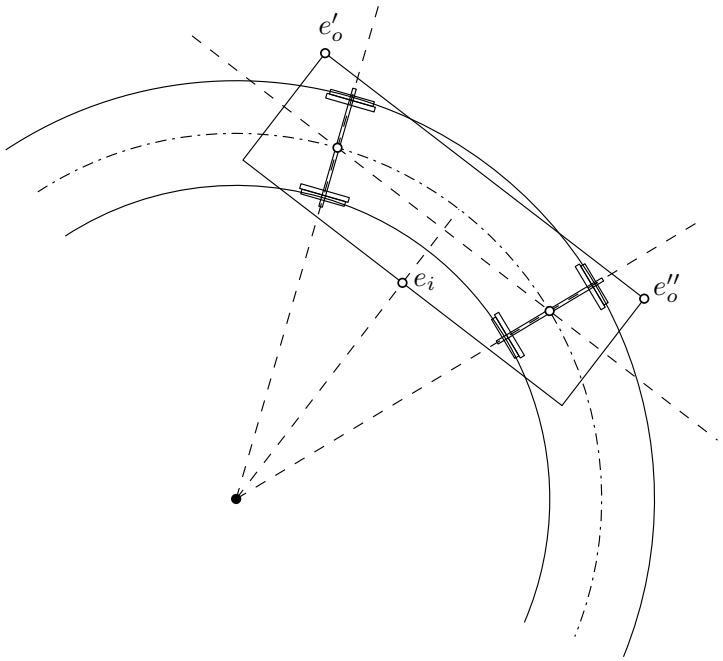


Figure 26: Calculating the extreme points to estimate the dynamic outline

to calculate the radii $r_{e_i}, r_{e'_o}, r_{e''_o}$ (Figure 27b) and compare them to the radius r_{curve} to determine the dynamic threshold between *obstacle space* and *object space*.

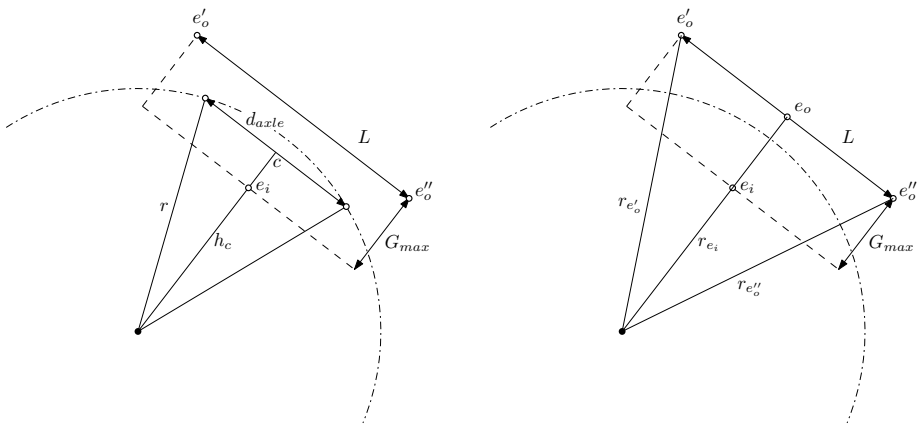
As seen in Figure 27a, the pivot points of the two axles form a triangle with the center of the curve whose height h_c (derived by Equation 4.1) corresponds to the radius of the curve described by the center of the train. This radius (h_c) is related to r_{e_i} through the width of the train (*loading gauge*) shown in Equation 4.2. The point e_o is the extension of the line from the center of the curve through e_i to the opposite side of the train. The radius to this point is $r_i + G_{max}$ which is the radius to the inner point plus the width of the train. This point (e_o) forms a right-angled triangle with the center of the curve and the point e'_o (or e''_o, respectively). Using Equation 4.3 we can compute the radius $r_{e'_o}$ (or $r_{e''_o}$, respectively).

$$h_c = \sqrt{r^2 - \left(\frac{d_{axle}}{2}\right)^2} \tag{4.1}$$

$$r_{e_i} = h_c - \frac{G_{max}}{2} \tag{4.2}$$

$$r_{e'_o} = \sqrt{(h_c + \frac{G_{max}}{2})^2 + \left(\frac{L}{2}\right)^2} \tag{4.3}$$

If the train is not symmetrical *w.r.t.* the axis defined by $\overline{e_i e_o}$, the formulas for $r_{e'_o}$ and $r_{e''_o}$ will differ slightly from each other, as the two



(a) Radius to e_i (b) Radius to e'_o and e''_o

Figure 27: Calculation of the dynamic gauge

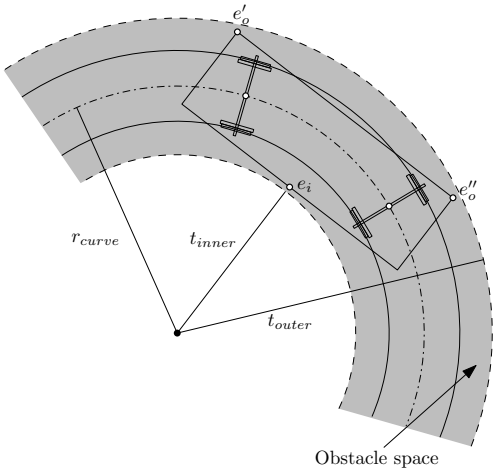


Figure 28: Dynamic thresholds based on the dynamic gauge separate the *obstacle space* from the *object space*

edges of the right-angled triangle wont be of equal length ($\frac{L}{2}+\Delta$, $\frac{L}{2}-\Delta$ versus $\frac{L}{2}$, $\frac{L}{2}$). We assume however that the train is symmetrical, so that $r_{e''_o}$ can be ignored as it is the same as $r_{e'_o}$.

After calculating r_{e_i} and r_{e_o} the thresholds for the *obstacle space* are derived as $t_{inner} = (r_{curve} - r_{e_i}) + \epsilon$ *and* $t_{outer} = (r_{e'_o} - r_{curve}) + \epsilon$ which define the separation between the *obstacle space* and the *object space* shown in Figure 28.

5 TRACK DETECTOR

Lane detection for driver assistance systems is a topic that gained a lot of attention during the last ten years ([7, 28, 31, 32, 57, 27, 60]). Some approaches work on the acquired images directly, which represent a perspective projection of the scene ([7, 57, 6]), and some perform a conversion of the scene into a top down view called Inverse Perspective Mapping (IPM) ([28, 32, 51, 27]). Most systems use a simple model to describe the lane, which also applies to railroads. However, only few systems are designed specifically for railroad detection ([33, 53]). An obstacle detection system for trains is proposed in [53]. In [33], a vision-based system for collision avoidance of rail track maintenance vehicles is proposed. It is based on detecting railroad tracks by applying techniques similar to lane detection for driver assistance systems. The spatial period of the sleepers and the distance between the rails is used to calibrate the camera parameters. A piecewise quadratic function is then fitted to candidate rail-pixels in the acquired (perspective) image and compared to the previous frame. However, to the best of our knowledge, no fully functional vision based obstacle detection system for railways exists to date. This work uses the well researched field of lane detection and tracking and extends it to the field of train applications.

5.1 COMPARISON STREET VS. RAILWAY

While the basic task of lane detection in street and railway scenarios is the same, there are several properties that require special attention and may help us to improve the robustness of automatic train systems. Table 2 lists some important differences, which are discussed subsequently.

First of all, the lane width on railways is obviously fixed along the whole path. Otherwise, the train would not be able to drive the complete track. The width of lane markings on streets, however, depends primarily on the type of road and is generally limited by a minimum width required by law, which in turn depends on the country the street is located in. The lateral offset of a car relative to the center of the lane is variable, which is especially true for wider lanes. We will later see that a fixed lateral offset can be exploited to reduce the possible track candidates.

Road lane markings are designed in a way that they are optimally visible to a human observer (Figure 29b). Once they wear off too much, they are repainted. In contrast, the only purpose of railroad

Street	**Railway**
variable lane width	fixed lane width
variable lateral offset	zero lateral offset
varying type of lane markings	fixed "lane markings"
general lane appearance is relatively homogeneous	several different (inhomogeneous) "lanes"
lane markings are designed for optimal visibility	visibility is not guaranteed
lane markings have no volume and thus don't cast shadows on the ground	tracks have a certain height and thus cast shadow on the ground
construction sites and obstacles can easily change the path of a car within a road	vehicle path through the world coordinate system is fixed
the speed of a car is generally adapted to weather and visibility conditions	automatic trains are operated at nearly constant speed independent of most weather conditions
the horizontal movement of a car-mounted camera is low due to the low height of the vehicle	swinging of the train causes substantial change of the camera position along the path

Table 2: Properties of street lanes and *railway tracks*

(a) Inhomogeneous lanes between railroad lines (average case scenario)

(b) Good visibility of lane markers and very homogeneous roads

(c) Summer with vegetation

(d) Bad lighting because of shadow

(e) Summer with no vegetation

(f) Good winter conditions

Figure 29: Comparison between road lanes and *railway tracks*, and overview of different scenarios

tracks is to guide the train. It is just a side effect, if the tracks are easily visible to an observer. Although in many situations the tracks are very prominent, in general, visibility is affected by changes in lighting and weather in a much stronger way than road lane markings. An advantage of rails over lane markings is that they are constantly grinded each time a train rolls over them. This keeps the top of the rails from corroding. The appearance of the street itself is also very homogeneous (Figure 29b), compared to the track bed of railways (Figure 29a). The track bed or, in general the space between the rails, consists for example of gravel, asphalt, snow or even grass (Figure 29a). The last significant difference in the visual appearance is the volume of the tracks. Lane markings are flat and basically have no volume or height. This means that they can not cast shadows on the ground and thus the detection of the lane marking itself is invariant to the position of the sun, if no other object casts a shadow on the street. Tracks, however, have a certain height, i.e. several centimeters, and thus cast shadows, which create additional edges in the image and *weaken* the visual appearance of the real edge.

If we combine the fact that the lateral offset is fixed and that the position of the track can not be changed easily, it is clear that the train always moves on a fixed path with respect to the *world coordinate system*. This allows a much tighter integration of a-priori information like the prediction of the vehicle position at upcoming points in time. However this is only true for the whole vehicle, since trains have a strong trend to swing left/right especially at higher speeds. This is due to the great mass in combination with the suspension that is designed to make the ride comfortable for the passengers. This strong swinging, combined with the fact that the train is considerably higher than a regular car, results in a displacement of the camera system that can not be predicted easily. This means that, even if two frames are acquired at the exact same position on the track at two different points in time, the position and orientation of the camera with respect to the world coordinate system is not the same.

A final, but very significant property are the weather conditions. Trains are operated at constant speed over a much greater range of weather conditions than a normal car. For example, even if the track is nearly fully covered with snow, the trains are still operated with no or only a slight reduction in speed, because they need to keep their timetable.

An overview of common scenarios is provided in Figure 29. We can see that the amount of vegetation that is allowed on the track has a large impact (Figure 29c) on the appearance of the space between the tracks. Figure 29e shows one of the best case scenarios for track detection where the top of the rail reflects light quite well. However, one can not rely on this feature. Under bad lighting conditions the

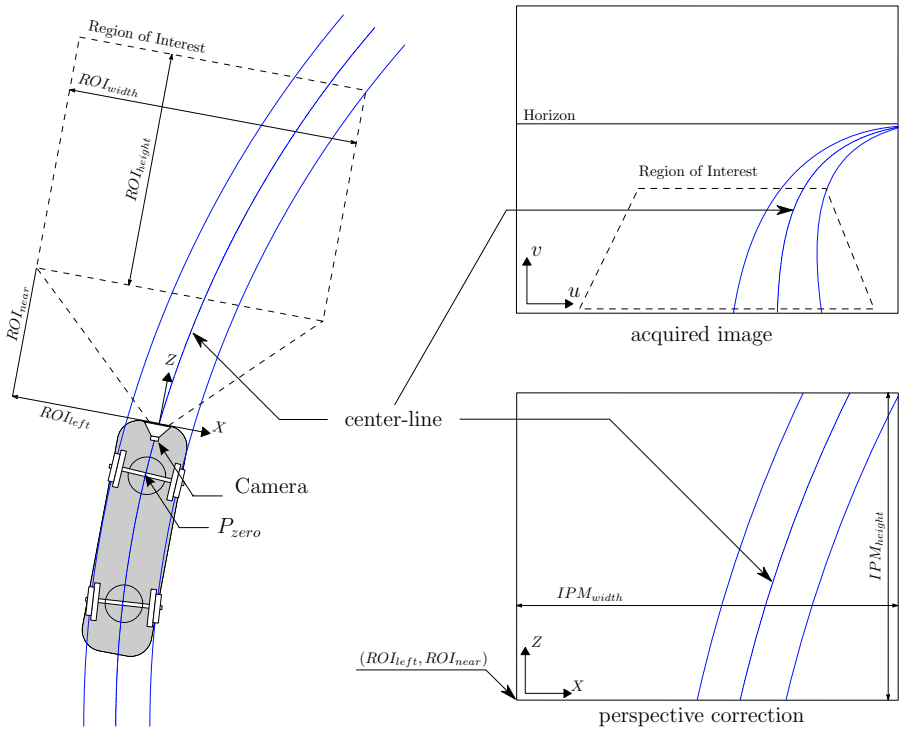


Figure 30: Acquisition of the IPM image

brightness difference between the tracks and the surrounding area gets problematically low (Figure 29d).

5.2 INVERSE PERSPECTIVE MAPPING

Based on the observations of the railway track properties, we are able to design algorithms that are optimized for those scenarios. By transformation of the perspective view into a birds-eye orthogonal view, also called IPM ([27]), we gain the ability to directly check all the geometric constraints that our algorithm requires. In addition, the transformation also makes it easier for appearance based algorithms to find matching regions, because the perspective projection does no longer deform the objects depending on their location and thus the track width remains constant over the whole image.

Figure 30 shows the IPM step. The camera acquires the scene in front of the train, which is transformed through a perspective projection (Figure 30) (*acquired image*). While it is possible to find parallel lines in perspective images [57], it is much simpler, if one has access to the undistorted view. As our algorithm heavily relies on the fact that the tracks are parallel with constant distance at all times, it makes sense to perform an IPM prior to the track detection. We also mentioned that the train undergoes a swinging which translates and rotates the camera in the world coordinate system. To be able to correctly calculate an Inverse Perspective Mapping, we use the camera parameters to

calculate the perspective projection of every point in the *birds eye view*. This is slightly more complicated than warping the input image, but provides higher flexibility in dealing with the moving camera and non-planar surfaces (which are assumed by image warping techniques).

To calculate the IPM, a region of interest in *world coordinates* is defined. For example, 6 meters left, 6 meters right and from 5 to 45 meters in front of the camera. Currently, we also require the world in front of the camera to be a planar surface and thus assume a z-Coordinate of zero. This could be changed in the future by integrating the surface curvature from LIDAR data.

The *extrinsic* and i*ntrinsic camera parameters* are calibrated offline since they are not going to change once the system is in place.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ROI_{left} + \frac{ROI_{width} * x_{ipm}}{IPM_{width}} \\ ROI_{near} + \frac{ROI_{length} * y_{ipm}}{IPM_{height}} \\ 0 \end{pmatrix} \tag{5.1}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = T_{ext} + R_{ext} * \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{5.2}$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{x' * f_x}{z'} + c_x \\ \frac{y' * f_y}{z'} + c_y \end{pmatrix} \tag{5.3}$$

We need to scale and offset the points in the IPM image to fit in the desired Region of Interest (ROI) (as seen in equation 5.1). ROI_{left} denotes the outermost left point of our (physical) region of interst and ROI_{near} defines the closes point of our region of interest. Combined with the width (ROI_{width}) and height (ROI_{height}), the region of interest is completely defined in physical space (see Figure 30), because, for now, we assume the ground to be a flat surface. In equation 5.2, the extrinsic camera parameters R_{ext} (rotation of the camera) and T_{ext} (translation of the camera) are used to transform the world coordinates of our ROI into the camera coordinate system. These points are then projected onto the image plane by using Equation 5.3 with the *intrinsic camera parameters* f_x, f_y (focal length) and c_x, c_y (center of projection). This finally establishes a relation between the points in the IPM image (x_{ipm}, y_{ipm}) and the points in the acquired image (u, v). This is done for every pixel in the IPM image and thus reverses the perspective projection of the camera as seen in Figure 30 *(perspective correction)*.

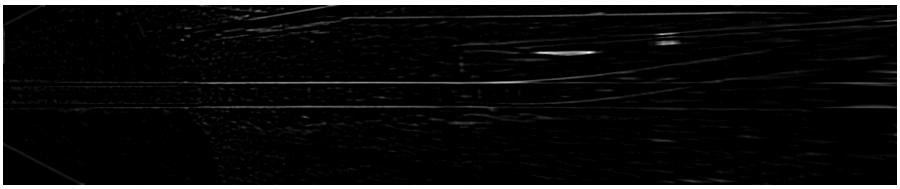
5.3 PRE-PROCESSING

The first step of the processing chain is some preprocessing after which only candidate pixels for edges should remain. Starting with the input image (Figure 31a) from the camera, the detector creates a *birds eye view* by performing the Inverse Perspective Mapping (IPM) from Section 5.2 on the image. The resulting IPM image is shown

(a) Example image for track detection

(b) Inverse perspective mapping (birds-eye view)

(c) Inverted image depending on the scene (snow/no snow)



(d) Difference of Gaussian edge image

Figure 31: Preprocessing steps of the track detector (images are rotated 90° clockwise)

in Figure 31b. If the camera parameters are calibrated correctly, the mapping of the tracks should be perfectly parallel to the y axis of the image.

Depending on the scenario, the grayscale values of the image may be inverted. For example, if the environment is white due to snow, the the image should be inverted to make the track bright and the background dark. This is a setting that does not change during the operation and can be adjusted at startup.

The edges in the image are enhanced using a Difference of Gaussian (DoG) filter which calculates the difference of two images with a different sigma of the Gaussian kernel. For our camera setup the best results are achieved by choosing $\sigma_1 = 7$ and $\sigma_2 = 3$.

5.4 DYNAMIC MASK

Consecutive views of the track are heavily correlated. This is not surprising because the train can not jump from one position to another. This also means that the position of the track in two consecutive images is very likely to be close. We exploit this fact by masking unlikely regions in the IPM image based on the previous track.

Using the previously detected track, C_{t-1} in Figure 32, the track mask is calculated as the space between two parallel curves, F_L and F_R, of C_{t-1}. The distance between $F_L$and C_{t-1} (as well as the distance between F_R and C_{t-1}) is half the mask width $\frac{w_{mask}}{2}$. The mask width is defined as

$$w_{mask} = w_{track} + \frac{\lambda}{1 - \Delta t} \tag{5.4}$$

where w_{track} is the track width (in our case 1 meter), λ is a growth factor that specifies how big the mask should be extended after one second and Δt is the age of the previousy detected track C_{t-1}. If the previously detected track is very recent, the track mask is hardly extended, as the train can only move at a certain speed. After a short period of time, the train will still be very close to the previous position, resulting in a track that is almost unchanged in the current view. If a track is incorrectly detected, the track mask will remove parts of the rails, but only temporarily. When the track can no longer be detected, the previously detected track grows older and the track mask will grow as well, at some point again including the real rails. Tracks older than *one* second are no longer considered. If the previous track is older than *one* second, there was probably a mis-detection and the track mask is increased to cover the whole image.

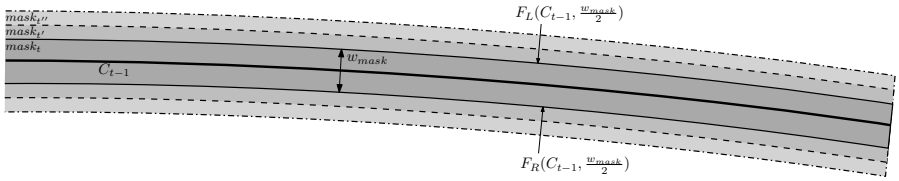


Figure 32: Track mask example at three different ages

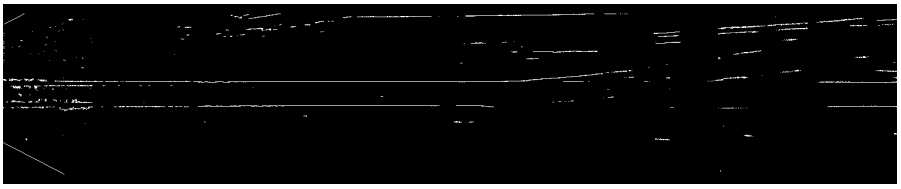


Figure 33: Extraction of valid edge pixels by detecting local maxima with a sliding window approach

5.5 LOCAL MAXIMA SEARCH

After the preprocessing step in Section 5.3, many incorrectly enhanced edges remain (Figure 31d). Even if the lighting is bad we have to assume that the rails are still a more prominent feature than the immediate background. We are using a sliding window approach to detect local maxima in the image.

The image is split into ten equally sized slices along the y axis. For every slice the threshold $(t_0, t_1, ..., t_9)$ for the *local maxima detection* is calculated individually (Figure 34). This is because the whole image spans a big region in the scenery which causes the IPM image to get very blurry for points that are far away from the camera, which is obvious, since lines at the top of the image are extrapolated from very few image points compared to lines at the bottom of the image, which are very close to the camera.

We assume that the horizontal gradient at rail-edges is relatively high compared to the the immediate surroundings. However, we can not assume that it is the strongest gradient in the whole line. There might be light reflections or other structures that have a very strong gradient. Instead of taking the maximum gradient, we calculate thresholds as a quantile of the horizontal gradients of ten equidistant lines within one slice (Figure 34). The quantile is configurable, but our tests have shown that the 99% quantile provides overall acceptable results.

The local maxima are detected for each line by sliding a window of 31 pixels over the line (Algorithm A.1). The algorithms searches for a value that is lower than the current value minus t_{slice} from the current point 15 pixels to the left. If such a value is found the algorithm goes on and does the same with the 15 pixels to the right. If both sides contain such a value and all values are below the center pixel, the center point is marked as a local maximum. This local maxima image

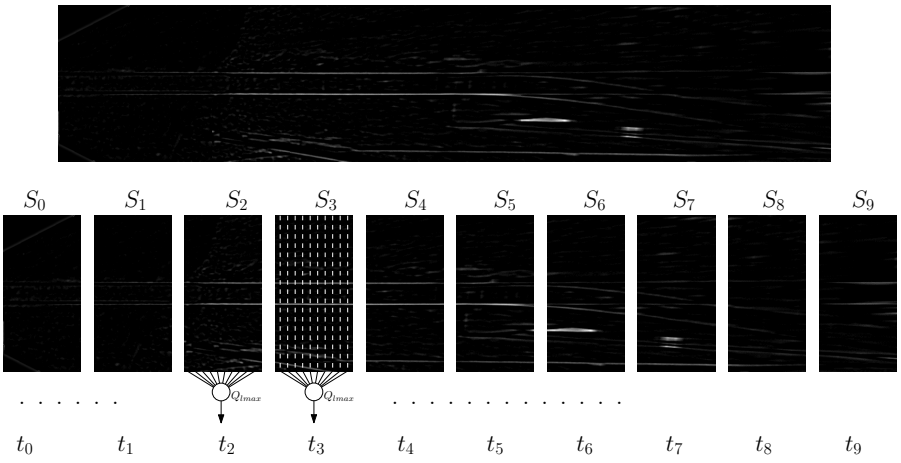


Figure 34: Slicing of the preprocessed image and calculation of the slice thresholds

will have many gaps in the rails especially in the parts close to the train. This image is then enhanced with morphological operations. The supported operations are

- Erosion - v / dilation - V, with a 3×3 vertical structuring element

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{5.5}$$

- Erosion - h / dilation - H, with a 3×3 vertical structuring element

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.6}$$

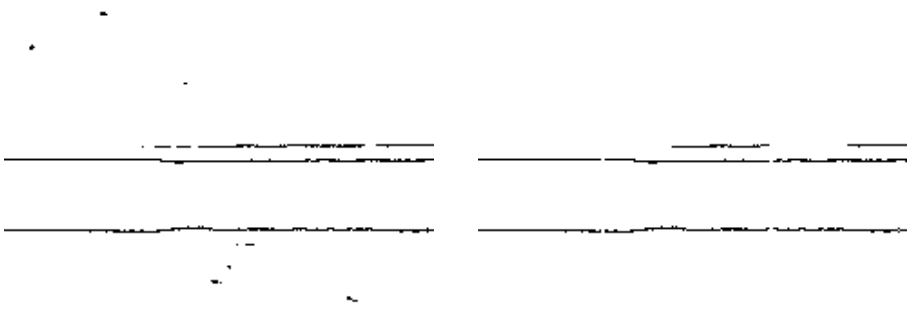
- Erosion - s / dilation - S with a 3×3 rectangular structuring element

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5.7}$$

Our algorithm parses a filter command and applies the operations in the corresponding order. The most successful filter on our data is VV , which are two vertical dilations.

5.5.1 *Block based thresholds*

The slice-based threshold selection works well, if the rails are very prominent inside the whole slice. However, under bad lighting conditions, or when the rails are inside a shadow while the rest of the slice



(a) (b)

Figure 35: a.) Results of the blob extraction following a b.) removal of too small contours

is not, this approach will fail. To cope with such scenarios, we have to change the assumption that the rail edges are among the strongest horizontal gradients within a whole slice.

It is still assumed that the rail edges have a very strong horizontal gradient, but the area within which the thresholds are calculated is reduced. The vertical slicing is a mechanism to cope with changes of the rail appearance along the track. Similarly, splitting a slice further along the horizontal axis will result in a more localized threshold selection. However, instead of splitting the slices, we are splitting the whole image in equally sized blocks of 50×50 and calculate the threshold for each block as the 99% quantile of the horizontal gradients. The rest of the *local maxima detection* remains the same as in the slice-based version.

5.6 BLOB DETECTION (CONNECTED COMPONENTS)

The output of the local maxima search is not necessarily a one pixel wide line and almost never a straight line segment. The general idea is to extract the contours of the connected components in the local maxima image. This is achieved by using the *findContours* function of the OpenCV library, which is based on the border following algorithm described in [50]. The result of the blob extraction is a list of contours of connected components in the image. A contour is described as a list of (u, v) image points. When the contours of lines without holes are extracted, the result (Figure 35a) looks just like the result from the local maxima search. However, contours with a length below a certain threshold are removed (Figure 35b).

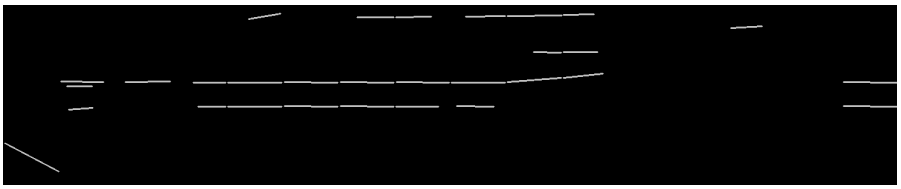


Figure 36: Fitted line segments to the connected components

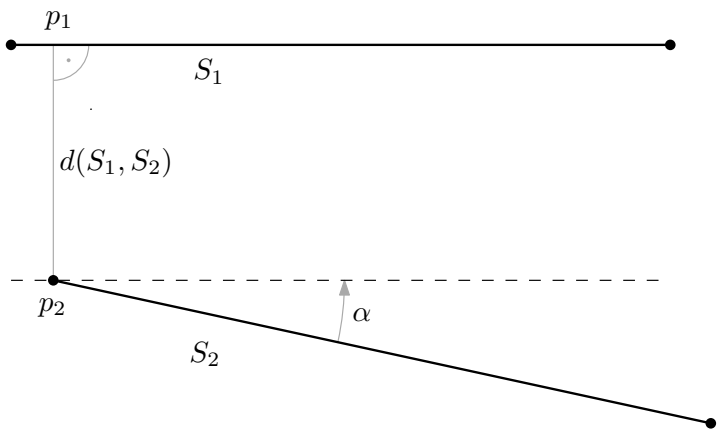


Figure 37: Distance and angle between two segments

5.7 LINE-SEGMENT FITTING

The contours of candidate lines are not yet useful to further processing, because they consist of too many points and may not describe a line segment at all (e.g., if a circular blob was extracted). Each line segment in a slice corresponds to a real world distance of approximately 4-6 meters. Our tests have shown that, at this scale, we can safely approximate the extracted contours with a straight line (Figure 36), because the curvature of the track is not big enough to produce an error that would impact further processing. Lines that are shorter than a certain threshold are removed to eliminate spurious detections.

5.8 LINE-SEGMENT FILTERING

All previous steps belong to generic pre-processing, which does not take into account the intrinsic track constraints like the parallelism of the rails and the exact track distance. These properties can be used to reduce the track candidates even further. Because of the transformation from the *perspective view* into a *birds eye view* (Inverse Perspective Mapping), the measurement of these properties is quite simple.

We first calculate the *infimum distance* (Equation 5.8) between every pair of line segments in the slice.

$$d(S_1, S_2) = \inf \{ \|p_1 - p_2\|_2 : p_1 \in S_1, p_2 \in S_2 \} \tag{5.8}$$

We then calculate the angle between all pairs of segments (S_i, S_j) with a distance that does not deviate from the target distance w by more than a certain threshold t_d . If the angle is below the threshold t_α, the track segment is a valid candidate.

$$g(w, t_d, t_\alpha) = \{(S_i, S_j) : |d(S_i, S_j) - w| < t_d, |\angle(S_i, S_j)| < t_\alpha, \forall S_i, S_j \in S\} \tag{5.9}$$

The set $g(w, t_d, t_\alpha)$ of all valid segments in a slice is give by Equation 5.9. An example for the angle $\angle(S_1, S_2)$ between the two segments S_1 and S_2 is shown in Figure 37 as α.

The rails of the track are almost perfectly parallel since they have a constant distance over the whole track with only a slight variation ($< 5cm$). Accordingly non-parallel segments can only be encountered due to measurement errors (*i.e.*, slightly wrong camera parameters, errors in the pre-processing). Therefore, the threshold for the angle- and distance variation should be selected very conservatively. Our tests have shown that good values for t_α are 11 *degrees* and *ten* pixels for t_d.

The output of this step are not only single segments that conform to the filtering constraints, but rather pairs of segments that matched the filtering constraints. These pairs are theoretically already corresponding parts of the left and the right rail.

5.9 CURVE FITTING

To derive a track estimate from the candidate segments, we need to apply some kind of track model and find the model parameters that are best supported by the detected candidates.

There are various possible models for street lanes. The most common algorithms use piecewise linear models ([25, 4]), b-splines ([56]) or polynomial models ([31, 29]). Until recently, the railway track design was dominated by clothoids ([36]) because they linearly increase the curvature with the path length (Figure 38). This results in a smooth change of the centripetal force.

In theory, using the clothoid curves as the underlying model and trying to find model parameters that represent the detected line segments would be the most accurate procedure. There are several techniques ([35, 2]) to fit piecewise clothoid curves to lines. They are, however, time-consuming and in our case not necessary. We only have very short curve segments within which the curvature slightly changes. We also need a simple parameterization with very few parameters. Piecewise spline approximations would result in a near perfect representation of the track, but at substantial processing costs and the fact that two curves which are very similar may have substantially differing approximations.

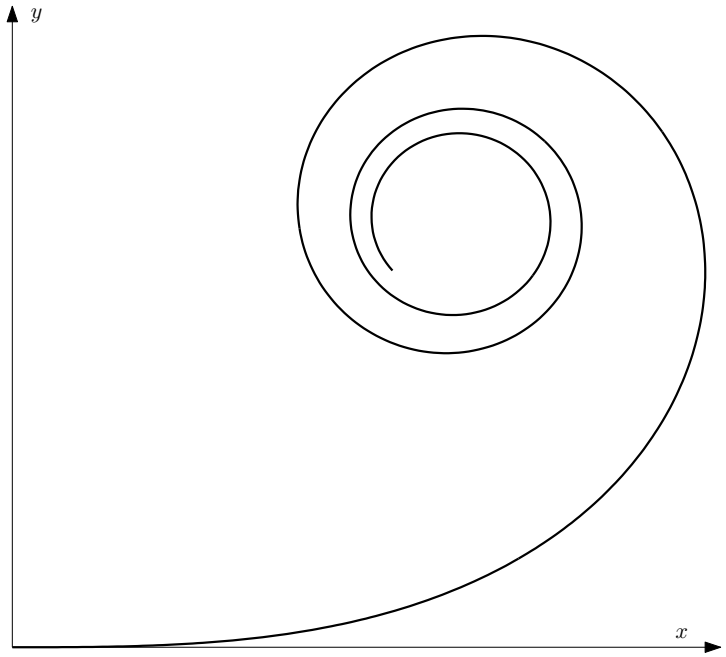


Figure 38: Part of a clothoid curve of degree 2

5.9.1 *Polynomial fitting*

One possible curve approximation are 2^{nd} order polynomial curves

$$f(x) = a_2 \cdot x^2 + a_1 \cdot x + a_0 \tag{5.10}$$

The parameters of such a curve can be estimated with least squares fitting by solving the following system

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{5.11}$$

The solution $\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$ is calculated by pre-multiplying with the the inverted matrix $\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}^{-1}$

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{5.12}$$

But since the input data is subject to noise and errors, the matrix has to be inverted by calculating a pseudo-inverse, which then yields the solution for $\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$ as

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = pinv \left(\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \right) \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{5.13}$$

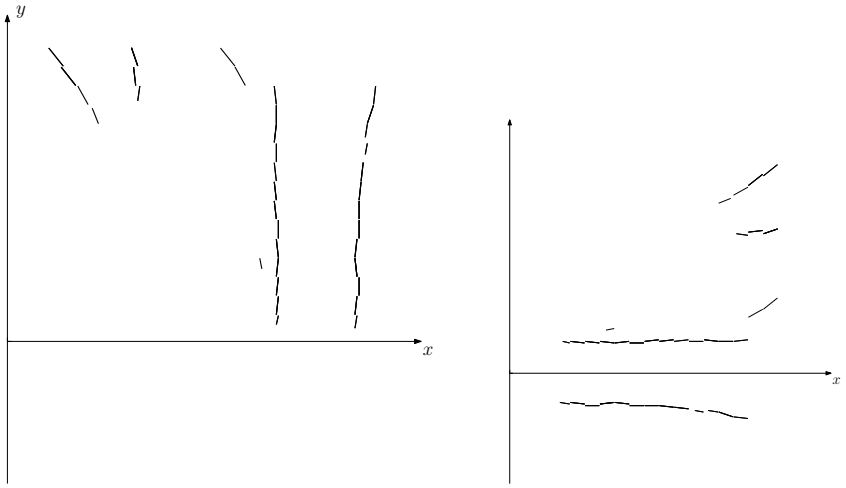
If we use higher order polynomials, the fitting becomes subject to noise and yields generally worse results. They are a better fit for the data, but not the underlying track.

5.9.2 *Rotated polynomial curve*

We derive the best fitting curves after the *line-segment filtering* of Section 5.8 using the technique described in Section 5.9.1. This means that our input data is a set of point pairs describing the line segments. These points will be used as the input for the *polynomial fitting*. They are in image coordinates and, because the tracks are extending roughly along the z axis of the train, the rails extend along the y-coordinates in the image. Fitting a polynomial to these points would be very unstable, or not possible at all, if the points are not a function of x.

To fit a polynomial curve to a generic dataset, the data has to be transformed (rotation,translation) into a coordinate frame in which the points to be fitted are a function of x.

The first step is to determine the rotation of the points by *least-squares* fitting a line to them using the *fitLine* function from the OpenCV library (Figure 67). The rotation is thus simply $atan\left(\frac{dy}{dx}\right)$. The translation parameters are set to transform the leftmost point into the origin. The transformed points are then fitted with a 2^{nd} order polynomial as described in Section 5.9.1. The result from the fitting procedure is stored in combination with the rotation and translation parameters as a tuple $(a_0, a_1, a_2, \theta, u_o, v_o)$. If the curve needs to be evaluated, we need to specify a rectangular region of interest which is usually the whole image, i.e. $(0, 0, width, height)$. This rectangle is



(a) Unrotated

(b) Rotated and translated to enable curve fitting

Figure 39: Segments used for polynomial fitting

rotated and translated according to the parameters θ, u_0, v_0. To evaluate the polynomial only within the rotated rectangle, the algorithm calculates the intersection of the polynomial with all four lines of the rectangle.

Each line of the rectangle can intersect with the polynomial in at most two points, which results in a maximum of eight intersection points $\{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8\}$. The example in Figure 40 shows all eight intersection points. Since we are only interested in the part of the curve that is inside the rectangle, the lower bound of our evaluation interval can not be less than u_l and the upper bound can not exceed u_u. The initial upper and lower bounds (u_u, u_l) are calculated as

$$u_l = min\{u_{P_1}, u_{P_2}, u_{P_3}, u_{P_4}\} \tag{5.14}$$

$$u_u = max\{u_{P_1}, u_{P_2}, u_{P_3}, u_{P_4}\} \tag{5.15}$$

An intersection point is only valid if it lies on the corresponding segment of the rectangle. For example, in Figure 40 the point I_2 is valid because it lies on the segment $\overline{P_1P_3}$ whereas I_3 is not valid since it lies outside the segment $\overline{P_1P_2}$. If valid intersections are found, we can assume that at least two intersections are valid. The only case in which only one intersection would be found is if the parabola touches exactly one of the segments in only one point. In this case, the whole curve is outside the rectangle and does not need to be evaluated anyway.

In all other cases the limits of the evaluation interval (u_l, u_r) can be calculated as

$$u_l = min\{u_{I_j} : I \; is \; valid\} \tag{5.16}$$

$$u_u = max\{u_{I_j} : I_j \; is \; valid\} \tag{5.17}$$

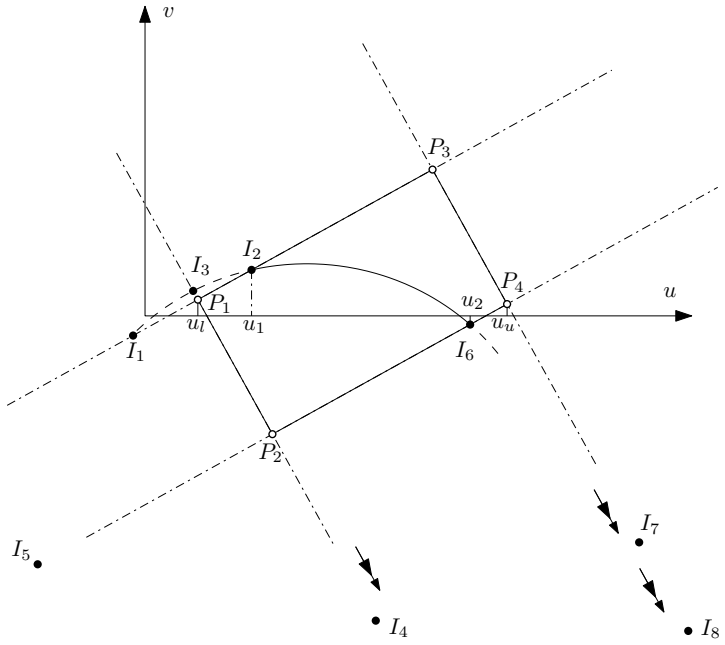


Figure 40: Calculating the evaluation interval

5.9.2.1 *Fixed rotation*

This algorithm provides a generic method to fit a polynomial to a set of points. However, our input data is heavily constrained along the y axis. As shown in Figure 39a, most segments after the line segment fitting in Section 5.7 are almost orthogonal to the x axis. Therefor a simplification of the rotated polynomial curve fitting is to rotate the points 90° clockwise, which can be further simplified by switching the x and y coordinates.

5.9.2.2 *Matching quality*

After the curve is fitted to the input points, it is approximated with line segments by sampling the curve inside the image rectangle $(0, 0, IPM_{width}, IPM_{height})$. The matching quality is the average of the squared minimum normal distances of all input points to the line segments of the approximated curve

$$m(C, T) = \frac{1}{\#T} \sum_{p \in \mathrm{T}} inf\{\|p - p_S\|_2 : p_S \in S, S \in \mathrm{C}\} \tag{5.18}$$

where C is the set of all line segments of the approximated curve, T is the set of all input points and $\#T$ is the number of input points.

5.9.3 *Track candidates*

Simply adding all points from all detected line segments into the fitting procedure does not work because the least-squares fitting will

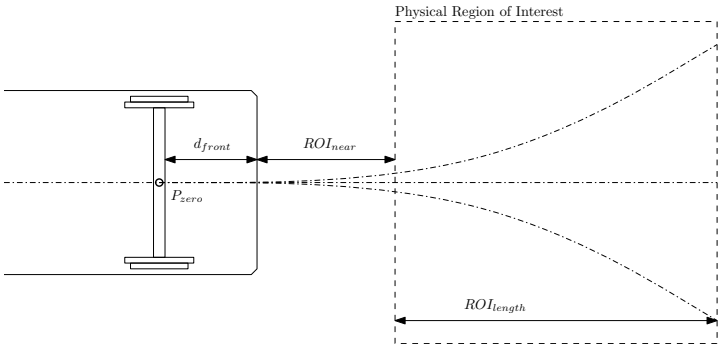


Figure 41: Track centerline must pass through the point P_{zero}

just fit a curve to the data that does not correspond to the real track. Unless there is a substantially higher number of good track segments compared to the number of false detections we need to find a way to determine the best track candidate from all segments. This requires additional information of the track properties to remove unwanted segments.

The two strongest properties are

- track width
  As mentioned in earlier sections, the track width is fixed to *one* meter. Thus, the left and right rail are half a track width away from the centerline. Any segment that violates this constraint after the track fitting is either a false detection or the fitted track is invalid

- the centerline *must* pass through the point P_{zero}
  Figure 41 shows three different track curves that all pass through the point P_{zero} which is the pivot point of the frontal axle. This is obvious since the width of the axle can not change during operation and the position of the pivot point can not change either

Using these properties, we can define a metric that outputs how good the track segments fit our model. The first step is to convert the point P_{zero} in Figure 41 to image coordinates by scaling $d_{zero} + ROI_{near}$ according to the relation between ROI_{length} and the height of the IPM image (IPM_{height}). The resulting image coordinate of P_{zero} is

$$I_{zero} = \left(0, \frac{d_{zero} + ROI_{near}}{ROI_{length}} \cdot IPM_{height}\right) \tag{5.19}$$

which assumes that the physical region of interest is symmetrical along the x axis. This is true because we do not change the physical ROI during operation and must cover both sides of the track equally.

The output of the *line-segment filtering* from Section 5.8 are pairs of segments. Since we are not interested in the individual left and right rails, the start and end points of the left and right rails are averaged

to get a centerline from each pair. Now, all centerline segments from the first slice are combined individually with the point I_{zero} and a track curve is fitted to every one of those combinations. Calculating the matching score with Equation 5.18 yields the parameter that corresponds to the inverse of the *"track"-ness*. If it is zero, we have a perfect match and assume that this is a valid track, higher values are worse. The user can define a threshold t_c at which the curve candidate is no longer a valid track. All candidates below the threshold are kept and are combined with the candidates of the next slice. In addition, the algorithm assumes that track segments from previous slices may not have been visible or undetected, so every segment of the slice is added as a new track candidate.

5.9.3.1 *Selection of the best candidate*

An overview of the algorithm is shown in Algorithm A.2. All of the candidates conform to the defined threshold. However, the algorithm assumes that the best track candidate will contain the most track segments. This is a reasonable assumption since only one track segment per slice can be chosen for a given track candidate and thus a candidate with many segments is "supported" by segments from several slices. If two candidates have the same number of supporting segments, the one with the lowest matching score is selected. The reason why the matching score alone can not be used is that a track candidate with very few segments may have a better fitting polynomial, but does not necessarily depict a valid track.

5.10 HOUGH MAP

Polynomial fitting provides an accurate estimation of the track in front of the train for $\approx 75\%$ of the tested frames. However, the polynomials do not necessarily represent possible tracks. This means that a track candidate which best fits the data might not be a possible track at all. In reality, the number of possible tracks is very limited. There is an upper bound on the curvature of the track and the rails in front of the train emerge all from exactly the same point P_{zero} . This was already mentioned in Section 5.9.1. Such constraints can be used to search the scene for rails that are from a predefined set of model parameters. However, simply searching a predefined set of parameters for matching points in the image may be very time consuming, especially if the number of parameters is high. Furthermore, most of the points in the image will be checked several times, whether they contain an edge pixel or not.

A well known solution to this problem is the *hough transformation*, which builds a parameter space for the given model (line, circle). For every pixel in the image, the values in the parameter space for each model the pixel may belong to are increased. The *hough trans-*

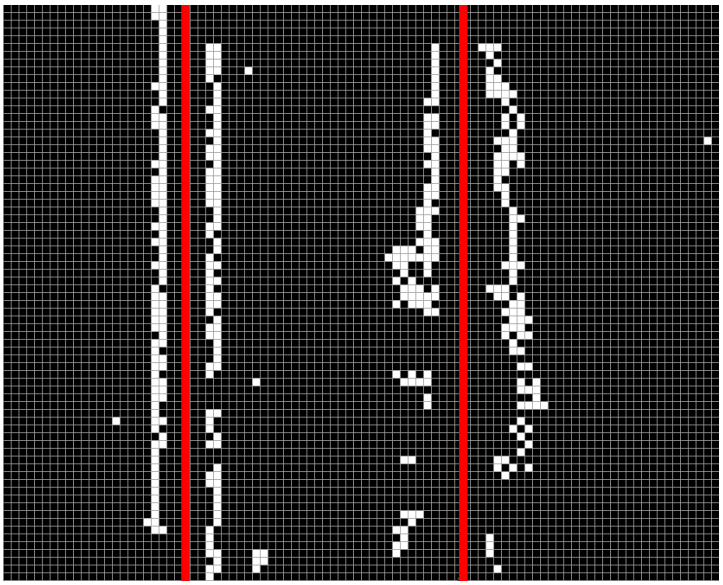


Figure 42: Local maxima in wrong places

form can, in theory, be expanded to more complex models, but the time and space requirements are growing exponentially with the number of dimensions. Since our model already has six parameters $(a_0, a_1, a_2, \theta, u_0, v_0)$, the hough transform would outright use up all the available memory, and more. We can reduce the number of dimensions to three, if we use the fixed rotation approach which will always rotate 90° clockwise around the origin. But the bigger problem in our case is the fact that the edge pixels are often not detected in the correct places due to varying lighting conditions. Figure 42 shows part of a track where many edge pixels were detected, but due to changes in lighting conditions, the border of the rails were detected instead of the ridge.

This could be prevented by changing the filter parameters and the detection function, moving the problem from one place in the track detection algorithm to another. However, intuitively, the general track information can still be derived from the image, even if the edge pixels are misplaced. We need some kind of fuzzy detection scheme, which uses the fact that many edge pixels might be placed almost correctly, but not exactly where they belong.

Instead of just adding to the accumulators for the parameters that correspond to the edge points, we also add lower values to parameters that correspond to neighboring pixels. This approach is closely related to the work in [23]. This approach requires fewer curve candidates to detect the tracks because a curve will also be detected if it matches a parameter only approximately, visualized in Figure 43. If we assume that the white edge pixel belongs to curve C_2, but was detected in the wrong position in the convetional *hough transformation*, it would not increase the accumulator for these parameters, even if the pixel really belongs to the curve C_2.

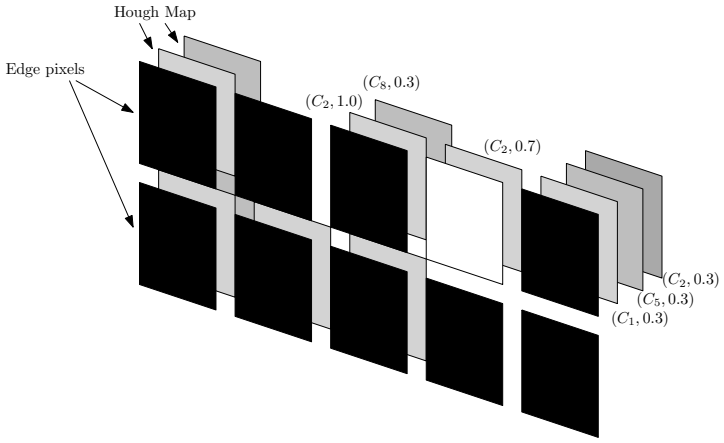


Figure 43: Hough map

We want a misplaced edge pixel to still have an impact on the accumulator of the given parameter. In our case, the white edge pixel will increase the accumulator for the curve C_2 by 0.7. This is implemented as an array of arrays of parameters. Every point in the *hough map* has an underlying array which holds the parameters. Each parameter is augmented by the weight which describes how strongly that cell corresponds to the parameter. The weight is 1.0 if the curve passes exactly through the cell and 0.0 if the cell is not related to the parameter at all. The algorithm works in three steps:

1. *Hough map* initialization.
   All possible curves (parameter sets) are sampled within the region that is covered by the *hough map*. The curves are added to the according cells with their weight

2. *Hough map* evaluation
   For every edge pixel in the edge image, the parameters belonging to the *hough map* cell are incremented according to the parameter weight

3. Parameter selection
   The list of parameters is sorted by their accumulator and the parameter with the highest accumulator value is chosen

The first step is the computationally most expensive one, but has to be done only once, unless the parameters or the *hough map* size changes. For every point (x_o, y_o) in the sampled curves, we add the parameter in a $N \times N$ neighborhood to the *hough map*. The weights are calculated from a 2D Gaussian distribution with $\mu = 0$ and a certain σ

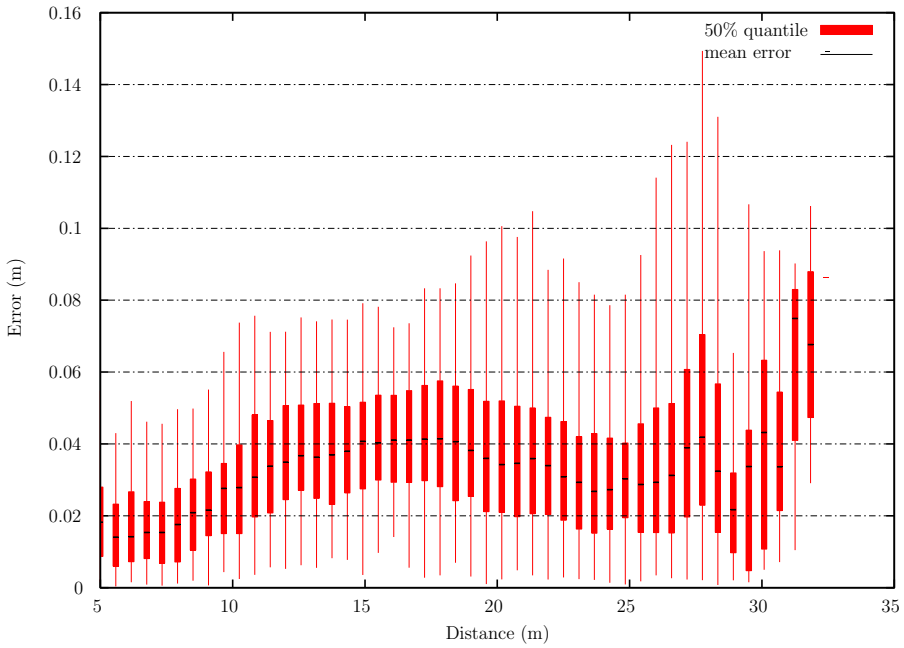
$$weight(x,y) = \begin{cases} |x - x_0| < \epsilon, & \frac{1}{2\pi\sigma^2} \cdot e^{-\left(\frac{(x-x_o)^2+(y-y_o)^2}{2\cdot\sigma^2}\right)} \\ otherwise, & 0 \end{cases} \tag{5.20}$$

In our tests good results were achieved with $\epsilon = 9$ and $\sigma = 1.0$. Steps two and three are executed for every edge image. For every pixel, we calculate the corresponding position of the hough map cell $(x_H, y_H) = \left(x \cdot \frac{w_H}{w_I}, y \cdot \frac{h_H}{h_I}\right)$, iterate over the parameters in the cell and increase their accumulators. After all edge pixel have been processed, each parameter has a certain accumulator value. This value has to be normalized by the total number of cells the curve occupies in the *hough map*. If the curve has no self intersections, the normalization factor could also be approximated with the total curve length instead of the number of cells covered. The parameter with the highest value has the strongest support by the edge image and is thus the detected track curve. Before the next edge image is processed, the accumulators are reset. The algorithm is described in pseudo-code in Algorithm A.3. If we do not normalize the values, shorter curves, which cover less cells, would be ranked down even if they are a much better fit than longer curves.

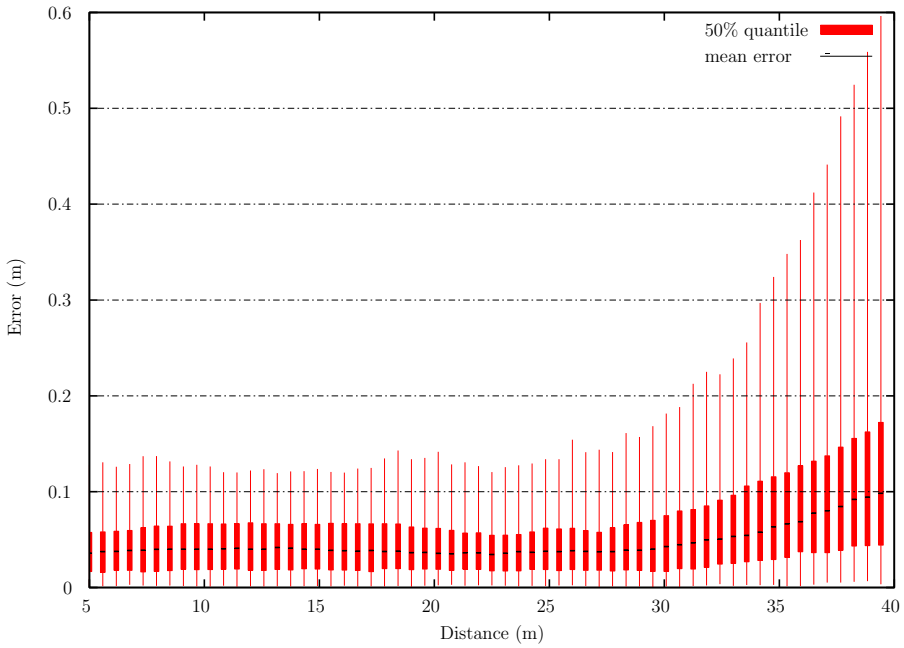
5.11 RESULTS

For our tests we used the *Basler scout scA1300-32gm* gray-scale camera mounted inside the driver's cabin with a vacuum cup. To evaluate the quality of the track detection, we manually marked the left and the right rail in 377 images to generate a ground truth. A centerline between the left and right rail was interpolated to make the data comparable to the output of the detectors. For all frames in the test sets, the horizontal difference (w.r.t to the train coordinate system) between the detected centerline and the ground truth centerline is calculated. These values are converted from pixels to meters and called the *detection error.* Giving a single value to specify the quality of the detection is not sufficient because the *detection error* usually increases with the distance from the train. Instead, we plot the *detection error* over the distance from the train.

The first scenario is a winter scene with snow coverage on the railway and the surroundings. The train moves through a right turn and a car is crossing the tracks in the middle of the curve. Four frames of this scenario are shown in Figure 70. The upper most images show the view of the camera with an overlay of the detected track, using the track detection with polynomial models from Section 5.9.3. The middle images show the IPM view of the *region of interest* in front of the train. The ROI in our tests is always *four* meters to the left, *four* meters to the right, starts *five* meters in front of the train and extends 40 meters in front of the train. The bottom-most images show the outputs of the local maxima detection from Section 5.5 after applying the track mask from Section 5.4. The track mask in Figure 70a is not applied because it is the first image in the scenario without any previous detected tracks. Figure 44a shows the detection error using the polynomial models.

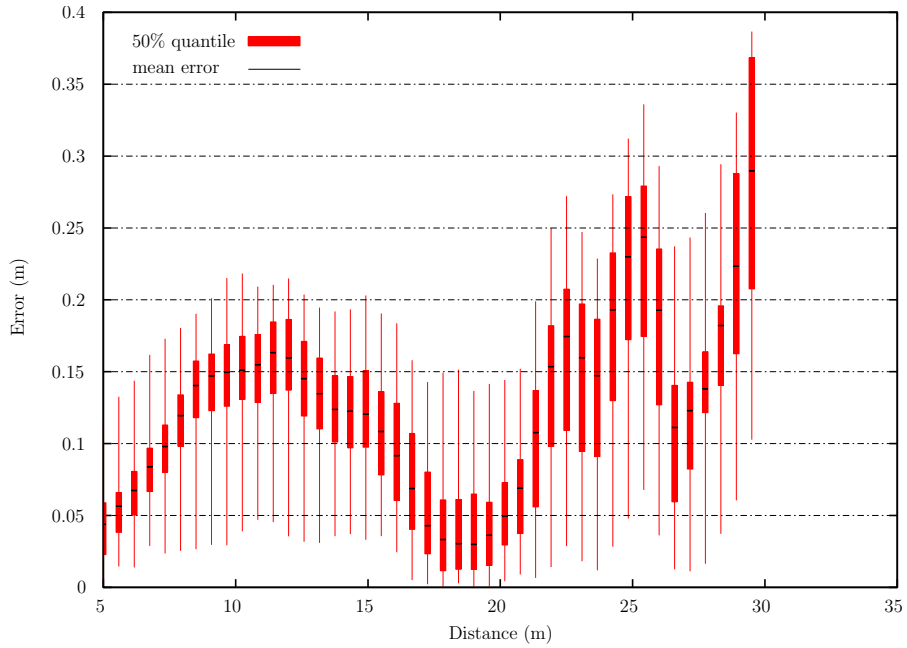


(a) *Winter* scene

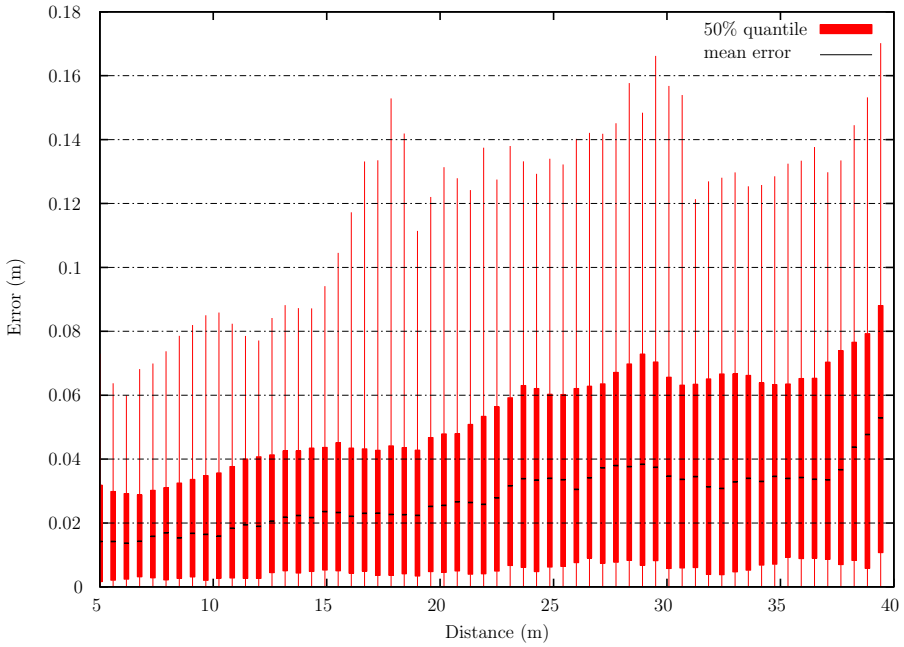


(b) *Problematic trees* scene

Figure 44: Detector error with polynomial model



(a) *Winter* scene with artificially degraded detection result by using the *Problematic trees* track base



(b) *Problematic trees* scene

Figure 45: Detector error in *Hough-map* detector

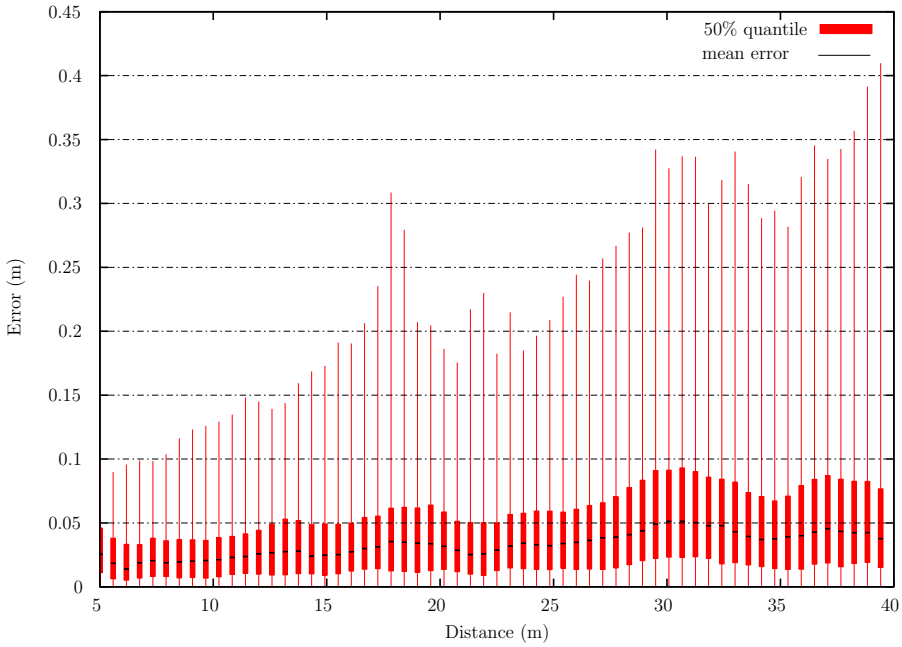


Figure 46: *Problematic trees* scene with heavily reduced track base

Even at a distance of 32 meters 75% of the detections are less than nine centimeters away from the correct track and 97.5% (represented by the whiskers of the plot) of the detections are closer than 15 centimeters.

In the second scenario (*Problematic trees* scene, Figure 69), the train drives along a forest, past two train stations and over a small bridge which amounts to about 1.5 kilometers (the whole track has a length of 14.9 kilometers). The weather is cloudy with slight rain. The local maxima images in Figures 69a-69d are shown without a track mask. This is only included for completeness. The detector itself applies the mask like in the *Winter* scene. This is one of the more challenging scenarios because the trees around the track are very strong track candidates after the IPM conversion. This is especially noticeable in Figure 69d. However, Figure 44b shows that 75% percent of the detections up to a distance of 40 meters are still closer than 18 centimeters to the correct track, if we look at the 97.5% quantile, which includes everything except for complete *mis-detections*. The detected track is at its maximum 0.6 meters away from the correct track at a distance of 40 meters from the train.

For the *hough map* approach in Section 5.10, the ground truth from the *Problematic trees* scene is used to initialize the *hough map* which consist of 377 curve candidates. Many of these candidates are almost equal because the trains drives along several straight track segments. As input to the *hough map* detector, we use the local maxima images without applying a track mask and with a *VV* filter (Section 5.5). Sample input to the detector is shown in Figures 69a-69d in the bottom most images. It is clearly visible that the detector has to cope with many incorrect edges. Figure 45b shows that the *hough map*

detector outperforms the polynomial model detection in Figure 44b. Even at a distance of 40 meters, 75% of the detections are closer than 10 centimeters to the correct track, with 97.5% being closer than 17 centimeters. This result is a bit skewed because the tracks used to initialize the *hough map* are from the ground truth of those images. The detector still needs to determine the correct track candidate for the edge-images, but it has an "unfair" advantage because it can theoretically find the exact track for the corresponding image. To prove that the *hough map* approach also works with a generic track data set, we initialize the *hough map* with a small subset of the ground truth from the *Problematic trees* scene. The reduced dataset consists of every fourth track candidate from the ground truth, which means that 75% of the candidates are omitted. In this case it is obvious that the error must increase because the *hough map* can only output track candidates that are known prior to the creation of the map. But it still should be able to find the best matching candidate to a given dataset instead of returning random results. Figure 46 shows that the overall detection is still very good since 75% of the detected tracks are closer than 10 centimeters to the original track, despite the fact that 75% of the ground truth data have been omitted. Furthermore, 97.5% of the detected tracks are closer than 45 centimeters to the correct track, even at a distance of 40 meters.

Finally, the *hough map* based on the full ground truth of the *Problematic trees* scene is used on the *Winter* scene data. Figure 45a shows that, even though the data contains no track candidates that are a perfect fit for the data, still 97.5% of the detected tracks are closer than 40 centimeters to the correct track at a distance of 30 meters.

These results show that the polynomial model detector provides a good overall solution for our problem, but it can considerably misdetect tracks, if the conditions are bad enough. In comparison, the *hough map* detector provides very robust results, but needs a good track base that represents the whole spectrum of possible tracks in front of the train. To improve the *hough map* results, one could extract the possible track candidates from a map of the whole railway and make sure that a diverse set of track candidates is used to initialize the map.

Part IV

SENSOR SIMULATION

6 SENSOR SIMULATION

6.1 MOTIVATION

To get the approval of the Austrian railway authority for an autonomous train, numerous requirements have to be met. One obvious requirement is to prove that the algorithms work in every possible scenario. This is, however, a task that can not be achieved in a finite time span. If we take only a very simple scenario, e.g. "a person is on the tracks", we would need to vary the size, position and posture of the person as well as the speed of the train. If we have three setups for the size, the position and the posture and five different train speeds, we would need to test 135 scenarios. Assuming that we have to do each test five times (which is an arbitrarily chosen low single digit number) to get a (somewhat) reliable result, this would already mean 675 tests. Since the system has to cope with sensor defects, it needs to be done for combinations of the available sensors simulating sensor failures. So even for this simple test, which does not even include vehicles, other trains, several people, different weather conditions etc., there would already be the need for several thousand tests. And even if all those tests would be feasible, any changes to the system that need recertification would require substantial time and result in non-neglectable costs to do all these tests again.

In software development, there is a standard solution to this problem, namely unit tests. If one can prove that a low-level system works flawless within given bounds, tests of higher level systems can be based on the assumption that the lower level systems are without errors. If changes to the low-level systems are made and they still pass these tests the whole system is assumed to be working correctly.

To transform this approach to our scenarios, we need a way to test the low-level object detection algorithms individually. Normally, we would test the whole system as mentioned above, but then we could not argue about the correctness of an individual component. If a component is changed, the whole system has to be re-evaluated to prevent unexpected side effects. Doing a high-level test of the system means checking whether the vehicle operates as expected in certain situations. There would only be few variables to check, but a huge amount of tests to gain a certain degree of confidence that the system works correctly is required.

Testing only a single component however requires much more knowledge about the data that is provided to the system. For example, an object segmentation algorithm that works on data from a LIDAR will

give each laser-return an ID to which object it belongs. To verify the correctness of such an algorithm, one needs to know this information. This could be achieved by manually labeling each point in the data which is virtually impossible for regular scenarios (i.e. a Velodyne HDL-64E S2 outputs $\sim$ 100.000 points per rotation). Another way is to create virtual data and simulate the output of the sensors. Given such a system, one could create an equivalent to unit tests in software engineering for point-cloud algorithms.

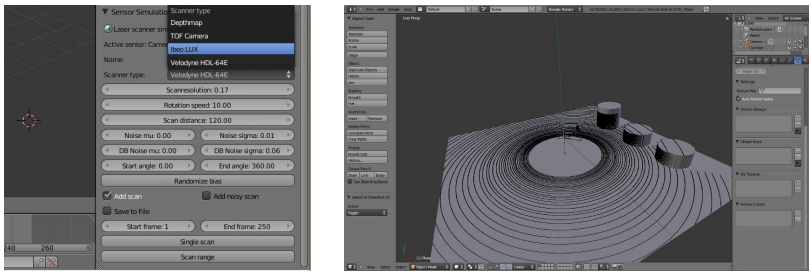
The simulator developed during this project enables us to build a ground truth that can be used to verify the correctness of algorithms on a per measurement (per point in a point-cloud) basis. Since most of the currently available algorithms are tested on recorded (usually not publicly available) sensor data and algorithmic evaluations rely on visual inspection of the results, this simulator will enable a better comparative analysis. In addition, it can be used to do an exact quantitative and qualitative analysis of the algorithm output.

Some authors tackle that problem by implementing their own sensor simulations, but most *home-brewed* approaches follow unrealistic simplifications, like using subdivision methods to generate point clouds.

The software we developed represents an approach to tackle that shortcoming: we provide a unified simulation and modeling environment which is capable of simulating several different types of sensors, carefully considering their special (physical) properties. This is achieved by integrating the simulation tool directly into *Blender*[1], a 3-D content creation suite. With this combination it is possible to model the test scenarios with arbitrary level of detail and immediately simulate the sensor output directly within the modeling environment. The Blender Sensor Simulation (BlenSor) [2] toolkit is completely integrated within *Blender* (see Fig. 47a) and does not require any custom scripts or tedious editing of configuration files to adjust the sensors. Yet, it is possible to access the underlying scanning functionality from custom code in case researchers want to modify the core functionality.

The strong focus on offline data creation for algorithm development and testing allows BlenSor to focus on usability and features. BlenSor does not need to satisfy any external dependencies required to enable compatibility with robotics frameworks for instance. The output is either i) written to a file (in a format explained in Section 6.3.7) or ii) added as a mesh within the sensor simulation. This facilitates direct interaction with the simulated (i.e *scanned*) data. Even though real-time capabilities have been left out on purpose, the simulation can be used together with *Blender's* physic engine, thus enabling to simulate complex scenarios with physical interaction of objects.

1 http://www.blender.org
2 http://www.blensor.org



(a) Every sensor has different parameters, which can easily be modified and which are stored in a *blend* file

(b) Example of a simple scan simulation. Single scans can be directly viewed and manipulated (and even analyzed) within Blender.

Figure 47: The sensor simulation interface is a part of the *Blender* GUI. It can be used just like any other feature of Blender.

6.2 SIMULATION

Compared to robot simulation software ([9, 55]), BlenSor focuses on the simulation of the sensors themselves rather than the interaction of sensor-equipped robots with the environment. In fact, we are able to care a lot more about specific sensor properties, since there are no real-time constraints. Such properties are for example a realistic noise model, physical effects like reflection, refraction, reflectivity and sophisticated casting of rays that do not just describe a circle around the scanning center. The simulation accuracy can be increased with simple changes to the sensor code if features that are not yet available are required. The implementation details of the various sensor types in the following sections describe the simulation state at the time of writing. Due to the strong focus on offline simulation, we are able to simulate scenarios with a higher degree of detail than what is currently possible with existing robot simulators (e.g. MORSE ([9]).

6.3 SCANNING PRINCIPLE

Some sensors simulated by BlenSor basically rely on the fact that the speed of light is finite and that light is at least partially reflected from most surfaces. To be more specific, the measured reflection is affected by i) the traveling distance of the emitted light, ii) the amount of light arriving at the sensor and iii) the precise measurement time. In general, one or more rays of light are emitted from a range measurement device in the form of a light pulse. The rays travel along straight lines to a potential object. Once the rays hit an object, a fraction of the light gets reflected back to the sensor, some part gets reflected in different directions, and another part may pass through the object (in the case of transparent materials) in a possibly different direction.

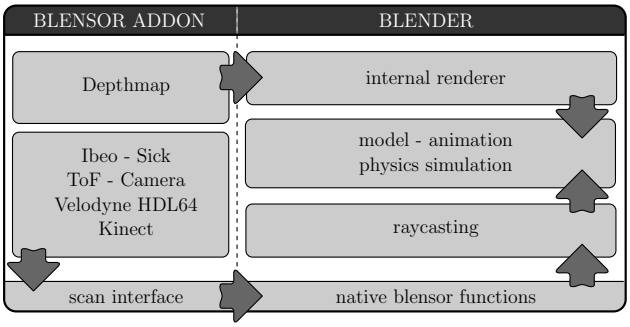


Figure 48: Interaction between BlenSor and Blender

This is in fact closely related to ray-tracing techniques in computer graphics. Thus the modification of a ray-tracing program to match the sensor characteristics seems just natural. Although *Blender* provides an interface to cast rays from within the Python programming language, the functionality is limited and runtime performance inevitably suffers due to the computational demand to simulate a huge number of laser rays. BlenSor tackles this problem by patching the *Blender* code-base, shown in Figure 48, to provide an interface that can cast an arbitrary number of rays *simultaneously*. It also allows Python code to access material properties of the faces that are hit by the rays. For increased efficiency, reflection is handled directly within Blender. By using this interface, the sensors developed using the Python interface can set up an array of ray directions and hand the actual ray-casting over to the patched *Blender* core. Then, a ray-tree is built by *Blender* to allow efficient ray-casting. This modification processes all rays (and calculates reflections if needed) and returns the distances of the hits as well as the *objectID* for each ray. Eventually, the sensor code calculates sensor dependent noise and other physical features. This is described in the following sections.

6.3.1 *Rotating LIDAR*

A rotating LIDAR has a sensor/emitter unit rotating around the center of gravity and thus creates a 360° scan of the environment. As a representative of this class of sensor type, BlenSor implements a Velodyne HDL-64E S2 scanner. This sensor can detect objects with a (diffuse) reflectivity of 10% ($= r_{lower}$) at a distance of 50 meter ($= d_{lower}$) and objects with a (diffuse) reflectivity of 80% ($= r_{upper}$) at a distance of 120 meter ($= d_{upper}$). As already mentioned, the amount of light reflected back to the sensor depends on the distance of the object. The decrease in reflected light is compensated within the scanner electronics by lowering the threshold during the scan interval. Unfortunately, this process can not be correctly reproduced by BlenSor, since the information about threshold adaption is not available from the manufacturer. It is, however, possible to approximate this process

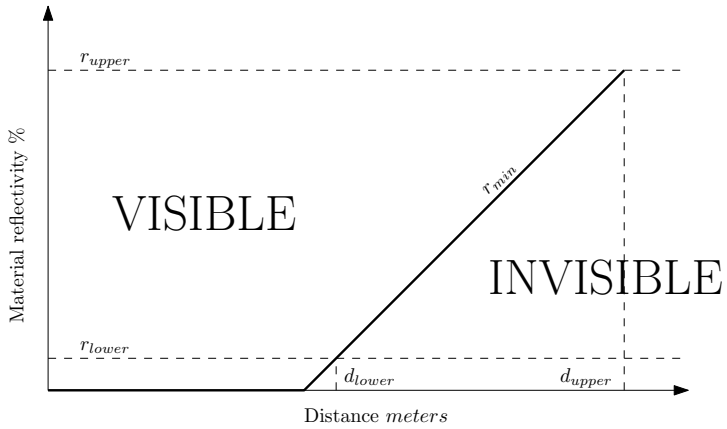


Figure 49: Reflectivity model based on surface distance

by means of linear interpolation of the minimum required reflectivity. We use the 10% and 80% marks listed in the data sheet of the sensor. Objects closer than 50 meters are detected as long as their reflectivity is $> 0\%$. Objects at a distance ($dist$) between 50 and 120 meters are detected if their reflectivity is $\geqq r_{min}(dist)$ (Figure 49), according to Equation (6.1). These values can easily be adapted by the user, if an empiric evaluation of the sensor provides different results than the information from the manufacturer, or if the user wants to simulate a different environment like haze or fog. As this effect is calculated on a per-ray basis, it is even possible that a single object is only partially visible if it has a low reflectivity and is far away from the scanner (cf. Figure 50).

$$r_{min}(dist) = r_{lower} + \frac{(r_{upper} - r_{lower}) \cdot dist}{d_{upper} - d_{lower}} \tag{6.1}$$

Once all rays have been cast, we have to impose sensor-specific errors to the *clean* measurements ($dist_{real}$). Our error model currently consists of two parts. The first part is a distance bias ($noise_{bias}$) for each of the 64 laser units. This bias remains the same in each rotation, but the noise characteristics can be changed by the user. Experiments with a real Velodyne HDL-64E S2 revealed that the reported z-distance of a plane normal to the laser's z-axis may differ up to 30 centimeters for any two laser units (combination of a laser and a detector). This is close to the actual numbers provided in the sensor fact sheets. The second part of our error model accounts for the fact that each single measurement ($dist_{noisy}$) is subject to a certain noise as well. Thus, a per-ray noise ($noise_{ray}$) is applied to the distance measurements. The final (noisy) distance is formally given by

$$dist_{noisy}(yaw, pitch_i) = dist_{real}(yaw, pitch_i) + \epsilon_{bias,i} + \epsilon_{ray} \tag{6.2}$$

with $\epsilon_{bias,i} \sim \mathcal{N}(0, \sigma_{bias})$ and $\epsilon_{ray} \sim \mathcal{N}(0, \sigma_{ray})$, where $\mathcal{N}(\mu, \sigma)$ denotes a Normal distribution with mean μ and variance σ.

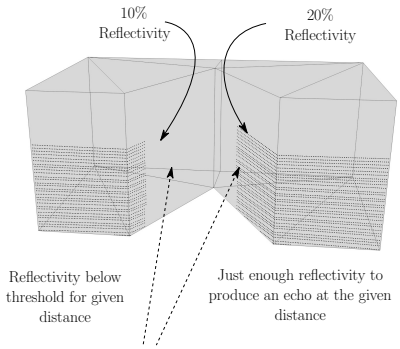


Figure 50: Objects with low reflectivity

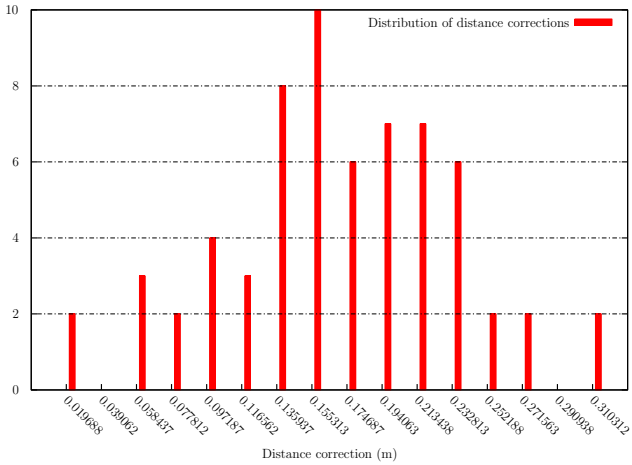


Figure 51: Distribution of distance corrections from the Velodyne configuration

6.3.1.1 *Simulated versus real data*

The parameters for the simulated Velodyne are derived from the calibration data provided by the manufacturer. Figure 51 shows a histogram of distance correction values from the calibration data which are distance correction values for the 64 lasers. Using this information we are able to derive μ and σ for the Normal distribution and use them as the configuration for the simulated scanner. The calibration data leads to $\mu = 0.167422\,meters$ and $\sigma = 0.063108$. To compare the quality of the simulation to real world data, we are scanning a real wall with the HDL-64E S2 as well as a simulated wall with the simulated Velodyne three meters away from the scanner.

The Velodyne scanner used for the comparison is not the same as the one from which the configuration data was extracted to prevent a too strong bias towards the measured data. We can see in Figure 52, which shows the histograms of the errors in the distance measurements to the

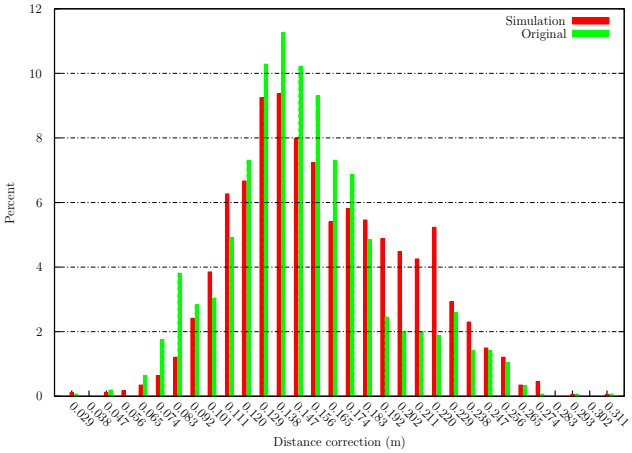


Figure 52: Comparison of distances errors between a real and simulated scan of a wall

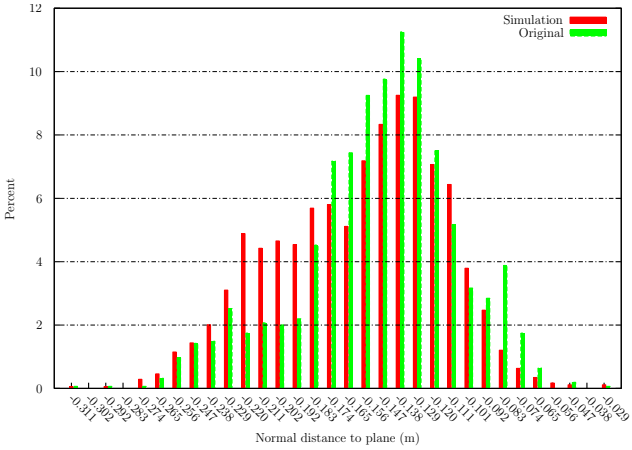


Figure 53: Comparison of normal distances from a real and simulated scan to a wall

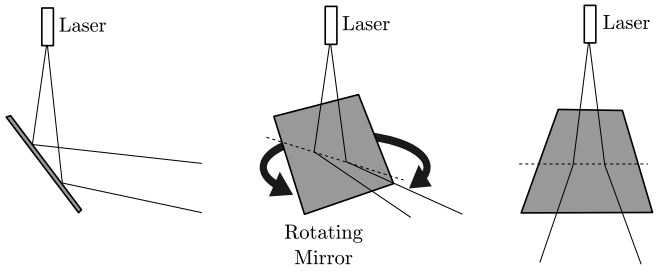
three meter wall, that both scanners have very similar characteristics. This is of course expected since the simulated distance error is applied to the correct measurements and the error values are drawn from a Gaussian distribution. But such a result would also be possible for data that does not look like the scanned surface. We need to compare values that are also dependent on the angle at which the rays hit the surface. We chose the normal distance from the plane to the points for this task. The normal distance of a measurement to the plane depends not only on the depth measurement but also on the angle at which the ray hits the surface. It is apparent from the histogram in Figure 53 that the normal distances of the measurements to the plane are very similar in the simulated and the real scan. This, in combination with the comparison of the distances errors, suggests that the simulated scans have a characteristic very similar to the real scans.

6.3.2 *Line LIDAR*

Representative for the Line LIDAR type sensors, BlenSor implements a hybrid scanner that can be best described as a combination of an *Ibeo LUX* and a *SICK LMS* sensor with a few modifications. According to the fact sheet of the Ibeo LUX sensor, it can detect obstacles with a (diffuse) reflectivity of 10% up to 50 meters and has an average scanning distance of about 200 meters.

The basic principle of measuring distances is described in Section 6.3.1. A Line BlenSor, however, implements a slightly different method to direct the rays. In contrast to the Velodyne HDL-64E S2 scanner, the line scanner has fixed laser emitters which fire at a rotating mirror. Depending on the position angle of the mirror, the rays are reflected in different directions. The measurement itself is the same as in most other laser-based time of flight distance measurement systems. We highlight the fact that the rotating mirror does not only affect the yaw angle of the laser beams, but also the pitch angle.

In its initial position (i.e., the yaw is $0°$) the mirror reflects the rays at the same yaw angle and with the same pitch angle between the rays as they are emitted by the lasers (cf. Figure 54a). When the yaw angle of the mirror is in the range $[0°, 90°]$, the rays have a yaw and pitch angle which is different from the angles when emitted by the lasers (cf. Figure 54b). Finally, when the mirror reaches a yaw angle of $90°$, the pitch angle of all lasers becomes the same. The former pitch angle between the lasers has become the yaw angle between the lasers (cf. Figure 54c). The noise model for the measurements is the same as in Section 6.3.1 due to the same scanning principle.



(a) $\alpha = 0°$ (b) $\alpha \in [0°, 90°)$ (c) $\alpha = 90°$

Figure 54: The pitch and yaw angles of the outgoing rays are affected by the different yaw angle α of the mirror as it rotates. The angles of the rays are unaffected only in the mirror's initial position.

6.3.3 *Time-of-Flight (ToF) Camera*

In contrast to the BlenSor sensors of Sections 6.3.1 and 6.3.2, a Time of Flight (ToF) camera does not need a narrowly focused beam of light for its measurements. Consequently, ToF cameras do not use lasers to emit the light pulse. Instead, the whole scene is illuminated at once and the *Time of Flight* is measured with a special type of imaging sensor. Compared to LIDAR sensors, a ToF camera has the advantage of a substantial increase in resolution, however, at the cost of limited measurement distance. In terms of simulation, a ToF camera does not differ much from the other sensors, though. The sensor has a per-ray noise, but a higher angular resolution. Some ToF cameras use the *phase difference* of a modulated light signal to calculate the distance the light has traveled. This simplifies the detector because it only needs to be able to detect the *phase difference* of a moderately high frequency (i.e. $20MHz$) signal rather than the time a single pulse has traveled. However, this has a big disadvantage. Since the ToF camera does not pulse the light, but sends out a continuous modulated signal, the phase-difference can only be measured within 0 and 2π, even though the distance might be larger than the light can travel within a single wavelength. Thus, the distance to the object is $\frac{c*(\phi+n\cdot 2\pi)}{\omega}$ where c is the speed of light, ϕ is the measured phase distance and ω is the modulation frequency. This may lead to ambiguities in the distance measurements, because, if ω is for example tuned to a distance of 10 meters, a measurement of 15 meters would appear to the camera like a measurement of 5 meters . This effect is called *Back-folding*: objects at a certain distance may appear closer than they really are (cf. Figure 55). *Back-folding* can be enabled in BlenSor which causes all distance measurements in the upper half of the maximum scanning distance to be mapped into the lower half according to the following equation

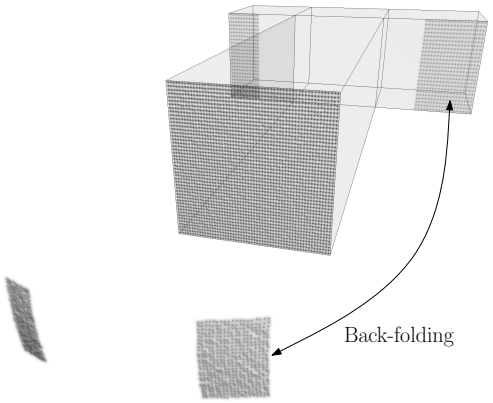


Figure 55: *Back-folding* effect of Time-of-Flight cameras

$$dist_{backfolding} = \begin{cases} dist_{real}, & dist_{real} < \frac{maxdistance}{2} \\ dist_{real} - \frac{maxdistance}{2}, & else. \end{cases} \tag{6.3}$$

6.3.4 *Stereo sensors*

This is a class of sensors that does not measure the time the light travels, which could be seen as the direct way to measure distances, but rather triangulates the distance of a point or object seen from two different viewpoints. The term *"seen from two different viewpoints"* does not necessarily mean that they are passively observed, but that there is an uninterrupted path from both projection centers to the measured point. In fact, several stereo sensors ([30]) use one *active* (projector) and one *passive* component (camera) to perform the triangulation. In this case, the projector emits a well-known pattern onto the scene, which is then detected by the camera. The pattern has to be designed in such a way that a small portion of the pattern is sufficient to locate its position in the complete pattern that is projected. This way, the stereo system knows which rays that are passing through the projection center of the projector correspond to rays through the camera projection center which in turn reproduce the pattern on the image sensor. In combination with the known parameters (baseline, focal length, ...), the 3D point can then be triangulated (in *sensor coordinates*).

Simulating such a sensor with a ToF camera is not sufficient because the ToF camera needs only one path to the measured point, namely the path from the projection center of the camera to the measured point in the scene. They are shown as the strong black lines in the left part of Figure 56. However, a stereo system needs to see the measured point from both projection centers at the same time. If a part of an

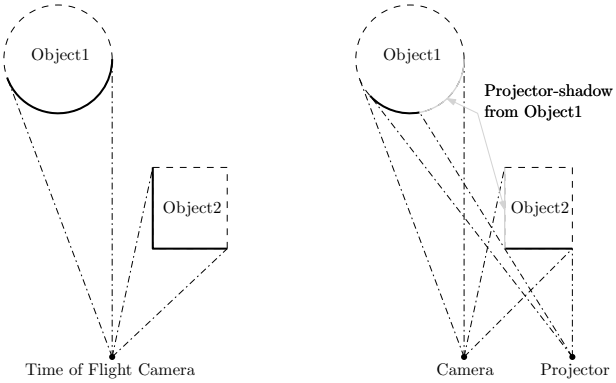


Figure 56: Incomplete depth measurements in stereo setups due to occlusions

object is in the shadow from even one of the two projection centers, no measurement is possible. This is an additional constraint that will reduce the number of valid measurements compared to a Time-of-Flight camera. An example is shown int Figure 56 as gray lines on *Object1* and *Object2*.

6.3.4.1 *Simulating Stereo sensors with ray-tracing*

We could cast rays from both projection centers onto the scene and try to find matching points in the results. This is, however, a very time-consuming task and would also process all points for which clearly no measurement is possible due to occlusions. Our approach does the scanning in a two step process.

1. Cast all rays from the projection center C_p (Figure 57) onto the scene. Some may hit a target, while some may not. Two examples are shown as points P_1 and P_1' in Figure 57.

2. For all valid points that the projector generated, corresponding rays from the camera center C_C are cast onto the scene, shown as r and r' in Figure 57.

3. Those rays that intersect the scene in the same points as the projector rays are valid measurements. In Figure 57, the points P_1 and P_2 represent a valid measurement, whereas P_1' and P_2' are not the same points because r' hits a different surface before the ray reaches the point P_1'.

All corresponding camera/projector rays that meet in the same point are (at least in theory) valid depth measurements. This does not yet take into account the matching window but will correctly simulate holes in the depth-map due to occlusions.

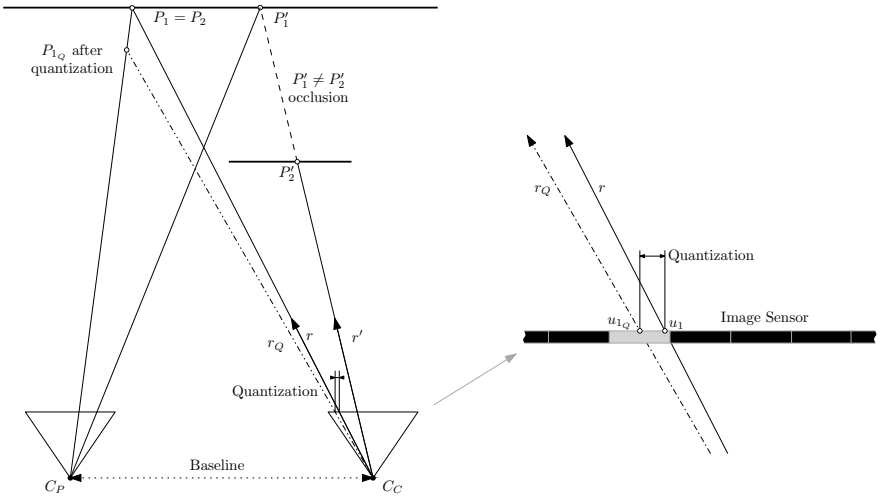


Figure 57: Scanning principle of the virtual stereo sensor

6.3.4.2 *Disparity quantization*

A stereo sensor measures the pixel position of a point either in both sensors or in the sensor and implicitly in the projector (by knowing where in the pattern the point came from). The accuracy of the measurements is limited by the size of a sensor pixel. Even if the detection is accurate to the sub-pixel level there is a dependency between the depth measurement and the sensor properties. The ray r to point P_1 in Figure 57 hits the image sensor in the point u_1. Due to the limited resolution of the sensor, this point can not be expressed in pixel coordinates. The pixel which is illuminated most by the ray r is at position u_{1_Q}. Subsequently, the stereo algorithm detects the point P_1 through the ray r_Q. Instead of detecting the point P_1, the quantization of the image sensor results in a detection of the virtual point P_{1_Q}.

In case of the Microsoft Kinect Sensor this quantization is performed not on a pixel level, but accurate to $\frac{1}{8}$th of a pixel. Even though the detection is done on a sub-pixel level, there is still a substantial quantization of the depth values, which can not be ignored. The quantization of the simulated Kinect sensor is calculated as

$$d_Q = sgn(d)\frac{\lfloor |8 \cdot d| + 0.5 \rfloor}{8} \tag{6.4}$$

where d is the real un-quantized disparity.

6.3.4.3 *Matching window*

Stereo systems are usually built from two cameras (passive stereo) or a camera and a projector (active stereo). Independent of the type of stereo system, the part that calculates the disparity between both viewpoints must find unambiguous matchings between the views ([47]) . A common method to find correspondences in stereo pairs is a block-based matching using for example Sum of Absolute Differences

(SAD) as a metric for similarity between two blocks. Although we do not know exactly how the Kinect sensor works, we can observe the pattern that is projected by the Kinect onto the scene. Using this information and the patent [44], we can deduce that the Kinect sensor uses a pseudo-random pattern of speckles. The information in [12] suggests that the Kinect sensor might use different patterns depending on the distance range that is detected. However the exact algorithm that is used is not available to us.

To simulate the block-based matching between the views and the projected speckle pattern, the simulator post-processes the disparity-map. A pseudo-random speckle-map of size $640x480$ with the same speckle density as projected by a real Kinect sensor is used.

- Only values in the disparity-map that have a corresponding speckle in the speckle-map are considered as a valid measurement.
- For every potential valid measurement a certain amount of speckles within a $9x9$ window around the point must have disparity values that are within a certain threshold to the disparity of the center pixel.
- If enough speckles with valid values are available the disparity of the current speckle is calculated as a weighted average of the valid disparities. The weight depends on the (x, y) distance of the speckles to the center speckle.
- All disparity values that have no valid measurement are filled as follows

$$d(p) = \begin{cases} valid & stored\ value \\ invalid & d(p_i) : \begin{array}{c} \|p_i - p\| < \|p_j - p\| \\ \|p_i - p\| < t \\ p_i, p_j \in stored\ values \end{array} \end{cases} \tag{6.5}$$

  where p is a (x, y) coordinate inside the disparity-map, *valid* means a position in the disparity-map that has a corresponding speckle in the speckle map and t is a threshold distance that is determined empirically.

6.3.5 *Reflection*

A special property of all supported sensor types is the total reflection of rays. If a ray hits a reflecting surface, it does not immediately produce a measurement. Instead, the ray is reflected at the intersection point with the object and may hit another object at a certain distance. The ray might get reflected again, or not hit an object within the

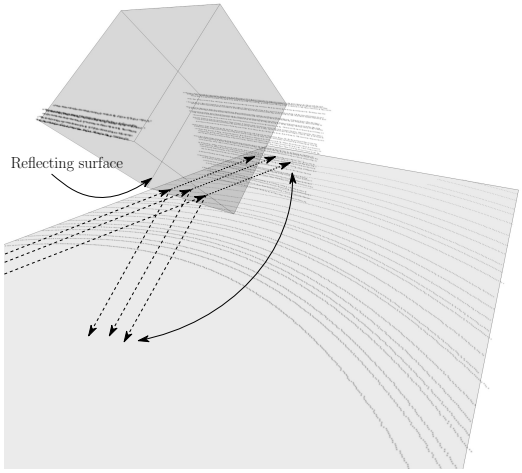


Figure 58: Totally reflecting surfaces cause points to appear farther away

maximum scanning range. Figure 58 illustrates the case when several rays reflected from an object hit another object with a reflectivity above the necessary measurement threshold. As a result, the measured points appear farther away than the object because the rays did actually travel a greater distance. The sensor, however, does not *know* this fact and consequently projects a virtual object behind the real one.

6.3.6 *Color information*

Most laser scanners provide some form of intensity value which may provide information on the surface that was hit. This property is not yet simulated by BlenSor but it can to some degree be emulated by using color information from the model surfaces. Materials can be applied to objects, surfaces or even individual faces. BlenSor uses the same material configurations as Blender but only a subset of the parameters is supported. Supported materials are

- *Color materials* which are defined by the diffuse color parameter. This can be used to color objects, surfaces and faces with a single Red Green Blue (RGB) color.
- *Image texture materials* can be mapped either as UV mapped textures or by automatically mapping them based on generated textures.
- *Procedural texture materials* are all material types that are generated by Blender internally. All procedural texture types from Blender are supported by BlenSor which includes for example simple noise (Figure 59a) based on random values, cloud noise

(a) Random values (b) Cloud noise (c) Voronoi noise

Figure 59: Example noise types in BlenSor

(Figure 59b) which provides a smooth noise and Voronoi noise (Figure 59c).

The color information does also provide the reflectivity of the material. The total energy of the color is used as the reflectivity. A completely black material has a reflectivity of 0%, whereas a completely white material has a reflectivity of 100%. Furthermore, BlenSor can combine all texture types with colored surfaces to non-uniformly manipulate the colored surfaces. This enables the simulation of materials that do not have uniform reflectivity. For example, in Figure 64c the walls and the floor are covered with a cloud material to introduce some non-uniformity in the reflections.

6.3.7 *Ground Truth*

An important advantage of BlenSor is the ease at which the ground truth for the simulated scenes can be generated. BlenSor currently supports four output modes

1. The information about the real distance of a ray and the object identifier of the hit object is stored along with the clean & noisy real world data. Every measurement consist of 15 data fields, the *timestamp* of the measurement, *yaw* and *pitch* angle, the measured *distance*, the *noisy distance*, the x, y and z coordinates of the measured points (i.e. *clean* data), the coordinates of the noisy points, the *objectID* of the object that was hit with the color (r, g, b) of the surface at the hit-point.

2. Output in PCD format which is the file format used by the Pointclouds Library (PCL)[43]. Every scan is written into a separate file which is further separated into a file with *clean* data and a file with *noisy* data. Support for the PCD format makes it very easy to use the generated data with the PCL library and allows for easy algorithm testing.

3. Simulated Kinect data can also be exported as a 16-bit grayscale image which corresponds to the depth-map. The 2^{16} possible values are scaled in such a way that they cover the maximum scanning distance configured in the sensor panel.

4. BlenSor extends the Blender functionality to facilitate exporting of a floating point depth map, rendered at an arbitrary resolution. This depth map can then be used as a ground truth for many algorithms that work on 2.5D data, such as the work of Dolson et al. [8].

6.4 BUILDING A SIMULATION

To build a static or dynamic scene for sensor simulation, we can rely on the standard tools of *Blender*. Any object can be added to the simulation and objects can be imported from other *blend* files. This resembles the situation of a 3-D modeling artist building a scenery. Technically, there is no limit on the level of scene detail (except RAM of course), but too much detail will result in considerable simulation times. Some material properties (for example the *diffuse reflection* parameter) have an impact on the sensor simulation. The materials can be distributed through *blend* files and we already made some available on the BlenSor website. This enables other researchers to reuse the materials in their own simulations. In BlenSor, the cameras are placeholders for the actual sensor devices. Once the scene has been modeled and animated, the user selects a camera that is going to *impersonate* the sensor, adjusts its physical properties and eventually simulates the scanning process. No editing of configuration files or any manipulation of scripts is necessary. The simulation is started and configured directly from the camera settings panel. If the simulation is run in *single scan* mode the user has the option to add the ground truth and/or the noisy real world data to the scene (cf. Figure 47b). This allows for a direct visual verification of the simulation. The scene can easily be adjusted and scanned again. Different scans can coexist in BlenSor, thus allowing a direct comparison of different sensor parameters as well as the scene itself.

6.4.1 *Simulation accuracy versus processing costs*

Sensors like ToF cameras and the Stereo-cameras scan the whole scene at once. If we neglect the time it takes to acquire the information (reflected light) from the scene, the scan happens at a single point in time. Even if objects and the sensor are moving, the simulation of a single scan does not need to handle this explicitly. This is, however, not true for rotating LIDAR scanners which are continuously sending out laser pulses, while the scanner rotates.

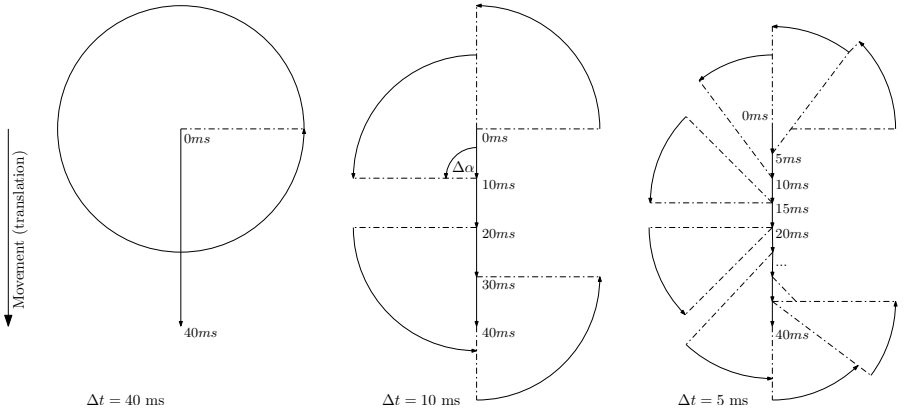


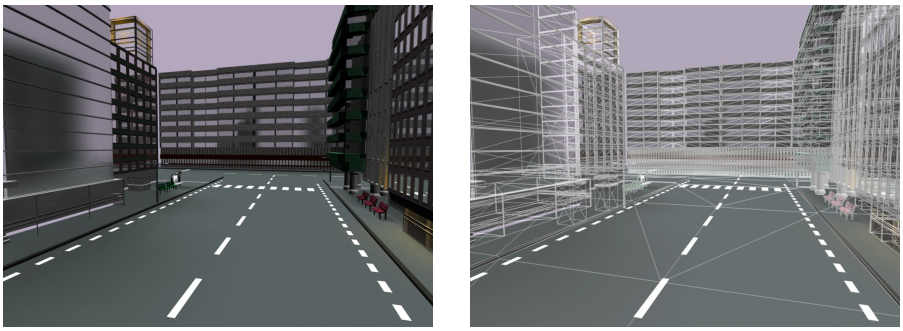
Figure 60: Scan area of a single scan at different simulation intervals

For example, if the scanner rotates with 10 Hz, it takes 0.1 seconds for a single rotation. During this time, the whole scene may change (i.e. moving cars). To capture this behavior correctly, one needs to split up a rotation into smaller parts and update the scene after each of these parts. A regular animation in *Blender* is rendered with 25 frames per second. Thus, a scanner with a rotation speed of 10 Hz would cover $\frac{10}{25}$th of a rotation in a single frame. This is still coarse, but takes into account that a complete rotation does not happen at a single point in time.

These values can be configured arbitrarily based on the requirements of the simulation.

- Setting the frame rate to the same value as the rotation speed of the scanner will result in the maximum simulation speed. A whole rotation will be scanned at once and then the animation is updated, as shown in Figure 60 in the left most image. However, this is a very coarse time resolution which may not be desired.
- Setting the frame rate to $\omega \cdot \frac{2 \cdot \pi}{\phi}$, where ϕ is the angular resolution of the scanner, will result in the best possible simulation. For every scanned angle, the animation is updated. However, this will result in a very long simulation time.

The selection of a useful simulation interval heavily depends on the simulated scenario. If only slowly moving objects or even a completely static scene are simulated, the simulation interval may be extended to cover the whole rotation. Lowering the simulation interval to represent single angle increments of the scanner, however, will almost always be above the practically required precision. For example a useful simulation interval for a scanner mounted on a train that moves with $\sim$ 50km/h and that rotates with $25Hz$ would be $10ms$. The train moves with $13.\bar{8}m/s$ and thus covers a distance of $\frac{13.\bar{8}}{25} = 0.5\bar{5}$ meters within one rotation. At a rotation speed of $25Hz$, a single rotation takes $40ms$, which means that it takes *four* simulation steps to cover a whole



(a) Overview (b) Wire-frame overlay

Figure 61: *Street* scene with 85903 vertices and 164166 faces

rotation at a simulation interval of $10ms$ which is shown in Figure 60 in the center. Between each simulation step the scanner moves $\frac{0.5\bar{5}}{4} = 0.13\bar{8}$ meters, which is also the maximum error introduced by the simulation compared to a real scan. Because the last scanned angle ($\Delta\alpha = \frac{\pi}{2}$ in Figure 60) within a single simulation interval, should originate from a scanner position that is $0.13\bar{8}$ meters away from the position where the first angle ($\Delta\alpha = 0$ in Figure 60) was scanned.

An overview of the relation between simulation intervals and total simulation time is given in Table 3 based on the scene in Figure 61. This scene has 85903 vertices and 164166 faces. The scans have been done on a version of the model that does not have any color information. A colored rendering of the scene is shown in Figure 61a.

The pure simulation time without any output is shown in the first row. The shortest possible simulation time at a rotation speed of $25Hz$ is achieved with a simulation interval of $40ms$, which amounts to a complete rotation per simulation step and requires 6.158 seconds for a rotation. If the output of the scan is also written to a file, a complete rotation requires 7.385 seconds. Using a simulation interval of $1ms$, the simulation time increases to 84.09 seconds with a considerably increased accuracy of only *nine* scanned degrees within a single simulation step. Fortunately, the simulation time does not scale linearly with the inverse simulation interval. Comparing the $40ms$ case with the $10ms$ case in Table 3, we can see that the simulation time roughly doubles, while we achieve a fourfold increase in simulation accuracy. In the $1ms$ case, the simulation interval is only $\frac{1}{40}th$ of the coarse simulation and simulation time increases roughly by a factor of 14. The increase in time is not linear with respect to the increase of the simulation accuracy, because the overhead for each scan (building the ray-casting structures) is independent of the simulation interval. The optimal choice for the simulation interval depends on the required simulation-accuracy-simulation-time tradeoff.

	40 ms Interval	10 ms Interval	1 ms Interval
Scan only	6.158s	11.965s	84.09s
Scan to file	7.385s	13.262s	85.289s

Table 3: Velodyne simulation times of a single rotation of the *Street* scene with 25 Hz

6.4.2 *Using the Physics Engine*

Physics simulation is possible through the internal physics engine of *Blender*. BlenSor can simulate any scene that can also be rendered. In order to simulate physical processes, we just need to set up the physics simulation and record the animation data, while the physics simulation is running. This has the advantage that the physics simulation needs to be run only once, while the actual sensor simulation can be run as many times as necessary without the need to recalculate the physics.

6.4.3 *Exporting Motion Data*

To facilitate quantitative analysis of algorithms, it is necessary to know the exact position and orientation of all (or at least the important) objects in the simulation. The data of the objects can be exported as a text file describing the state of an object over the scan interval. The user may choose between exporting all or only a selection of the objects in the scene. Exporting only selected objects may be beneficial for large and complex scenes. To export only selected objects, the user literally selects one or more objects within Blender and calls the *Export Motion Data* functionality which was added by BlenSor.

6.5 EXPERIMENTAL RESULTS

Our first experimental results in Figure 62a show a crossing scene with four cars. The car closest to the camera is also the position of the sensor. To demonstrate the strength of BlenSor, we use the Velodyne HDL-64E S2 sensor to scan the scene. Figure 62b shows the scene scanned with MORSE, Figure 62c shows the scene scanned with BlenSor. Compared to the BlenSor results, it is clearly visible that MORSE uses only a rudimentary simulation of the sensor. As a matter of fact, this is no real surprise since the primary focus of MORSE is on real-time simulation of whole robots and less on accurate simulation of sensors with all their properties. The BlenSor scan in contrast shows a much denser scan and a noise level similar to what we would expect with a real Velodyne HDL-64E S2 sensor. It is also important to note that the pitch angle of the laser sensors used by Velodyne is not evenly spaced.

Velodyne	Ibeo LUX	Time-of-Flight	Depth-map
8.462 [s]	4.943 [s]	5.290 [s]	11.721 [s]

Table 4: Processing time in seconds of different sensors in a complex scene.

Relying on an exemplary calibration file provided by Velodyne, we distribute the pitch angles accordingly.

In our second experiment, illustrated in Figure 63, we scan a fairly complex scene with 237000 vertices with different sensors. The terrain has been modified by a displacement map to resemble an uneven surface (e.g., an acre). Even though the scene is quite complex, the scanning time for a single simulation interval (in this case 40*ms*) is still between 4.9 and 12.8 seconds (see Table 4 for details). Scanning was done on a Intel Core i5 2.53Ghz machine with 3 GB of RAM running a Linux 2.6.31-14 kernel. The memory usage over the scan is 228 MB.

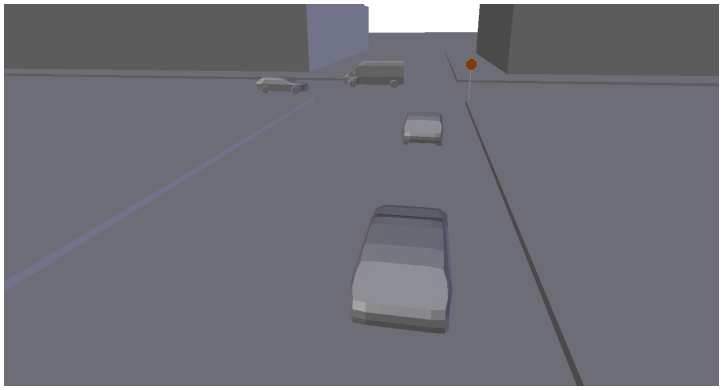
Our third experiment, shown in Figure 64, uses a rough reconstruction of the scene in Figure 64a modeled after the scene used in [10]. A rendered version of the model is shown in Figure 64c. The walls and the floor are augmented with a procedural cloud noise texture to simulate non-uniform reflectivity of surfaces. Using all features described in Section 6.3.4, especially the disparity quantization and the matching window, the simulator creates a virtual Kinect scan (Figure 64d) that has all the properties also present in a real Kinect depth map (Figure 64b). The scan in Figure 64d shows shadows (occlusions) along the guard railing, shorter visibility in certain areas due to different material reflectivity and irregular object boundaries due to depth interpolation and irregularly distributed speckles in the speckle map.

6.5.1 *Reproducibility*

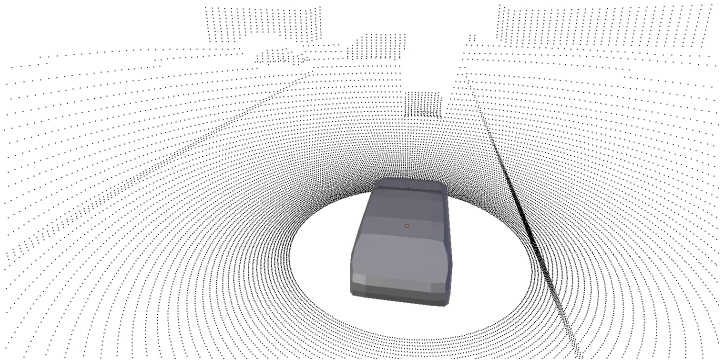
One of the key motivations of developing BlenSor was to allow full reproducibility of research results. BlenSor stores all sensor settings in a *blend* file. Furthermore, the raw scan data can be provided as well in order to allow other researchers to make comparative studies without having to run the simulation again. Nevertheless, storing all needed information in one compact file makes it extremely easy to share the simulation setup. It further enables other researchers to easily modify, adapt or extend the scenarios.

6.5.2 *Scalability*

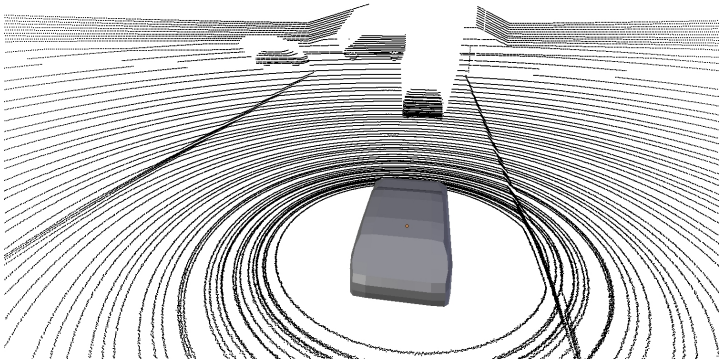
Although sensor simulation is usually a resource intensive task, smaller scenes are rendered almost in real-time by BlenSor. Larger and/or more complex scenes may require substantially more processing time, though. To cope with this problem, BlenSor is designed to allow dis-



(a) Rendered scene

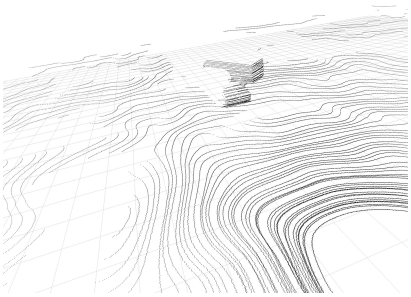


(b) Sim. using MORSE

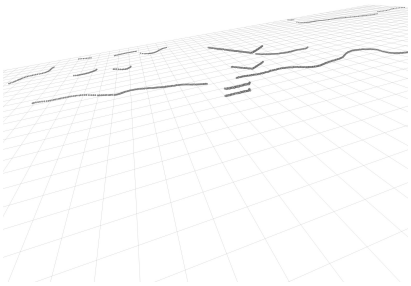


(c) Sim. using BlenSor

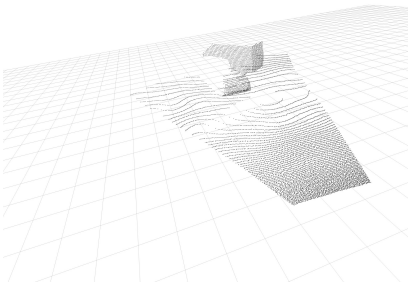
Figure 62: Simulation of a simple scene with MORSE and BlenSor using the implemented Velodyne HDL-64E S2 sensor.



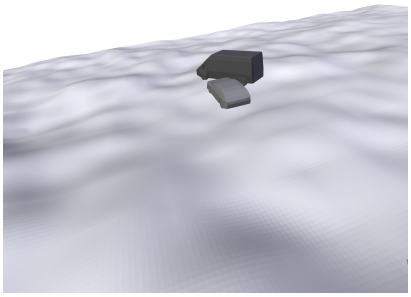
(a) Velodyne scan



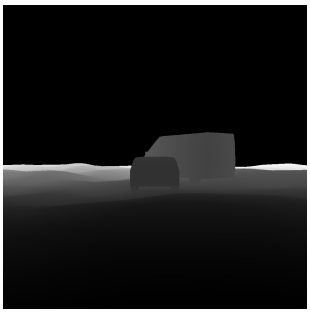
(b) Ibeo scan



(c) ToF camera scan

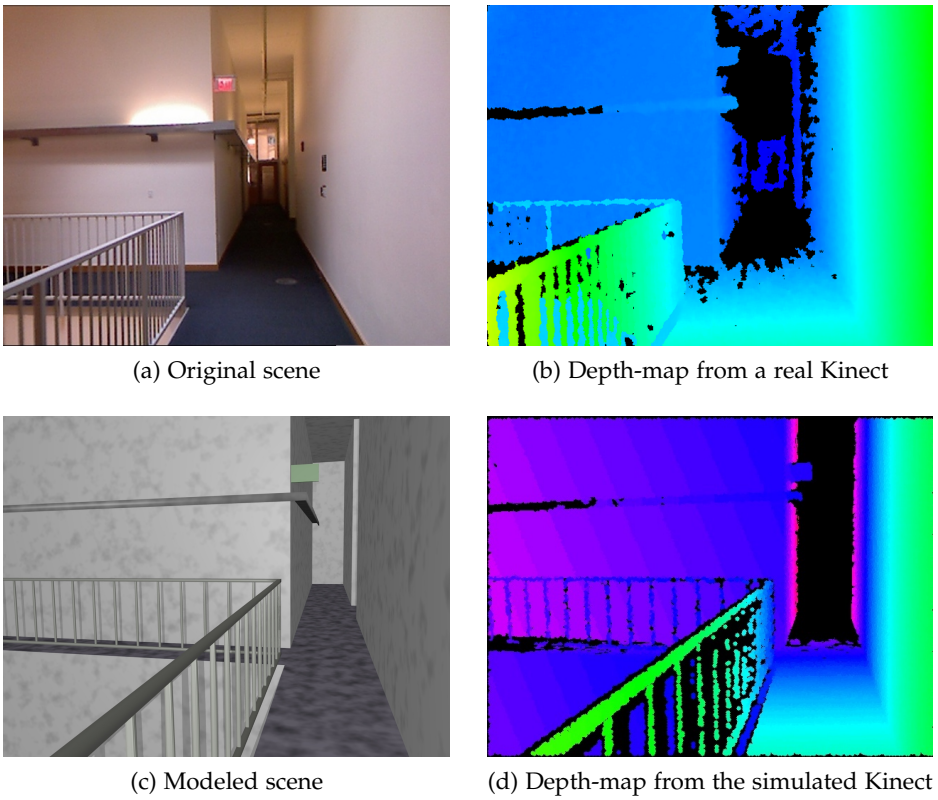


(d) Rendered scene



(e) Ground-truth

Figure 63: Simulation of a scene with a large amount of vertices. The scene consists of a rough terrain, simulating an acre, with a near collision of two cars. The figures in the top row show the simulated sensor output of BlenSor, the figures in the bottom row show the rendered scene (i.e. the *camera view*) as well as the ground truth (i.e. a 2000×2000 high-resolution depth map).



(a) Original scene

(b) Depth-map from a real Kinect

(c) Modeled scene

(d) Depth-map from the simulated Kinect

Figure 64: Simulated Kinect versus real Kinect depth-map

tribution of the *blend* file to multiple hosts by splitting the simulated time interval into corresponding sub-intervals. Since the parts are non-overlapping, each host (or thread) can work on its specific sub-interval. Since we do not make use of the processing power of the Graphics Processing Unit (GPU) (which is usually the case for simulators that rely on a game engine), we can run several instances of the simulation on a multi-core machine at the same time as well.

Part V

CONCLUSION

7 CONCLUSION

In the previous sections we have shown how to calibrate different types of sensors to each other and to the vehicle coordinate system. We have also shown how the information from the sensors can be classified into obstacles and non-obstacles by analyzing the tracks in images. Furthermore, we introduced a sensor simulation that should provide the basis for the analysis of algorithms that are used to detect objects in data provided by 3-D sensors. The calibration techniques are necessary to build a system for fusing the output of the different sensors to a better view of the scene. In this final chapter we will take a look at how the techniques from the previous chapters could be used to improve obstacle detection. These are techniques for which it is fair to assume that they would be successful, but further research is needed to prove their correctness.

7.1 FALSE POSITIVE DETECTION

Even if the track is known very accurately, there are sometimes slight deviations from the correct track in front of the train. This is especially true when the track is calculated based on the current GPS position from a priori knowledge, like a map of the railway. If the position is only slightly off and the track is not just a straight segment, the expected track will differ from the real track in front of the train. If the positioning is solely based on GPS and odometry, this can hardly be avoided since the GPS position is affected for example by trees that occlude the line of sight to the GPS satellites and odometry on rails is affected by the temperature, humidity etc.

Throughout our track, there are several spots that are prone to errors due to mis-localization. Another big problem is the vegetation that is sometimes so close to the track that it almost touches the train. These objects can not be accurately classified into *obstacles* or *non-obstacles*. A train operator will ignore such objects because of additional knowledge that is not derived from the scene itself. The train operator may have remembered that such an *"obstacle"* poses no threat to the train and can safely be ignored.

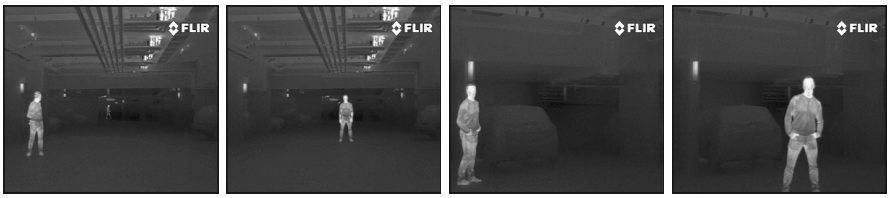
To improve an autonomous system, we have to integrate such outside knowledge. If an object is classified as an obstacle although it is in fact no real *obstacle,* the system should store this information in a database and reclassify it as a *non-obstacle* the next time it is detected.

7.2 DEPTH MAP ESTIMATION

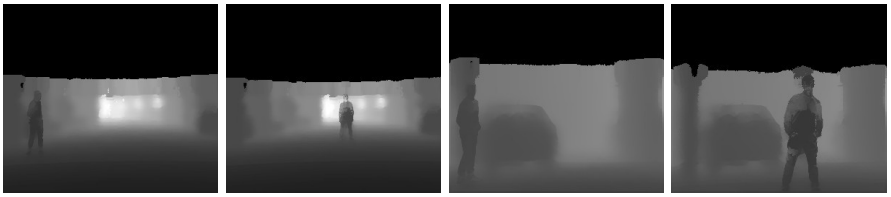
The joint IR camera, visible light camera and LIDAR calibration from Section 3.1.3 provides the basis to combine data from three different sensing technologies. Aside from classifying objects detected in one sensor with corresponding data from other sensors, this can also be used to improve the output of one or several sensors. One use-case is to obtain high resolution depth maps by up-sampling a point cloud using additional information from a visible light camera. Such an approach is described in [8]. Our objective is to demonstrate that by using IR images as an input to the algorithm, we obtain reliable depth information for living objects even in environments with no or bad light conditions. To simulate such a scenario, we choose an underground parking garage where the only source of illumination is an emergency light. In order to use the Bumblebee XB3 camera in this scenario, we would have to adjust exposure time to a maximum, consequently introducing an unacceptable amount of motion blur.

Regarding the parameter setting of the up-sampling algorithm, we build position vectors p_i as (g, u, v) for each IR image pixel where g represents the intensity value of the pixel (and hence encodes the IR information) and u, v represent the pixel location in 2-D. In contrast to the original work, we do not have color information available and further omit the time t in the composition of the position vectors. The extension to a dynamic environment is out of the scope of this work, but is straightforward in the framework of [8]. Obviously, range values outside the IR image plane are discarded. Since the algorithm performs an instance of Gaussian filtering, we have to set the standard deviations for each dimension. As in [8], we perform a grid search to determine the optimal parameters. Figure 65 shows a set of IR images (top row) captured in the parking garage, as well as the corresponding (dense) depth maps (bottom row).

To illustrate the difference in the depth maps when relying on images from the (visible-light) Bumblebee XB3 camera, a low light scenario is shown in Fig. 65. The images were acquired in the same parking garage with only emergency light. In Figure 66b, we can see that the camera picture is almost completely black, except for the emergency light which illuminates a part of the car. In such a scenario, the up-sampling code degenerates to a mere smoothing of the laser range measurements. Consequently, the person standing in front of the wall (see Fig. 66b) is almost invisible. In contrast, the up-sampled data based on the IR camera image (see Fig. 66a) preserves the contours of objects and thus the person clearly stands out from the wall. This system is especially useful for the detection of humans (and living objects in general) due to the fact that they emit a substantial amount of IR radiation.

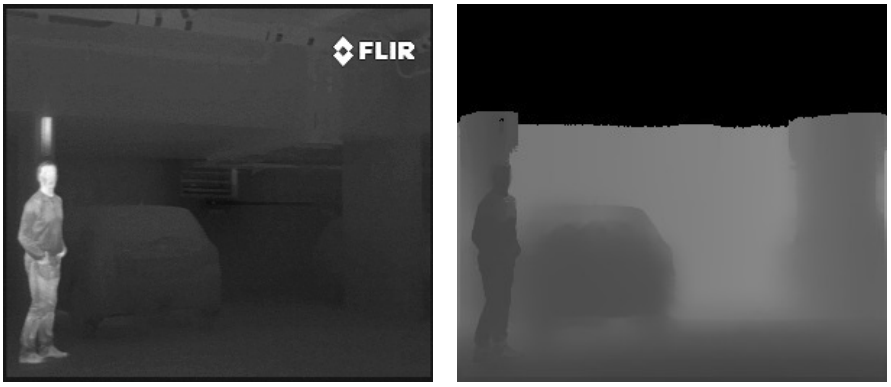


(a) FLIR images. Parking garage with only emergency light

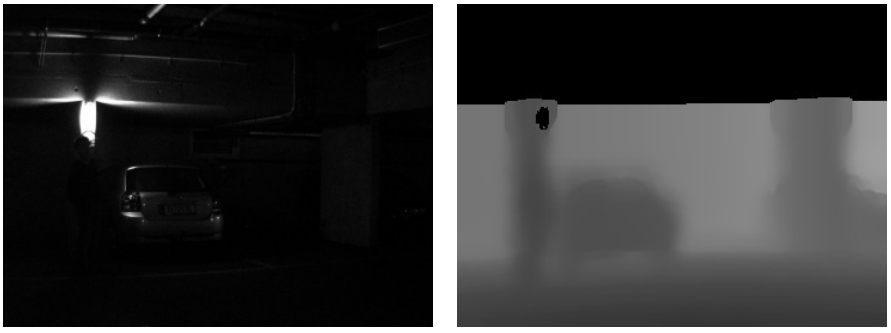


(b) Corresponding (dense) depth maps

Figure 65: FLIR images and corresponding depth maps (calculated using the algorithm of [8]).



(a) using the PathfindIR IR image



(b) using the (visible-light) Bumblebee XB3 image

Figure 66: Depth map comparison of a scene (i.e. parking garage) where the only source of illumination is emergency light (simulates night conditions).

Further research in this field should consider a combination of the approach in [8] with our work by using the combined information of a IR camera and a visible light camera to up-sample the information from the LIDAR. This would not only provide a valid up-sampling in bad lighting conditions, but also use the information from visible light cameras which have usually higher resolutions than IR cameras and may include color information as well.

7.3 SENSOR SIMULATION

To our knowledge BlenSor includes one of the most accurate Kinect simulations at the time of this writing. The simulators for the LIDAR scanners are also very realistic, while still maintaining a high level of usability. However simulation times are still a major aspect worth improving. All calculations inside BlenSor, which includes the ray-casting engine and all depth-map operations, are serialized, despite the fact that most operations are independent of each other. For example, the ray-casting of individual rays does not depend on other rays and could be done in parallel. In fact *Blender* itself uses a master-worker algorithm which calculates the whole image by splitting the rendering region into smaller tiles. These tiles are distributed to several worker threads, which evenly distribute rendering of a single image to all CPU cores.

Splitting the internal operations of BlenSor into smaller parts that can be calculated in several independent threads will provide serious speed improvements and is thus a valuable direction in which BlenSor development should be heading.

7.4 ACKNOWLEDGEMENTS

We want to thank members from the *"Institut für Mess- und Regelungstechnik"* of the *Karlsruhe Institute for Technology* (KIT) for providing Velodyne datasets and Maurice Fallon from the *Computer Science and Artificial Intelligence Laboratory* (CSAIL) of the *Massachusetts Institute of Technology* (MIT) for Kinect datasets.

Part VI

APPENDIX

A

PSEUDO-CODE

Algorithm A.1 Local maxima detection

```
for each slice do
    t_slice ← calculate 99% quantile of the horizontal gradients from
ten lines
end for
for every line in image do
    for every point p in line do
        local_max = false
        for points 15 points to the left of p do
            if value > v(p) then
                local_max ← false
                break
            end if
            if value < v(p) − t_slice then
                local_max ← true
                break
            end if
        end for
        if no local maxima then
            break
        end if
        for points 15 points to the right of p do
            if value > v(p) then
                local_max ← false
                break
            end if
            if value < v(p) − t_slice then
                local_max ← true
                break
            end if
        end for
        if local maxima then
            mark point p
        end if
    end for
end for
```

Algorithm A.2 Building track candidates and selecting the best one

```
Intitial candidate list c is empty
for every slice do
    for every candidate pair in this slice do
        m ← centerline between pair
        for every candidate in c do
            fit curve to points of m and the candidate
            if matchingscore < t_c then
                if candidate size > 2 segments then
                    update candidate
                else
                    add new candidate
                end if
            end if
        end for
        fit curve to points of m and I_zero
        if matchingscore < t_c then
            add m with I_zero to the candidate list
        end if
    end for
end for
if only one candidate has the biggest number of points then
    result ← candidate with biggest pointcount
else
    best_candidates ← candidates with biggest pointcount
    result ← candidate with lowest matching value from
best_candidates
end if
```

Algorithm A.3 Hough-map

▷ Initialisation stage

$H \leftarrow N \times M$ array
for all parameters p **do**
  for sampled points (x_0, y_0) of p **do**
    for $(x, y) \in 9 \times 9$ window around (x_0, y_0) **do**
      $H(x, y, p) \leftarrow max(H(x, y, p), weight(x, y))$
    end for
  end for
end for

▷ Query stage

$P \leftarrow$ array of weights
$I \leftarrow R \times C$ image
for edge edge pixel (x, y) in image **do**
  for all parameters,weights (p, w) in $H(x \cdot \frac{M}{C}, y \cdot \frac{N}{R})$ **do**
    $P(p) \leftarrow P(p) + w$
  end for
end for
for all p in P **do**
  $p \leftarrow \frac{p}{\#\{(x,y):H(x,y,p)>0\}}$
end for
$track \leftarrow p \in P : \forall v \in P, p \geq v$

B

FIGURES

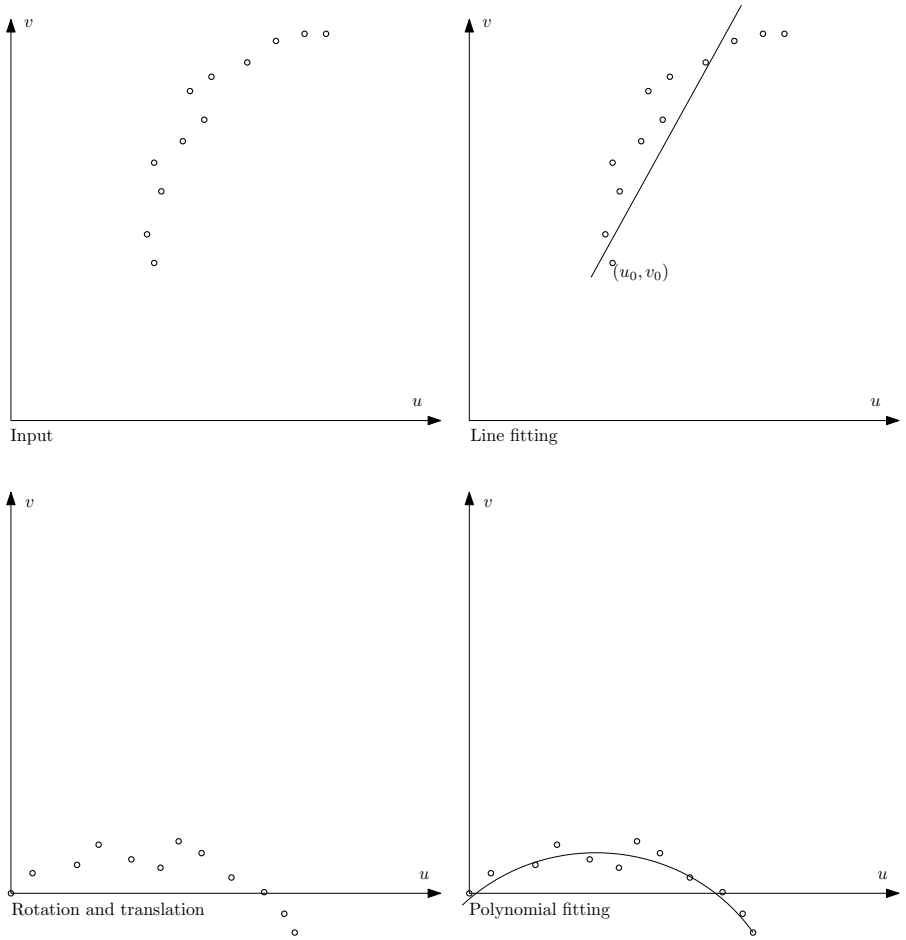


Figure 67: Fitting of a rotated polynomial

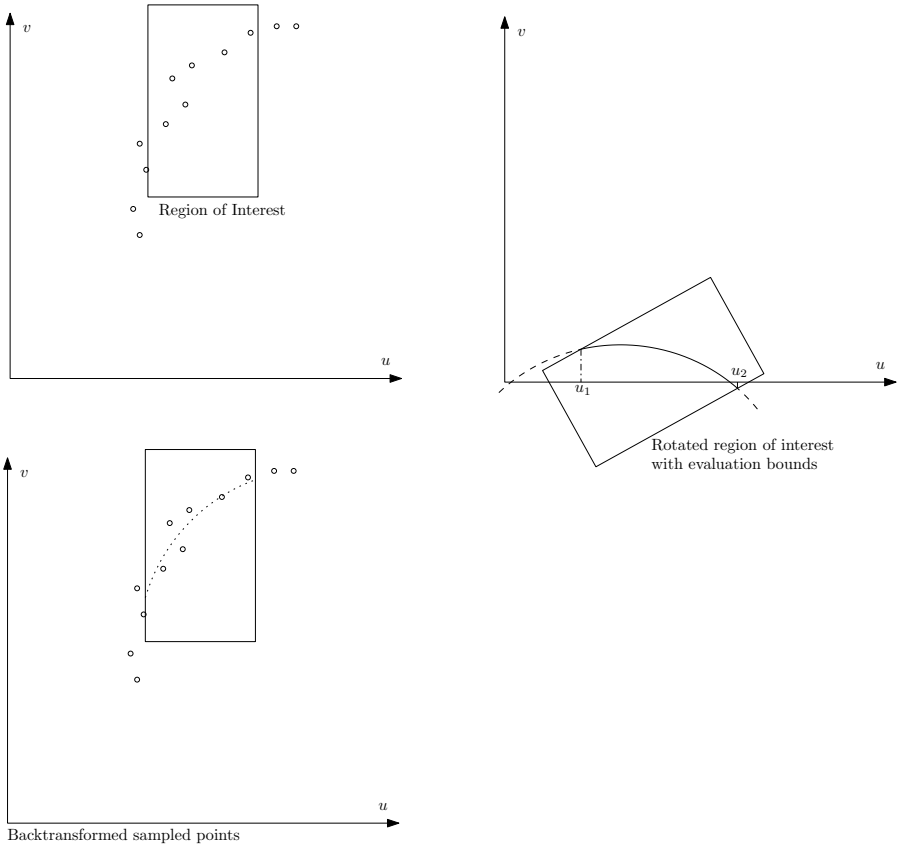
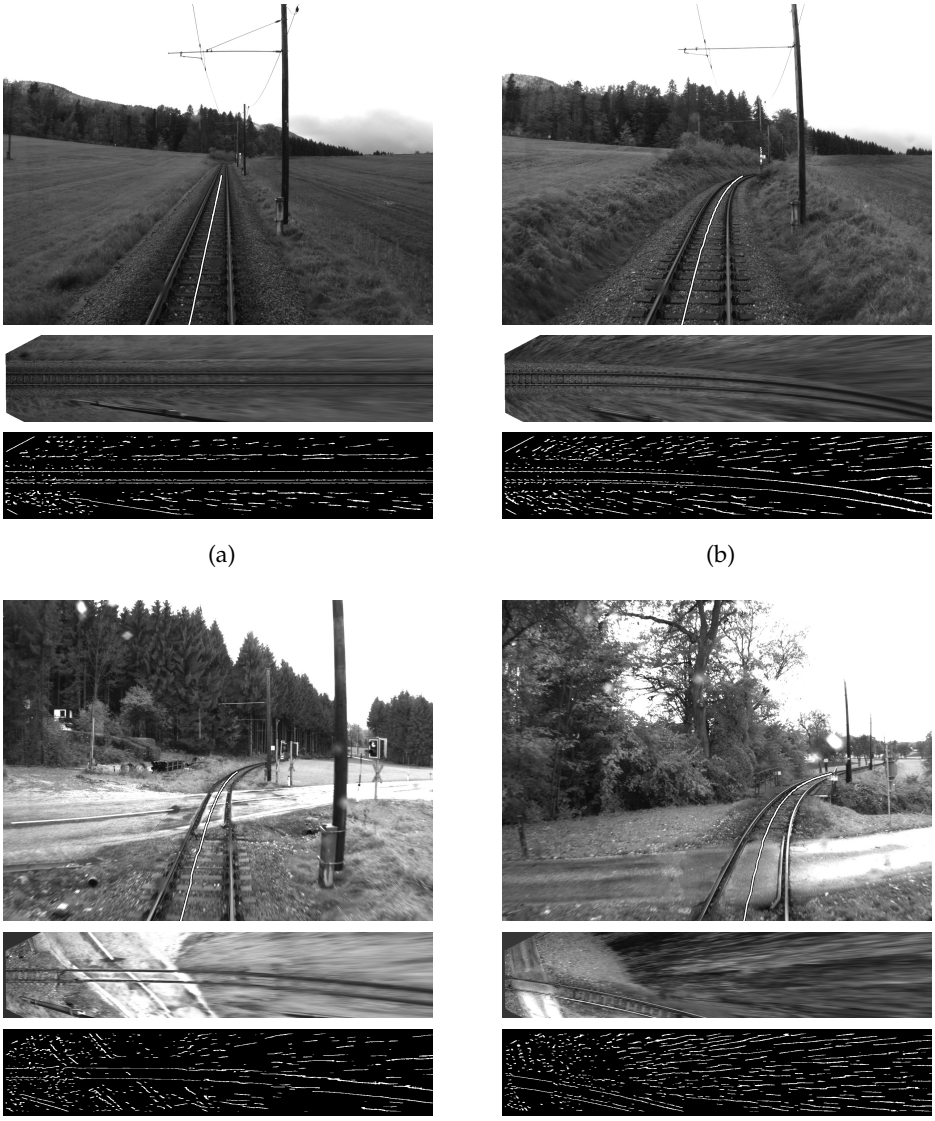


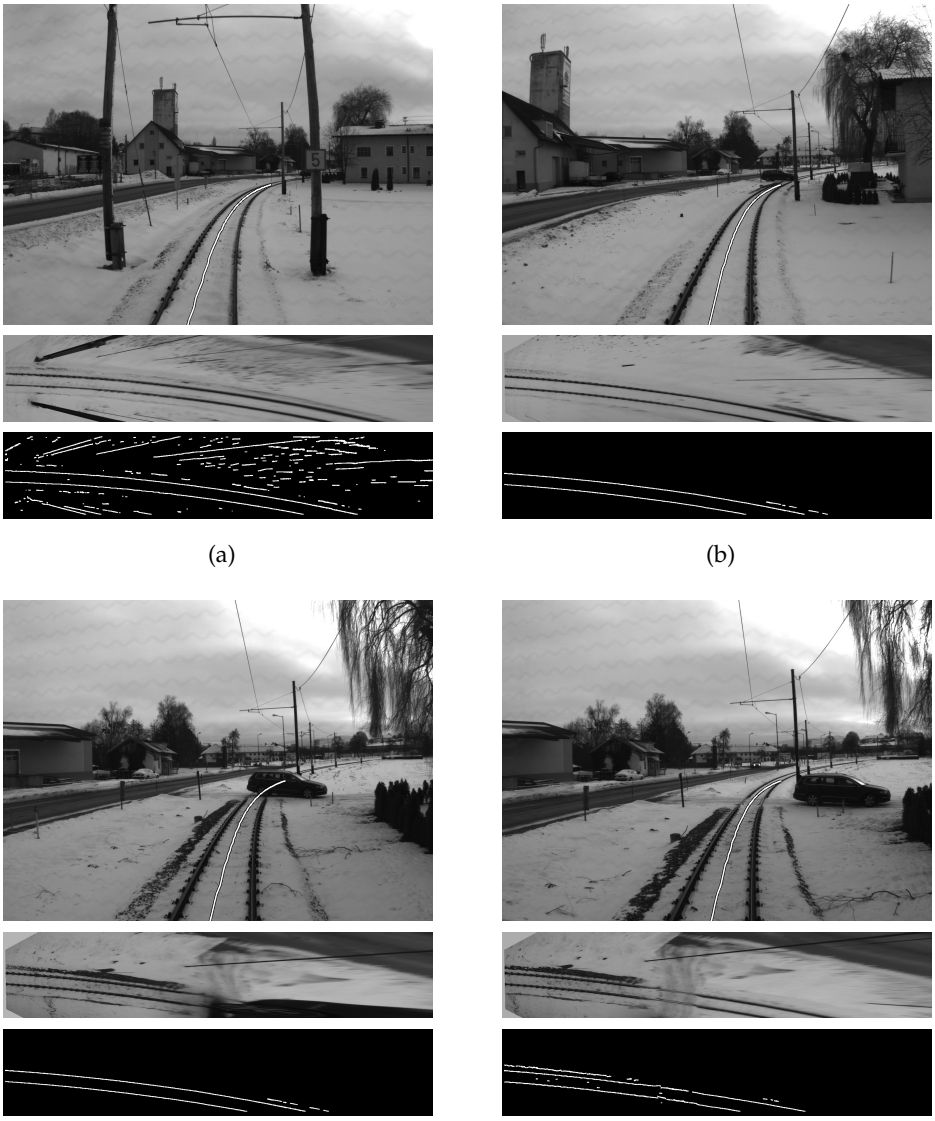
Figure 68: Sampling of the rotated polynomials within a region of interest



(a) (b)

(c) (d)

Figure 69: Examples from the *Problematic trees* scene. The first image is a visualization of the detected track, the second image is the IPM image and the last image is the result of the block based local maxima detection without the track mask.



(a) (b)

(c) (d)

Figure 70: Examples from the *Winter* scene. The first image is a visualization of the detected track, the second image is the IPM image and the last image is the result of the block based local maxima detection after applying the track mask.

BIBLIOGRAPHY

[1] Matthew Antone and Seth Teller. Scalable extrinsic calibration of omni-directional image networks. *Int. J. Comput. Vision*, 49(2-3):143–174, September 2002. ISSN 0920-5691. doi: 10.1023/A:1020141505696. URL http://dx.doi.org/10.1023/A:1020141505696.

[2] Ilya Baran, Jaakko Lehtinen, and Jovan Popović. Sketching clothoid splines using shortest paths. *Computer Graphics Forum*, 29 (2):655–664, 2010. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2009.01635.x. URL http://dx.doi.org/10.1111/j.1467-8659.2009.01635.x.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal*, pages 120–126, November 2000.

[4] Chen Chao, Wang Junzheng, Chang Huayao, and Li Jing. Lane detection of multi-visual-features fusion based on d-s theory. In *Control Conference (CCC), 2011 30th Chinese*, pages 3047 –3052, july 2011.

[5] Yisong Chen, Horace Ip, Zhangjin Huang, and Guoping Wang. Full camera calibration from a single view of planar scene. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Paolo Remagnino, Fatih Porikli, Jörg Peters, James Klosowski, Laura Arns, Yu Chun, Theresa-Marie Rhyne, and Laura Monroe, editors, *Advances in Visual Computing*, volume 5358 of *Lecture Notes in Computer Science*, pages 815–824. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-89638-8. URL http://dx.doi.org/10.1007/978-3-540-89639-5_78.

[6] H.-Y. Cheng, C.-C. Yu, C.-C. Tseng, K.-C. Fan, J.-N. Hwang, and B.-S. Jeng. Environment classification and hierarchical lane detection for structured and unstructured roads. *IET Computer Vision*, 4(1): 37–49, 2010. doi: 10.1049/iet-cvi.2007.0073. URL http://link.aip.org/link/?CVI/4/37/1.

[7] Radu Danescu and Sergiu Nedevschi. Probabilistic lane tracking in difficult road scenarios using stereovision. *Trans. Intell. Transport. Sys.*, 10(2):272–282, 2009. ISSN 1524-9050. doi: http://dx.doi.org/10.1109/TITS.2009.2018328.

[8] J. Dolson, J. Baek, C. Plagemann, and S. Thrun. Upsampling range data in dynamic environments. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*

(CVPR '10), pages 1141–1148, San Francisco, CA, United States, June 2010.

[9] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: Morse. In *International Conference on Robotics and Automation (ICRA)*, pages 46–51. IEEE, 2011. URL http://dblp.uni-trier.de/db/conf/icra/icra2011.html#EcheverriaLDL11.

[10] Maurice F. Fallon, Hordur Johannsson, and John J. Leonard. Efficient scene simulation for robust monte carlo localization using an rgb-d camera. In *IEEE International Conference on Robotics and Automation (ICRA 2012)*, pages 1663–1670. IEEE, 2012.

[11] M.A. Fischler and R.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, June 1981.

[12] Barak Freedman, Alexander Shpunt, and Yoel Arieli. Distance-varying illumination and imaging techniques for depth mapping. Patent Application, 11 2010. URL http://www.patentlens.net/patentlens/patent/US_2010_0290698_A1/en/. US 2010/0290698 A1.

[13] Oliver Gebauer and Wolfgang Pree. Towards autonomously driving trains. http://www.softwareresearch.net/fileadmin/src/docs/publications/C086.pdf, 2008. Workshop for Research on Transportation Cyber-Physical Systems.

[14] Andreas Geiger, Frank Moosmann, Omer Car, and Bernhard Schuster. A toolbox for automatic calibration of range and camera sensors using a single shot. In *International Conference on Robotics and Automation (ICRA)*, St. Paul, USA, May 2012.

[15] C. Glennie and D.D. Lichti. Static calibration and analysis of the Velodyne HDL-64E S2 for high accuracy mobile scanning. *Remote Sensing*, 2(6):1610–1624, June 2010.

[16] M. Gschwandtner, M. Liedlgruber, A. Uhl, and A. Vécsei. Experimental study on the impact of endoscope distortion correction on computer-assisted celiac disease diagnosis. In *Proceedings of the 10th International Conference on Information Technology and Applications in Biomedicine (ITAB'10)*, Corfu, Greece, November 2010.

[17] M. Gschwandtner, R. Kwitt, W. Pree, and A. Uhl. Infrared camera calibration for dense depth map construction. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV '11)*, pages 857–862, Baden-Baden, Germany, June 2011.

[18] M. Gschwandtner, R. Kwitt, W. Pree, and A. Uhl. Blensor: Blender sensor simulation toolbox. In *Proceedings of the 7th international conference on Advances in visual computing - Volume Part II*, volume 6939 of *ISVC'11*, pages 199–208, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24030-0. URL http://dl.acm.org/citation.cfm?id=2045195.2045219.

[19] Michael Gschwandtner, Wolfgang Pree, and Andreas Uhl. Track detection for autonomous trains. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Ronald Chung, Riad Hammound, Muhammad Hussain, Tan Kar-Han, Roger Crawfis, Daniel Thalmann, David Kao, and Lisa Avila, editors, *Advances in Visual Computing*, volume 6455 of *Lecture Notes in Computer Science*, pages 19–28. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-17276-2. URL http://dx.doi.org/10.1007/978-3-642-17277-9_3.

[20] Michael Gschwandtner, Jutta Hämmerle-Uhl, Yvonne Höller, Michael Liedlgruber, Andreas Uhl, and Andreas Vecsei. Improved endoscope distortion correction does not necessarily enhance mucosa-classification based medical decision support systems. In *Proceedings of the IEEE International Workshop on Multimedia Signal Processing (MMSP)*, Banff, Canada, 2012.

[21] E. Guillou, Daniel Meneveaux, Eric Maisel, and Kadi Bouatouch. Using vanishing points for camera calibration and coarse 3d reconstruction from a single image. *The Visual Computer*, 16(7): 396–410, 2000. URL http://dblp.uni-trier.de/db/journals/vc/vc16.html#GuillouMMB00.

[22] Lie Guo, Ping-Shu Ge, Ming-Heng Zhang, Lin-Hui Li, and Yi-Bing Zhao. Pedestrian detection for intelligent transportation systems combining AdaBoost algorithm and support vector machine. *Expert Systems with Applications*, 39(4):4274–4286, March 2012. ISSN 09574174. doi: 10.1016/j.eswa.2011.09.106. URL http://dx.doi.org/10.1016/j.eswa.2011.09.106.

[23] Joon H. Han, László T. Kóczy, and Timothy Poston. Fuzzy hough transform. *Pattern Recogn. Lett.*, 15(7):649–658, July 1994. ISSN 0167-8655. doi: 10.1016/0167-8655(94)90068-X. URL http://dx.doi.org/10.1016/0167-8655(94)90068-X.

[24] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[25] Albert S. Huang and Seth J. Teller. Probabilistic lane estimation using basis curves. In Yoky Matsuoka, Hugh F. Durrant-Whyte, and JosÃ© Neira, editors, *Robotics: Science and Systems*. The MIT Press, 2010. ISBN 978-0-262-51681-5. URL http://dblp.uni-trier.de/db/conf/rss/rss2010.html#HuangT10.

[26] Albert S. Huang, Edwin Olson, and David C. Moore. Lcm: Lightweight communications and marshalling. In *IROS*, pages 4057–4062. IEEE, 2010. ISBN 978-1-4244-6674-0. URL http://dblp.uni-trier.de/db/conf/iros/iros2010.html#HuangOM10.

[27] Fay Huang and Reen-Cheng Wang. Low-level image processing for lane detection and tracking. In *Arts and Technology*, volume 30 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 190 – 197. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-11577-6_24.

[28] Gang Yi Jiang, Tae Younf Choi, Suk Kyo Hong, Jae Wook Bae, and Byung Suk Song. Lane and obstacle detection based on fast inverse perspective mapping algorithm. *IEEE International Conference on Systems, Man and Cybernetics*, 4:2969 – 2974, 2000. ISSN 1062-922X.

[29] Cláudio Rosito Jung and Christian Roberto Kelber. Lane following and lane departure using a linear-parabolic model. *Image Vision Comput.*, 23(13):1192–1202, November 2005. ISSN 0262-8856. doi: 10.1016/j.imavis.2005.07.018. URL http://dx.doi.org/10.1016/j.imavis.2005.07.018.

[30] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012. ISSN 1424-8220. doi: 10.3390/s120201437. URL http://www.mdpi.com/1424-8220/12/2/1437.

[31] King Hann Lim, Kah Phooi Seng, Li-Minn Ang, and Siew Wen Chin. Lane detection and Kalman-based linear-parabolic lane tracking. *Intelligent Human-Machine Systems and Cybernetics, International Conference on*, 2:351–354, 2009. doi: http://doi.ieeecomputersociety.org/10.1109/IHMSC.2009.211.

[32] Christian Lipski, Bjorn Scholz, Kai Berger, Christian Linz, Timo Stich, and Marcus Magnor. A fast and robust approach to lane marking detection and lane tracking. *Image Analysis and Interpretation, IEEE Southwest Symposium on*, 0:57–60, 2008. doi: http://doi.ieeecomputersociety.org/10.1109/SSIAI.2008.4512284.

[33] Frédéric Maire. Vision based anti-collision system for rail track maintenance vehicles. In *AVSS*, pages 170–175. IEEE Computer Society, 2007. URL http://dblp.uni-trier.de/db/conf/avss/avss2007.html#Maire07.

[34] P. Marion, R. Kwitt, B. Davis, and M. Gschwandtner. Pcl and paraview - connecting the dots. In *IEEE CVPR Workshop on Point Cloud Processing (PCP '12)*, pages 80–85, 2012.

[35] James McCrae and Karan Singh. Sketching piecewise clothoid curves. *Computers & Graphics*, 33(4):452–461, 2009.

[36] D. S. Meek and D. J. Walton. An arc spline approximation to a clothoid. *J. Comput. Appl. Math.*, 170:59–77, September 2004. ISSN 0377-0427. doi: 10.1016/j.cam.2003.12.038. URL http://dl.acm.org/citation.cfm?id=1044207.1044210.

[37] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, Anna Petrovskaya, Mike Pflueger, Ganymed Stanek, David Stavens, Antone Vogt, and Sebastian Thrun. Junior: The stanford entry in the urban challenge. *J. Field Robot.*, 25(9):569–597, September 2008. ISSN 1556-4959. doi: 10.1002/rob.v25:9. URL http://dx.doi.org/10.1002/rob.v25:9.

[38] David Moore, Edwin Olson, and Albert Huang. Lightweight communications and marshalling for low-latency interprocess communication. Technical Report MIT-CSAIL-TR-2009-041, MIT, Cambridge, USA, Sep 2009.

[39] Frank Moosmann and Christoph Stiller. Velodyne SLAM. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 393–398, Baden-Baden, Germany, June 2011. URL http://www.mrt.kit.edu/z/publ/download/Moosmann_IV11.pdf.

[40] N. Otsu. A threshold selection method from gray-level histogram. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, January 1979.

[41] Morgan Quigley, Ken Conley, Brian P. 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*, 2009.

[42] Martin Rufli, Davide Scaramuzza, and Roland Siegwart. Automatic detection of checkerboards on blurred and distorted images. In *IROS*, pages 3121–3126. IEEE, 2008.

[43] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[44] Erez Sali and Assaf Avraham. Three-dimensional mapping and imaging. Patent Application, 10 2010. URL http://www.patentlens.net/patentlens/patent/US_2010_0265316_A1/en/. US 2010/0265316 A1.

[45] Joaquim Salvi, Xavier ArmanguÃ©, and Joan Batlle. A comparative review of camera calibrating methods with accuracy evaluation. *Pattern Recognition*, 35(7):1617 – 1635, 2002. ISSN 0031-3203. doi: 10.1016/S0031-3203(01)00126-1. URL http://www.sciencedirect.com/science/article/pii/S0031320301001261.

[46] D. Scaramuzza, A. Martinelli, and R. Siegwart. A toolbox for easily calibrating omnidirectional cameras. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System (IROS '06)*, pages 5695–5701, Beijing, China, October 2006.

[47] Yannic Schröder, Kai Berger, and Marcus Magnor. Super resolution for active light sensor enhancement. Bachelor thesis, March 2012.

[48] Yang Shang, Qifeng Yu, and Xiaohu Zhang. Analytical method for camera calibration from a single image with four coplanar control lines. *Appl. Opt.*, 43(28):5364–5369, Oct 2004. doi: 10.1364/AO.43.005364. URL http://ao.osa.org/abstract.cfm?URI=ao-43-28-5364.

[49] Arne Suppé, Luis E. Navarro-Serment, and Aaron Steinfeld. Semi-autonomous virtual valet parking. In *Proceedings of the 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '10, pages 139–145, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0437-5. doi: 10.1145/1969773.1969798. URL http://doi.acm.org/10.1145/1969773.1969798.

[50] S. Suzuki and K. Be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, April 1985. ISSN 0734189X. doi: 10.1016/0734-189X(85)90016-7. URL http://dx.doi.org/10.1016/0734-189X(85)90016-7.

[51] Sovira Tan, Jason Dale, Andrew Anderson, and Alan Johnston. Inverse perspective mapping and optic flow: A calibration method and a quantitative analysis. *Image and Vision Computing*, 24(2):153 – 165, 2006. ISSN 0262-8856. doi: DOI:10.1016/j.imavis.2005.09.023. URL http://www.sciencedirect.com/science/article/B6V09-4HNSPMD-1/2/94a053eabb06b5bddacc4eaf93278893.

[52] R.Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR '86)*, pages 364–374, Miami Beach, FL, United States, June 1986.

[53] Fraunhofer Institut Verkehrs und Infrastruktursysteme. Hinderniserkennung für Schienenfahrzeuge.

http://www.ivi.fhg.de/frames/german/projects/produktbl/hinderniserkennung.pdf, 2005.

[54] Christopher Urmson, Joshua Anhalt, J. Andrew (Drew) Bagnell, Christopher R. Baker , Robert E Bittner, John M Dolan, David Duggins, David Ferguson, Tugrul Galatali, Hartmut Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Alonzo Kelly, David Kohanbash, Maxim Likhachev, Nick Miller, Kevin Peterson, Ragunathan Rajkumar, Paul Rybski, Bryan Salesky, Sebastian Scherer, Young-Woo Seo, Reid Simmons, Sanjiv Singh, Jarrod M Snider, Anthony (Tony) Stentz, William (Red) L. Whittaker, and Jason Ziglar. Tartan racing: A multi-modal approach to the darpa urban challenge. Technical Report CMU-RI-TR-, Robotics Institute, http://archive.darpa.mil/grandchallenge/, April 2007.

[55] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, December 2008.

[56] Yue Wang, Dinggang Shen, and Eam Khwang Teoh. Lane detection using spline model. *Pattern Recogn. Lett.*, 21(9):677–689, July 2000. ISSN 0167-8655. doi: 10.1016/S0167-8655(00)00021-0. URL http://dx.doi.org/10.1016/S0167-8655(00)00021-0.

[57] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using b-snake. *Image Vision Comput.*, 22(4): 269–280, 2004.

[58] Ji Zhang and Dezhen Song. Error aware monocular visual odometry using vertical line pairs for small robots in urban areas. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*. AAAI Press, 2010.

[59] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11): 1330 – 1334, November 2000.

[60] Yong Zhou, Rong Xu, Xiaofeng Hu, and Qingtai Ye. A robust lane detection and tracking method based on computer vision. *Measurement Science and Technology*, 17(4):736, 2006. URL http://stacks.iop.org/0957-0233/17/i=4/a=020.